Table of Contents

Connected Vision API on the example of the Skeleton module

This document explains the Connected Vision API on the example of the Skeleton module. The workflow (Workflow Example) and all query endpoints (Resources) are generic to all Connected Vision modules.

Skeleton

Chapter 1. Overview

This is the API definition of the Skeleton Module.

1.1. Version information

Version : 0.1

1.2. Contact information

Contact : someone@example.com

1.3. URI scheme

Host : localhost:2020
BasePath : /
Schemes : HTTP

1.4. Tags

- Configuration : Configuration specific things.
- Host : Host specific things.
- Module : Module specific things.
- OutputPin : OutputPin specific things.

Chapter 2. Module Structure

2.1. Configuration

Depending on the module's task or purpose it may be necessary to conduct the same operation based on different parameter sets, using the same parameter set to process different input data or a combination of both. This is the point where configurations come into play. Each parameter set in combination with the input data source is represented by a unique configuration ID.
This configuration ID is used to control (start, stop, etc.) a configuration and also to query its results.

> **TIP** Synonymously you could use the words *"processing task"* or *"batch job"* instead of *"configuration"*, depending on how you'd like to think about it.

2.1.1. Configuration ID

The configuration ID is automatically issued upon the submission of the configuration to the server. The ID is simply the hash of the configuration combined with the input data source specification. Changing any parameter or input source results in a new configuration (ID).
On the other hand, two independently created configurations using the same parameter set and input will result in the same configuration ID providing an inherent duplication protection and performance optimization.

> **TIP** If the same parameters on the same inputs (resulting in the same configuration) were submitted previously, all results that have been already computed are available instantly.

2.1.2. Parameters

The configuration parameters are specified as properties of the configuration's `params` sub-object. The available (and/or required) parameters of a module are specified in the module description as JSON Schema.

The parameters can be of the following types:

- boolean
- integer (signed integer 64-bit)
- number (double 64-bit)
- string
- any (binary object)
- array
- object

2.2. Input

A module can have an arbitrary number of inputs that are connected to outputs of other modules.
The connection points of a module are called input pins and output pins, respectively.
Each input pin is used to request either metadata or binary data. Metadata is structured in JSON format (MIME-Type: application/json). Binary data is required to conform to the specified MIME type. If there is no appropriate MIME type for the data, a meaningful non-registered (e.g. application/x.my-type) MIME type can be used (see unregistered MIME).

The *Skeleton* module, for example, has two input pins:

- Detections: bounding box specified in relative coordinates [0.0, 1.0] with respect to the image dimensions
- PNG-Image: image encoded as PNG

2.3. Output

Similarly to the input, a module can have an arbitrary number of output pins. An output pin provides either metadata or binary data.

The *Skeleton* module, for example, iterates all detections provided by the input pin, tries to query an image with a timestamp equal to or greater than the timestamp of the detection and conducts a computation on this data. In the case of the *Skeleton* module the average color of the pixels within the bounding box is computed. The result of the operation is provided by the Average metadata output pin. The output data comprises the average color specified by an RGB color triplet.

Chapter 3. Workflow Example

3.1. Compose

A configuration is specified in JSON format. A sample configuration for the *Skeleton* module is shown below.

The configuration section for a single module without any input pin, that means, without any preceding module would be rather short. It would only contain the first few lines up until the `chain` property being an empty array. Such a module is often called a source or adapter module, enabling *Connected Vision* to retrieve data from any source (e.g. file system, YouTube, etc.). The actual source of the data is provided via the parameters of the configuration. It is also possible that the module is generating the data purely out of its configuration (e.g. the *SyntheticVideo* or *DummyBoundingBoxes* modules in the example below). On the one hand, these modules are very helpful for testing and development purposes. On the other hand, they are not very interesting from an educational point of view. This is due to the fact that they are not a good example for demonstrating a module workflow and/or development. They simply lack half of the functionality of a "normal" *Connected Vision* module.

For demonstration purposes, the *Skeleton* module is used. It has two input pins, one output pin and implements a simple algorithm. While the functionality of the *Skeleton* module is very basic, it uses the whole spectrum of the *Connected Vision* API and the algorithm is simple enough to not distract from the generic API usage.

Since the *Skeleton* module has two input pins consuming data from two different preceding modules, the `chain` property extends to the length of the sample configuration shown below. The preceding modules which provide the input data are the *SyntheticVideo* module and the *DummyBoundingBoxes* module. Both modules have a configuration of their own and their output pins are connected to the designated input pins of the *Skeleton* module.

The `chain` property is an array of chain objects. Each chain object needs to have two properties `connections` and `config`. The `connections` array contains connection objects with two properties: `inputPinID` is the name of the current module's input pin connected to the output pin of the preceding module identified by its `outputPinID`. In the sample config the name of the `inputPinID` and `outputPinID` pins are the same but this is not a requirement. It is required though, that the MIME type of the input- and output pin match. For metadata pins the properties of the input pin have to be at least a subset of the output pin properties.

```
{
    "name": "SyntheticVideo with DummyBoundingBoxes",
    "description": "computes the average color of a bounding box ",
    "version": 1,
    "id" : "",
    "profileID": 0,
    "moduleID": "Skeleton",
```

```
    "moduleURI" : "http://localhost:2020/Skeleton",
    "configFeatures":
    {
        "allowDynamicParameters": true
    },
    "params":
    {
        "dummy": 1,
        "dummy_dynamic_parameter": 100
    },
    "chain":
    [
        {
            "connections":
            [
                {
                    "inputPinID": "PNG-Image",
                    "outputPinID": "PNG-Image"
                }
            ],
            "config":
            {
                "name": "rectangle",
                "description": "a colored rectangle which moves from left to right",
                "version": 1,
                "id" : "",
                "profileID": 0,
                "moduleID": "SyntheticVideo",
                "moduleURI" : "http://localhost:2020/SyntheticVideo",
                "params":
                {
                    "recordingDateTime": 0,
                    "numberOfFrames": 256,
                    "height": 480,
                    "width": 640,
                    "fps": 10.0,
                    "sizeOfObject": 100,
                    "bgColor":
                    {
                        "R": 100,
                        "G": 200,
                        "B": 255
                    },
                    "fgColor":
                    {
                        "R": 100,
                        "G": 200,
                        "B": 0
```

```
                },
                "osdTextColor":
                {
                    "R": 0,
                    "G": 0,
                    "B": 0
                }
            },
            "chain": []
        }
    },
    {
        "connections":
        [
            {
                "inputPinID": "Detections",
                "outputPinID": "Detections"
            }
        ],
        "config":
        {
            "id": "",
            "name": "left to right",
            "description": "a bounding box which moves from left to right",
            "version": 1,
            "profileID": 0,
            "moduleID": "DummyBoundingBoxes",
            "moduleURI": "http://localhost:2020/DummyBoundingBoxes",
            "moduleLogoURI": "",
            "params":
            {
                "count": 10,
                "timestampStart": 0,
                "boundingBoxStart":
                {
                    "top": 0.4,
                    "left": 0.0,
                    "bottom": 0.6,
                    "right": 0.2
                },
                "timestampEnd": 100,
                "boundingBoxEnd":
                {
                    "top": 0.4,
                    "left": 0.8,
                    "bottom": 0.6,
                    "right": 1.0
                }
```

```
                },
                "chain": []
            }
        }
    ]
}
```

3.2. Submit

Once the configuration is composed, it has to be provided to the server using an HTTP POST request. Upon success, the configuration is returned with some of its properties such as the configuration ID being automatically populated. Submitting a configuration which already exists on the server has no effect.

3.3. Reset

A configuration can be resetted using the reset request. Possibly existing results are deleted and upon the next start of the configuration it is executed as if it was just submitted to the server for the first time.

WARNING	A reset also affects all predecessor modules used by this configuration and will delete the results of this modules as well. In order to reset only the current module but not its predecessors, the resetThis request can be used.

3.4. Start

A configuration can be started using the start request. This command initiates the processing of the input data and subsequently provides results at the output pins. If connected predecessor modules are not already running (or finished) they will be started implicitly.

3.5. Status

The current status of a config can be check using the status request.

3.6. Results

The results of the *Skeleton* module can be queried, once the config is started and computed at least a single result element. As mentioned in the module structure output section, the result of the *Skeleton* is simply a color value for each processed frame. Results can be queried by id, index or timestamp. For index and timestamp, also result ranges can be obtained. The various result request types can be found in the output pin section.

Chapter 4. Resources

4.1. Configuration

Configuration specific things.

4.1.1. POST /Skeleton/0/config

Parameters

Type	Name	Schema
FormData	**payload** *required*	string

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_Configuration
500	internal server error	Error
default	Unexpected error	Error

Consumes

- `application/json`

Produces

- `application/json`

4.1.2. GET /Skeleton/0/config?payload={config}&cmd=SET

Parameters

Type	Name	Schema
Path	**config** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_Configuration
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.3. GET /Skeleton/{configId}/config

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_Configuration
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.4. GET /Skeleton/{configId}/config?cmd=DELETE

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_Configuration
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.5. GET /Skeleton/{configId}/control/reset

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.6. GET /Skeleton/{configId}/control/resetThis

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.7. GET /Skeleton/{configId}/control/start

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.8. GET /Skeleton/{configId}/control/stop

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.9. GET /Skeleton/{configId}/control/stopThis

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.10. GET /Skeleton/{configId}/params

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ModuleParams
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.1.11. GET /Skeleton/{configId}/status

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ConfigurationStatus
404	not found	Error
500	internal server error	Error

HTTP Code	Description	Schema
default	Unexpected error	Error

Produces

- `application/json`

4.2. Host

Host specific things.

4.2.1. GET */hostStatus*

Responses

HTTP Code	Description	Schema
200	successful operation	HostStatus
default	Unexpected error	Error

Produces

- `application/json`

4.3. Module

Module specific things.

4.3.1. GET /Skeleton/configList

Responses

HTTP Code	Description	Schema
200	successful operation	< string > array
default	Unexpected error	Error

Produces

- `application/json`

4.3.2. GET /Skeleton/configListDetailed

Responses

HTTP Code	Description	Schema
200	successful operation	< Response 200 > array
default	Unexpected error	Error

Response 200

Name	Description	Schema
aliasIDs *optional*	a list of aliasIDs assigned to this config	< aliasIDs > array
commandList *optional*	a list of commands to control this config	commandList
id *optional*	config ID of this config	string
name *optional*	config name	string

aliasIDs

Name	Description	Schema
id *optional*	an aliasID assigned to this config	string
timestamp *optional*	creation timestamp of aliasID for this config	integer

commandList

Name	Schema
config *optional*	string
reset *optional*	string
resetThis *optional*	string

Name	Schema
start *optional*	string
status *optional*	string
stop *optional*	string
stopThis *optional*	string

Produces

- `application/json`

4.3.3. GET /Skeleton/init

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ModuleStatus
default	Unexpected error	Error

Produces

- `application/json`

4.3.4. GET /Skeleton/inputDescription

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_InputDescription
default	Unexpected error	Error

Produces

- `application/json`

4.3.5. GET /Skeleton/moduleDescription

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ModuleDescription
default	Unexpected error	Error

Produces

- `application/json`

4.3.6. GET /Skeleton/outputDescription

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_OutputDescription
default	Unexpected error	Error

Produces

- `application/json`

4.3.7. GET /Skeleton/release

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ModuleStatus
default	Unexpected error	Error

Produces

- `application/json`

4.3.8. GET /Skeleton/status

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_ModuleStatus
default	Unexpected error	Error

Produces

- `application/json`

4.4. OutputPin

OutputPin specific things.

4.4.1. GET /Skeleton/{configId}/Average/aftertime/{time}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**time** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.2. GET /Skeleton/{configId}/Average/beforetime/{time}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**time** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.3. GET /Skeleton/{configId}/Average/id/{id}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**id** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_OutputPin_Average

HTTP Code	Description	Schema
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.4. GET /Skeleton/{configId}/Average/index/{index}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**index** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	Skeleton_OutputPin_Average
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.5. GET /Skeleton/{configId}/Average/index/{startIndex}/{endIndex}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Type	Name	Schema
Path	**endIndex** *required*	integer(int64)
Path	**startIndex** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.6. GET /Skeleton/{configId}/Average/newest

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.7. GET /Skeleton/{configId}/Average/time/{startTime}/{endTime}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**endTime** *required*	integer(int64)
Path	**startTime** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

4.4.8. GET /Skeleton/{configId}/Average/time/{time}

Parameters

Type	Name	Schema
Path	**configId** *required*	string(string)
Path	**time** *required*	integer(int64)

Responses

HTTP Code	Description	Schema
200	successful operation	< Skeleton_OutputPin_Average > array
404	not found	Error
500	internal server error	Error
default	Unexpected error	Error

Produces

- `application/json`

Chapter 5. Definitions

5.1. Error

Name	Schema
error *optional*	string
status *optional*	integer

5.2. HostStatus

host status

Name	Description	Schema
hostID *required*	host id	string
systemID *required*	system ID	string

5.3. Skeleton_Configuration

config chain

Name	Description	Schema
aliasID *optional*	a list of (new) aliasIDs to be assigned to this config	< aliasID > array
callback *optional*	register a callback for live mode	callback
chain *required*		< chain > array
configFeatures *optional*	JSON schema of features of the config	configFeatures
description *required*	config description	string
initialParams *optional*		Skeleton_ModuleParams

Name	Description	Schema
moduleID *required*		string
moduleURI *required*	URI of module instance	string
params *optional*		Skeleton_ModuleParams
timestamp *optional*	time of modification **Default** : `0`	integer(int64)
version *required*	config version	integer(int64)

aliasID

Name	Description	Schema
timestamp *optional*	creation timestamp of aliasID for this config	integer(int64)

callback

Name	Description	Schema
moduleID *required*		string
moduleURI *required*	URI of module instance	string

chain

Name	Description	Schema
config *required*	config of sub module	string
connections *required*	connection between input pin of consumer module and output pin of producer module	< connections > array

connections

Name	Description	Schema
inputPinID *required*	id of input pin	string

Name	Description	Schema
outputPinID *required*	id of output pin	string

configFeatures

Name	Description	Schema
allowDynamicParameters *optional*	the config does support dynamic parameters **Default** : `false`	boolean

5.4. Skeleton_ConfigurationStatus

config status

Name	Description	Schema
chain *required*	Status of previous modules.	< string > array
estimatedFinishTime *optional*	estimated time to finish processing of this configChain **Default** : `-1`	integer(int64)
message *optional*	general message (e.g. description of last error) **Default** : `""`	string
moduleID *required*	ID of module	string
moduleURI *required*	URI of module instance	string
progress *required*	processing progress (0.0 - 1.0) **Minimum value** : `0.0` **Maximum value** : `1.0`	number
qualityOfService *required*	quality of service (QoS) parameters	qualityOfService
stableResults *required*	Index and timestamp range of available and stable (non-changing) results. One entry for each output pin.	< stableResults > array
startTime *required*	System time when the config was started. **Default** : `-1`	integer(int64)

Name	Description	Schema
status *required*	current status of config / job **Default** : `"n/a"`	enum (n/a, init, startup, running, stopping, stopped, finished, error, reset, cleanup)
systemTimeProcessing *required*	System time when the config was / is processed. This time is updated only during processing the config (i.e. status='running'). **Default** : `-1`	integer(int64)
timestamp *optional*	time of modification **Default** : `0`	integer(int64)

qualityOfService

Name	Description	Schema
compuationDuration *required*	computation duration based on the elapsed time between updates of the systemTimeProcessing parameter **Default** : `-1`	integer(int64)
compuationDurationAverage *required*	average of the computation duration of 10 preceding iterations based on the elapsed time between updates of the systemTimeProcessing parameter computed using the simple moving average method **Default** : `-1`	integer(int64)

stableResults

Name	Description	Schema
indexEnd *required*		integer(int64)
indexStart *optional*	**Default** : `0`	integer(int64)
pinID *required*	id of output pin	string
timestampEnd *required*		integer(int64)
timestampStart *optional*	**Default** : `0`	integer(int64)

5.5. Skeleton_InputDescription

Pin Description Schema

Type : < Skeleton_InputDescription > array

Skeleton_InputDescription

Name	Description	Schema
description *required*	long description of pin	string
maxPinCount *optional*	maximal number of this type of pin (use this if you want multiple input pins of the same type) **Default** : 1	integer(int64)
minPinCount *optional*	minimal number of this type of pin (set to 0 for optional input pin) **Default** : 1	integer(int64)
properties *optional*	JSON schema of data returned by pin (if MIME is application/json)	object
type *required*	MIME type of pin **Default** : `"application/json"`	string

5.6. Skeleton_InputPin_Detections

bounding box

Name	Description	Schema
boundingBox *required*	bounding box	boundingBox
configID *required*		string
objectID *required*		string
timestamp *required*	first detection	integer(int64)

boundingBox

Name	Description	Schema
bottom *required*	y position of bottom right point of the rectangle **Minimum value** : `0.0` **Maximum value** : `1.0`	number
left *required*	x position of top left point of the rectangle **Minimum value** : `0.0` **Maximum value** : `1.0`	number
right *required*	x position of bottom right point of the rectangle **Minimum value** : `0.0` **Maximum value** : `1.0`	number
top *required*	y position of top left point of the rectangle **Minimum value** : `0.0` **Maximum value** : `1.0`	number

5.7. Skeleton_ModuleDescription

Module Description Schema

Name	Description	Schema
APIVersion *required*	version of supported Connected Vision version	number
author *required*	author names of email address	string
configFeatures *optional*	JSON schema of features of the config	configFeatures
description *required*	long description of module	string
moduleFeatures *optional*	JSON schema of module features	moduleFeatures
moduleID *required*	unique ID of module	string
moduleURI *required*	URL of module instance. Will be updated by the module server.	string
params *optional*		Skeleton_ModuleParams
version *required*	version of module	number

configFeatures

Name	Description	Schema
allowDynamicParameters *optional*	the config does support dynamic parameters **Default** : `false`	boolean

moduleFeatures

Name	Description	Schema
supportsDynamicParameters *optional*	The module does support dynamic parameters. **Default** : `false`	boolean
supportsLive *optional*	The module does support live configurations. **Default** : `false`	boolean
supportsResume *optional*	The module does support resuming of stopped configurations. **Default** : `false`	boolean

5.8. Skeleton_ModuleParams

Skeleton Dummy Parameter

Name	Description	Schema
dummy *required*	just a place holder **Default** : `0`	integer(int64)
dummy_dynamic_parameter *required*	just a place holder for a dynamic parameter **Default** : `0`	integer(int64)

5.9. Skeleton_ModuleStatus

Module Status Schema

Name	Description	Schema
configsRunning *required*	list of currently processed configs / jobs	< string > array

Name	Description	Schema
configsWaiting *optional*	list of configs / jobs waiting to be processed	< string > array
moduleID *required*	ID of module	string
moduleStatus *required*	current status of module	enum (n/a, up, down)

5.10. Skeleton_OutputDescription

Pin Description Schema

Type : < Skeleton_OutputDescription > array

Skeleton_OutputDescription

Name	Description	Schema
description *required*	long description of pin	string
maxPinCount *optional*	maximal number of this type of pin (use this if you want multiple input pins of the same type) **Default** : 1	integer(int64)
minPinCount *optional*	minimal number of this type of pin (set to 0 for optional input pin) **Default** : 1	integer(int64)
properties *optional*	JSON schema of data returned by pin (if MIME is application/json)	object
type *required*	MIME type of pin **Default** : `"application/json"`	string

5.11. Skeleton_OutputPin_Average

This is a completely senseless output pin providing the average color of the area in the bounding box.

Name	Description	Schema
color *optional*	average color of bounding box	color
configID *required*		string

Name	Description	Schema
timestamp *required*	first detection	integer(int64)

color

Name	Description	Schema
B *required*	blue **Default** : `0`	integer(int64)
G *required*	green **Default** : `0`	integer(int64)
R *required*	red **Default** : `0`	integer(int64)