VCA Integration

v1.4.1r1

Chapter 1

Introduction

This is the API integration document for the VCAcore Video Analytics system.

The VCAcore Video Analytics system is available on a number of platforms:

- **VCAbridge**, an embedded hardware solution.
- **VCAserver**, an application and service for Windows and Linux.

The video processing features of the VCAcore API are identical across the various platforms. However, the system-specific settings are different, and these differences are described in the following sections.

Chapter 2

REST API Introduction

2.1 Configuration Tree

The REST API of VCAcore is used by Core's web UI to configure the web backend.

The VCAcore REST API allows the user to change its configuration tree, which is where the entire VCAcore configuration is stored. The configuration tree can be accessed using the URL below:

`http://SERVER_IP:PORT/api.json`

2.2 Interacting with the Configuration

2.2.1 Adding Objects to the Configuration Tree

To add an object to the configuration, send a POST request to the appropriate endpoint for the type of object you are adding.

For example, to add an element:

POST the element to `/api/elements` with the following payload:

```
{
    "typename": "rtsp",
    "location": "rtsp://192.168.1.1/stream",
    "user_id": "admin",
    "user_pw": "password",
    "do_rtsp_keep_alive": "FALSE",
    "protocols": "7",
    "name": "MY_RTSP_STREAM_1"
}
```

Response:

```
{
    "index": 2
}
```

The response is the index of the newly created object in the configuration tree, in this case the RTSP element at `/api/elements/2`. This index can be used as a reference to update this object.

It is not necessary to include all the properties of an object in the JSON payload of the POST request. Where a property is not specified, a default value will be used. For objects which have a `typename` property, this property is required.

2.2.2 Modifying Properties in the Configuration Tree

Any property can be modified by sending the appropriate PUT request to the full path of the property, with the desired value in the payload. For example, if you wanted to change the `name` property on the element that was just created, you would send a PUT request to `/api/elements/2/name` with the following payload:

```
"New Name"
```

It is also possible to update multiple properties in one request by sending a PUT request to the parent object. In this example, you could send a PUT request to `/api/elements/2` with the following payload:

```
{
    "user_id": "root",
    "user_pw": "pass",
    "name": "New Name"
}
```

These methods can be applied to any object in the `api.json`.

2.2.3 Getting Properties in the Configuration Tree

Any or all of the configuration tree can be retrieved from the REST API by sending a GET request to the desired endpoint. For example, sending a 'GET' request to `/api/elements/2/` would return:

```
{
    "typename": "rtsp",
    "location": "rtsp://192.168.1.1/stream",
    "user_id": "root",
    "user_pw": "pass",
    "do_rtsp_keep_alive": "FALSE",
    "protocols": "7",
    "name": "New Name"
}
```

Additionally, the entire configuration can be returned by sending a GET request to `/api`.

2.2.4 Error Handling

When a request was successful, the VCAcore web server responds with a `__2xx__` code.

Requests that add an object using POST will also have the index of the newly-created object in the response, as shown below:

```
{
    "index": 4
}
```

When a request has failed, the web server returns a `__4xx__` or `__5xx__` error code.

Additionally, the web server will return a descriptive error string that can be used to diagnose the problem. The error is returned as an object with an `error` property, and the error string as its value:

```
{
    "error": "An error has occured."
}
```

2.2.5 HTTP Request Headers

It is a requirement to add the `Content-Type: application/json` header to all REST API requests.

2.3 Custom Objects

2.3.1 Point Object

A point object represents a point in 2D space. An example point object is given below:

```
{
    "x": 200,
    "y": 400
}
```

The properties of a point object are as follows:

Property	Type	Description	Possible values
x	Unsigned Integer	The x coordinate of this point	Any unsigned integer between 0 and 65535 (inclusive)
y	Unsigned Integer	The y coordinate of this point	Any unsigned integer between 0 and 65535 (inclusive)

2.3.2 Colour Object

A colour object represents an RGBA colour value An example colour object is given below:

```
{
    "r": 100,
    "g": 200,
    "b": 140,
```

```
    "a": 90
}
```

The properties of a colour object are as follows:

Property	Type	Description	Possible values
r	Unsigned Integer	The red component of the colour	Any unsigned integer between 0 and 255 (inclusive)
g	Unsigned Integer	The green component of the colour	Any unsigned integer between 0 and 255 (inclusive)
b	Unsigned Integer	The blue component of the colour	Any unsigned integer between 0 and 255 (inclusive)
a	Unsigned Integer	The alpha component of the colour	Any unsigned integer between 0 and 255 (inclusive)

2.4 Licenses

2.4.1 Adding a License

To add a license, simply send a POST request to `/api/licenses/vca`. A sample payload is given below:

Note: The license string provided below is invalid. You will need to purchase a license for your hardware GUID.

```
{
    "license": "43214321EDFAEFDEAFDEAFDEAFDEADFADAAEEADAFDEAFEDA"
}
```

Property	Type	Description	Possible values
license	String	The license string	Any valid license string

2.4.2 Retrieving the Hardware GUID

To retrieve the hardware GUID, you need to send a GET request to `/api/hardware/guid.json` The GUID is returned as a string, as shown below:

```
"434E3D4AE43DAE34DEA43DEA43DE34ADE3A4DEA34DE4A3DE4"
```

Chapter 3

REST API Channels

3.1 Elements

There are two types of elements supported by VCAcore - file and RTSP elements. Elements are inputs to channels, and must be added before a channel is created.

3.1.1 Adding an RTSP Element

An RTSP element may be created added by sending a POST request to `/api/elements` endpoint. A sample RTSP element is shown below:

```
{
    "typename": "rtsp",
    "location": "",
    "user_id": "",
    "user_pw": "",
    "do_rtsp_keep_alive": "FALSE",
    "protocols": "7",
    "name": ""
}
```

Property	Type	Description	Possible values
`location`	String	The URI of this RSTP stream	Any string, can be empty
`user_id`	String	The username to be used to authenticate with the RTSP server	Any string, can be empty
`user_pw`	String	The password to be used to authenticate with the RTSP server	Any string, can be empty
`do_rtsp_keep_alive`	String	A boolean (represented as a string), specifying whether keep-alive should be enabled in this RTSP stream	`"TRUE"` or `"FALSE"`
`protocols`	String	The protocol to use for this RTSP stream	Set to `"4"` for RTSP over TCP, otherwise set to `"7"`

Property	Type	Description	Possible values
name	String	A user-specified name for this element	Any string, can be empty

3.1.2 Adding a File Element

An RTSP element may be created by sending a POST request to the `api/elements` endpoint. A sample file element is shown below:

```
{
    "typename": "file",
    "name": "",
    "location": "las-vegas.mp4"
}
```

Property	Type	Description	Possible values
location	String	The filename of the video (video files must be located in the `share/test-clips` subfolder of the install folder)	Any string, can be empty
name	String	A user-specified name for this element	Any string, can be empty

3.2 Channels

Once an element has been added, it can be linked to a channel.

3.2.1 Adding a Channel

Once a file or RTSP element has been added, and its index is known, this index can be used to link it to a channel A channel may be added by sending a POST request to `/api/channels`. A sample channel object is shown below:

```
{
    "name": "",
    "enabled": true,
    "input": 5,
    "output": null,
    "license": 1,
    "event_retrigger_time": 5,
    "tracking_engine": "object_tracker",
    "tracker": {
        "stationary_time": 5000,
        "stationary_hold_on_time": 60000,
        "minimum_object_size": 10,
        "detection_point": 0,
```

```
        "sensitivity_threshold": 4
    },
    "calibration": {
        "enabled": false,
        "height": 4,
        "tilt": 50,
        "fov": 40,
        "roll": 0,
        "pan": 0,
        "horizon": false,
        "grid": {
            "enabled": true,
            "stroke": {
                "r": 115,
                "g": 210,
                "b": 22,
                "a": 1
            },
            "fill": {
                "r": 136,
                "g": 138,
                "b": 133,
                "a": 0
            },
            "spacing": 2
        }
    },
    "tamper": {
        "enabled": false,
        "alarm_timeout": 20000,
        "area_threshold": 40,
        "low_light": false
    },
    "scene_change": {
        "mode": 1,
        "alarm_timeout": 3000,
        "area_threshold": 40
    },
    "annotation": {
        "zones": false,
        "objects": true,
        "class": false,
        "height": false,
        "speed": false,
        "area": false,
        "ticker": true,
        "blob_map": true,
        "dl_class": true,
```

```
        "system_message": true,
        "line_counters": true,
        "counters": true,
        "colour_signature": true,
        "tracker_internal_state": false
    },
    "classification": [
        {
            "name": "Person",
            "area": {
                "min": 5,
                "max": 20
            },
            "speed": {
                "min": 0,
                "max": 20
            }
        },
        {
            "name": "Vehicle",
            "area": {
                "min": 40,
                "max": 1000
            },
            "speed": {
                "min": 0,
                "max": 200
            }
        },
        {
            "name": "Clutter",
            "area": {
                "min": 0,
                "max": 4
            },
            "speed": {
                "min": 0,
                "max": 50
            }
        },
        {
            "name": "Group Of People",
            "area": {
                "min": 21,
                "max": 39
            },
            "speed": {
                "min": 0,
```

```
                "max": 20
            }
        }
    ],
    "stabilisation": {
        "enabled": false
    },
    "dl_filter": {
        "available": true,
        "classes": [
            {
                "label": "person",
                "allowed": true,
                "threshold": 0.5
            },
            {
                "label": "vehicle",
                "allowed": true,
                "threshold": 0.5
            }
        ]
    },
    "colour_signature": {
        "enabled": false
    }
}
```

Below is a list of properties of a channel object

Property	Type	Description	Possible values
`name`	String	A user-specified name for this element	Any string, can be empty
`enabled`	Boolean	A boolean value specifying whether this channel is enabled	`true` or `false`
`input`	Unsigned integer	The index of a file or RTSP element to use as the input for this channel	Any unsigned integer
`output`	Unsigned integer	This property is deprecated, and must always be set to `null`	Any unsigned integer, or `null`
`license`	Unsigned integer	The index of a license object to use as the license for this channel	Any unsigned integer
`event_retrigger_time`	Unsigned integer	The time (in milliseconds) that must elapse before an event is re-triggered	Any unsigned integer
`tracking_engine`	String	Defines the tracking engine that will be run on the channel	Any valid tracker engine string, `object_tracker` or `dl_people_tracker`
`tracker`	Object	Tracking engine specific settings	Any valid tracker object

Property	Type	Description	Possible values
`calibration`	Object	The calibration object specifying the calibration parameters of this channel	Any valid calibration object
`tamper`	Object	The tamper object specifying the tamper parameters of this channel	Any valid tamper object
`scene_change`	Object	The `scene_change` object specifying the scene-change parameters of this channel	Any valid scene-change object
`annotation`	Object	The `annotation` object specifying which metadata annotations are rendered on the channel	Any valid annotation object
`classification`	Array of classification objects	The array of classification objects to use for this channel	Any valid array of classification objects
`stabilisation`	Object	The `stabilisation` object specifying the stabilisation parameters of this channel	Any valid stabilisation object
`dl_filter`	Object	The object specifying the `dl_filter` parameters of this channel	Any valid stabilisation object
`colour_signature`	Object	The `colour_signature` object specifying the colour signature parameters of this channel	Any valid colour signature object

The following are the properties of a `tracker` object, properties apply to a specific `tracking_engine`:

Property	Type	Description	Possible values
`stationary_timeout`	Unsigned integer	`object_tracker` value: defines in ms, the time detected motion must be static before it is defined as abandoned/removed	Any unsigned integer
`stationary_tracked_on_timeout`	Unsigned integer	`object_tracker` value: defines in ms, the time a stationary object continues to be tracked for	Any unsigned integer
`minimum_object_size`	Unsigned integer	`object_tracker` value: the number of tracking pixels that detected motion must contain before it is classed as a tracked object.	Any unsigned integer
`detection_point`	Unsigned integer	`object_tracker` value: defines the location of the ground point for a bounding box. 0 automatic, 1 bottom mid of bounding box, 2 centre of bounding box	0, 1, or 2
`sensitivity_threshold`	Float	`object_tracker` value: defines how sensitive the object tracker is to movement	1-16

The following are the properties of a `calibration` object:

Property	Type	Description	Possible values
`enabled`	Boolean	A boolean value specifying whether calibration is enabled for this channel	`true` or `false`
`height`	Unsigned integer	The height of the camera in meters	Any unsigned integer
`tilt`	Unsigned integer	The 'tilt' of the camera in degrees	Any unsigned integer
`fov`	Unsigned integer	The 'field of view' parameter of the camera in degrees	Any unsigned integer
`roll`	Unsigned integer	The 'roll' value of the camera in degrees	Any unsigned integer
`pan`	Unsigned integer	The 'pan' value of the camera in degrees	Any unsigned integer
`horizon`	Boolean	A boolean value specifying whether the horizon is displayed in the calibration grid	`true` or `false`
`grid`	Object	An object specifying the calibration grid parameters	Any valid grid object

Below is a list of the properties of the `grid` object:

Property	Type	Description	Possible values
`stroke`	Object	A colour object specifying the stroke colour of the calibration grid	Any valid colour object
`fill`	Object	A colour object specifying the fill colour of the calibration grid	Any valid colour object
`spacing`	Unsigned integer	The spacing between lines in the calibration grid	Any unsigned integer

Below is a list of the properties of the `tamper` object:

Property	Type	Description	Possible values
`enabled`	Boolean	A boolean value specifying whether tamper detection is enabled for this channel	`true` or `false`
`alarm_timeout`	Unsigned integer	The alarm hold-off time (in milliseconds)	Any unsigned integer
`area_threshold`	Unsigned integer	The area threshold (percentage change) for tamper detection	Any unsigned integer
`low_light`	Boolean	A boolean value specifying whether low light tamper detection should be enabled	`true` or `false`

Below is a list of the properties of the `scene_change` object:

Property	Type	Description	Possible values
`mode`	Unsigned integer	An integer specifying the scene-change detection mode	0 for Disabled, 1 for automatic and 2 for manual
`alarm_timeout`	Unsigned integer	The alarm hold-off time (in milliseconds)	Any unsigned integer
`area_threshold`	Unsigned integer	The area threshold (percentage change) for tamper detection	Any unsigned integer

Below is a list of the properties of the `classification` object:

Property	Type	Description	Possible values
`name`	String	A user-specified name for this element	Any string, can be empty
`area`	Object	A threshold object to specify the minimum and maximum area cut-off for this class. The area unit is square meter	Any valid threshold object
`speed`	Object	A threshold object to specify the minimum and maximum speed cut-off for this class. The speed unit is kilometres per hour	Any valid threshold object

Below is a list of the properties of the `threshold` object:

Property	Type	Description	Possible values
`min`	Unsigned integer	The minimum value	Any unsigned integer
`max`	Unsigned integer	The maximum value	Any unsigned integer

Below is a list of the properties of the `stabilisation` object:

Property	Type	Description	Possible values
`enabled`	Boolean	A boolean value specifying whether the stabilisation is enabled for this channel	`true` or `false`

Below is a list of the properties of the `dl_filter` object:

Property	Type	Description	Possible values
`available`	Boolean	A system defined boolean value specifying whether the deep learning filter is available for this channel	`true` or `false`
`classes`	Object	A system generated list of object classes the deep learning filter can detect	classes object

Below is a list of the properties of the `classes` object:

Property	Type	Description	Possible values
`label`	String	A system-specified label for an object class	String
`allowed`	Boolean	A boolean value specifying whether the deep learning filter is checking for this 'label'	`true` or `false`
`threshold`	Float	A threshold to specify the minimum confidence score required for the object to be classed as the 'label'	Float between 0 - .99 inclusive

Below is a list of the properties of the `colour_signature` object:

Property	Type	Description	Possible values
`enabled`	Boolean	A boolean value specifying whether the colour signature is enabled for this channel	`true` or `false`

Finally, the observables must be added and linked to the channel. There are two observables: one for analytics, and one for loss of signal. Observables allow the events from a specific channel to trigger an action.

3.2.2 Channel Snapshots

A snapshot API is provided which will provide a JPEG encoded image from a defined channel. The resolution of the JPEG will be defined by the input resolution of the stream. A snapshot can be retrieved by sending a `GET` request to the snapshot endpoint:

`http://SERVER_IP:PORT/snapshot/CHANNEL_ID/latest`

It is a requirement to add the `Content-Type: image/jpeg` header to all requests to the snapshot API.

When a snapshot API request is sent to VCAcore, the JPEG image is generated on demand. JPEG encoding is a resource intensive process and therefore this API is not designed for channel monitoring at high frame rates. As such it is advised that calls made to this API are done so sparingly.

Chapter 4

REST API Observables and Other Sources

4.1 Overview

An observable is an object which monitors the metadata produced by VCAcore and generates an event when specific triggers are met. Commonly, observables are used to represent 'rules' which are associated with a channel that fires an event when a specific condition is met by objects tracked in that channel (e.g. presence or dwell).

An observable can also represent one of VCAcore's 'other sources', these work at a global, rather than channel, level triggering events when a specific global trigger is detected (e.g. VCAcore being Armed or when a scheduled time passes).

Observables can be linked to actions, so that when an event is generated by an observable, an action is performed.

4.1.1 General Concepts

To add an observable, send a `POST` request to the `/api/observables` endpoint. Please use the method mentioned earlier to add this element. A sample 'Presence' observable is shown below:

```
{
    "typename": "vca.observable.Presence",
    "channel": 0,
    "zone": 4294967295,
    "name": "Presence 5",
    "dependents": [],
    "triggers_action": true
}
```

Below is a list of properties that are common to all observables:

Property	Type	Description	Possible values
`name`	String	A user-specified name for this element	Any string, can be empty
`dependents`	Array	The array of observables this observable depends on	Any array of unsigned integers, can be an empty array

Property	Type	Description	Possible values
`triggers_action`	Boolean	A boolean specifying whether this observable can trigger actions	`true` or `false`

For a complete overview of observable types (Basic Rules, Filters, Conditional Rules and Other Sources), including their function and use cases, please see the full VCAcore Manual.

What follows is a list of all the supported types of observables in VCAcore:

4.2 Basic Rules

4.2.1 Abandoned Observable

The 'Abandoned' observable is a basic observable which generates an event when an object has been either left within a defined zone or when an object is removed from a defined zone. Please use the method mentioned earlier to add this element. A sample 'Abandoned' observable is shown below:

```
{
    "typename": "vca.observable.Abandoned",
    "channel": 0,
    "zone": 3,
    "name": "Abandoned 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`zone`	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.2 Appear Observable

The 'Appear' observable is a basic observable which generates an event when an object starts being tracked within a zone, e.g. a person who appears in the scene from a doorway. Please use the method mentioned earlier to add this element. A sample 'Appear' observable is shown below:

```
{
    "typename": "vca.observable.Appear",
    "channel": 0,
    "zone": 3,
    "name": "Appear 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.3 Direction Observable

The 'Direction' observable is a basic observable which generates an event when an object is moving in a specific direction. Please use the method mentioned earlier to add this element. A sample 'Direction' observable is shown below:

```
{
    "typename": "vca.observable.Direction",
    "channel": 0,
    "angle": 77,
    "anglethreshold": 51,
    "zone": 0,
    "name": "Direction 3",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
angle	Integer	The direction an object should be travelling to trigger an event	Any unsigned integer 0 - 360

Property	Type	Description	Possible values
anglethreshold	Integer	The threshold of angles accepted by the rule	Any unsigned integer 0 - 90
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.4 Disappear Observable

The 'Disappear' observable is a basic observable which generates an event when an object stops being tracked within a zone, e.g. a person who exits the scene through a doorway. Please use the method mentioned earlier to add this element. A sample 'Disappear' observable is shown below:

```
{
    "typename": "vca.observable.Disappear",
    "channel": 0,
    "zone": 3,
    "name": "Disappear 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.5 Deep Learning Presence Observable

The 'Deep Learning Presence' observable is a basic observable which generates an event when an object is present inside a zone and is also classified as one of the classes configured in the `channel` `dl_filter` settings. For this observable to generate an event, the `dl_filter` must have `enabled` set to true. The deep learning presence observable cannot be used an input for any other observable.

Please use the method mentioned earlier to add this element. A sample 'Deep Learning Presence' observable is shown below:

```
{
    "typename": "vca.observable.DLPresence",
    "channel": 0,
    "zone": 3,
    "name": "Deep Learning Presence 1",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.6 Dwell Observable

The 'Dwell' observable is a basic observable which generates an event when an object has remained in a zone for a specified amount of time. Please use the method mentioned earlier to add this element. A sample 'Dwell' observable is shown below:

```
{
    "typename": "vca.observable.Dwell",
    "channel": 0,
    "zone": 0,
    "interval": 10000,
    "name": "Dwell 3",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer

Property	Type	Description	Possible values
interval	Unsigned Integer	The interval value of this observable, in milliseconds	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.7 Enter Observable

The 'Enter' observable is a basic observable which generates an event when an object enter a zone e.g when an object crosses from the outside of a zone to the inside of a zone. Please use the method mentioned earlier to add this element. A sample 'Enter' observable is shown below:

```
{
    "typename": "vca.observable.Enter",
    "channel": 0,
    "zone": 3,
    "name": "Enter 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.8 Exit Observable

The 'Exit' observable is a basic observable which generates an event when an object leaves a zone e.g when an object crosses from the inside of a zone to the outside of a zone. Please use the method mentioned earlier to add this element. A sample 'Exit' observable is shown below:

```
{
    "typename": "vca.observable.Exit",
    "channel": 0,
    "zone": 3,
    "name": "Exit 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.9 Line Counter Observable

The 'Line Counter' observable is a basic observable which generates an event when an object is detected crossing a line. The referenced zone must be configured as a line, not a polygon.

Please use the method mentioned earlier to add this element. A sample 'Line Counter' observable is shown below:

```
{
    "typename": "vca.observable.Line_Counter",
    "channel": 0,
    "direction": "both",
    "zone": 5,
    "calibration_width": 0,
    "filter_shadows": false,
    "name": "Line Counter 10",
    "dependents": [ ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer

Property	Type	Description	Possible values
`direction`	String	Defines the direction the line counter detects movement	String - “both” or “a” or “b”
`zone`	Integer	The index of a zone to associate with this observable	Any unsigned integer
`calibration_width`	Float	Defines the expected width of an object to cross the line, counts to go up by more than 1	Any unsigned float - 0 to turn the calibration off
`filter_shadows`	Boolean	A boolean value specifying whether to filter shadows	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.10 Loss of Signal Observable

This observable generates events when a stream is interrupted. Please use the method mentioned earlier to add this element. A sample ‘Loss-of-signal’ observable is shown below:

```
{
    "typename": "vca.observable.LossOfSignal",
    "heartbeat_frequency": 1000,
    "channel": 0,
    "name": " - Loss of Signal",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its `dependents` array should always be empty.

4.2.11 Presence Observable

The ‘Presence’ observable is a basic observable which generates an event when an object is present inside a zone. Please use the method mentioned earlier to add this element. A sample ‘Presence’ observable is shown below:

```
{
    "typename": "vca.observable.Presence",
    "channel": 0,
    "zone": 4294967295,
    "name": "Presence 5",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.12 Stopped Observable

The 'Stopped' observable is a basic observable which generates an event when an object is stationary inside a zone for longer than the specified amount of time. Please use the method mentioned earlier to add this element. A sample 'Stopped' observable is shown below:

```
{
    "typename": "vca.observable.Stopped",
    "zone": 4,
    "duration": 10000,
    "channel": 0,
    "name": "Stopped 3",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

Property	Type	Description	Possible values
duration	Unsigned Integer	The interval value of this observable, in milliseconds	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.2.13 Tailgating Observable

The 'Tailgating' observable is a basic observable which generates an event when an object crosses through a zone or over a line within a set duration of one another. Please use the method mentioned earlier to add this element. A sample 'Tailgating' observable is shown below:

```
{
    "typename": "vca.observable.Tailgating",
    "channel": 0,
    "zone": 0,
    "duration": 2000,
    "name": "Tailgating 3",
    "dependents": [],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
channel	Integer	The index of the channel this observable is associated with	Any unsigned integer
duration	Unsigned Integer	The duration value of this observable, in milliseconds	Any unsigned integer
zone	Integer	The index of a zone to associate with this observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a basic rule, its dependents array should always be empty.

4.3 Filters

4.3.1 Speed Filter Observable

The 'Speed Filter' observable is a filter which generates an event when the object which has triggered the `input` observable is travelling between a min and max speed. For this observable to generate an event, the `channel` must have been calibrated. Please use the method mentioned earlier to add this element. A sample 'Speed Filter' observable is shown below:

```
{
    "typename": "vca.observable.Speed",
    "channel": 0,
    "input": 8,
    "minspeed": 3,
    "maxspeed": 10,
    "name": "Speed Filter 11",
    "dependents": [
        8
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer
`minspeed`	Integer	The minimum speed an object must be travelling to be accepted by the rule	Any unsigned integer
`maxspeed`	Integer	The maximum speed an object must be travelling to be accepted by the rule	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a filter, for this observable to function correctly, the input and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.3.2 Object Filter Observable

The 'Object Filter' observable is a filter which generates an event when the object which has triggered the `input` observable is classed as one of the classes in the `filter` array. A class must match one of the `channel classification` entries. For this observable to generate an event, the `channel` must have been calibrated. Please use the method mentioned earlier to add this element. A sample 'Object Filter' observable is shown below:

```
{
    "typename": "vca.observable.ObjectFilter",
    "channel": 0,
    "input": 8,
    "filters": [
        "Person",
        "Vehicle"
    ],
    "name": "Object Filter 12",
    "dependents": [
        8
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer
`filters`	Array	Array of strings defining object classes found under a `channel classification`	Any array of strings, can be an empty array

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a filter, for this observable to function correctly, the input and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.3.3 Colour Filter Observable

The 'Colour Filter' observable is a filter which generates an event when the object which has triggered the `input` observable has 5% or more of the any colour defined under `filters`. For this observable to generate an event, the `channel` must have the colour signature algorithm enabled. Please use the method mentioned earlier to add this element. A sample 'Colour Filter' observable is shown below with all ten possible colours defined under 'filters':

```
{
    "typename": "vca.observable.ColourFilter",
    "channel": 0,
    "input": 6,
    "filters": [
        "Black",
        "Grey",
        "Blue",
        "Brown",
        "Cyan",
        "Green",
        "Red",
        "Magenta",
        "White",
        "Yellow"
    ],
    "name": "Colour Filter 13",
    "dependents": [
        6
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer
`filters`	Array	Array of strings defining colours	Any array of strings, can be an empty array

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a filter, for this observable to function correctly, the input and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.3.4 Deep Learning Filter Observable

The 'Deep Learning Filter' observable is a filter which generates an event when the object which has triggered the `input` observable is classed as one of the classes configured in the `channel` `dl_filter` settings. For this observable to generate an event, the `dl_filter` must have `enabled` set to true. The deep learning filter observable cannot be used an input for any other observable. Please use the method mentioned earlier to add this element. A sample 'Deep Learning Filter' observable is shown below:

```
{
    "typename": "vca.observable.DeepLearningFilter",
    "channel": 0,
    "input": 5,
    "name": "Deep Learning Filter 14",
    "dependents": [
        5
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a filter, for this observable to function correctly, the input and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.3.5 Source Filter Observable

The 'Source Filter' observable is a filter which generates an event when the `input` observable triggers an event and the `source` observable is in an `on` state. Valid inputs for use as a `source` are either

the Schedule or HTTP other source observables. Please use the method mentioned earlier to add this element. A sample 'Deep Learning Filter' observable is shown below:

```
{
    "typename": "vca.observable.SourceFilter",
    "channel": 0,
    "input": 5,
    "source": 7,
    "name": "Source Filter",
    "dependents": [
        5
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer
`source`	Integer	The index of a HTTP or Schedule other source observable	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Note: As this is a filter, for this observable to function correctly, the input and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.4 Conditional Rules

4.4.1 And Observable

The And observable is a representation of the logical 'AND' operation. Please use the method mentioned earlier to add this element. A sample 'And' observable is shown below:

```
{
    "typename": "vca.observable.And",
    "channel": 0,
    "inputa": 4,
```

```
    "inputb": 5,
    "constrain_target": true,
    "name": "And 2",
    "dependents": [
        4,
        5
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`inputa`	Integer	The index of another observable, which becomes the first input	Any unsigned integer
`inputb`	Integer	The index of another observable, which becomes the second input	Any unsigned integer
`constrain_target`	Boolean	A boolean specifying whether this observable generates events per-target	`true` or `false`

For a list of properties common to all observables, please see the General concepts section on Observables.

Please note that as this observable is a Conditional Rule, for this observable to function correctly, the two inputs and the dependents property must be kept in-sync. As an example, if the observable has 3 as `inputa`, and 4 as `inputb`, its `dependents` property should be the list:

```
[
    3,
    4
]
```

4.4.2 Continuously Observable

The 'Continuously' observable generates an event when another event has been occurring continuously for a certain amount of time. The time parameter is user-specified. Please use the method mentioned earlier to add this element. A sample 'Continuously' observable is shown below:

```
{
    "typename": "vca.observable.Continuously",
    "interval": 1000,
    "channel": 0,
    "input": 6,
    "constrain_target": true,
    "name": "Continuously 3",
```

```
    "dependents": [
        6
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input to this one	Any unsigned integer
`interval`	Unsigned Integer	The interval value of this observable, in milliseconds	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

Please note that as this observable is a Conditional Rule, for this observable to function correctly, the two inputs and the dependents property must be kept in-sync. As an example, if the observable has 9 as `input`, its `dependents` property should be the list:

```
[
    9
]
```

4.4.3 Counter Observable

The 'Counter' observable generates an event when the value of `count` changes. The `count` value is defined by the `input` observables which either increment, decrement, or define its occupancy. Please use the method mentioned earlier to add this element. A sample 'Counter' observable is shown below:

```
{
    "typename": "vca.observable.Counter",
    "channel": 0,
    "count": -20,
    "x": 32767,
    "y": 32767,
    "increment_inputs": [
        2,
        6
    ],
    "decrement_inputs": [
        7
    ],
    "occupancy_inputs": [
```

```
        8
    ],
    "name": "Counter 17",
    "dependents": [
        2,
        7,
        8,
        6
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`count`	Integer	The index of another observable, which becomes the input to this one	Any unsigned integer
`x`	Integer	The x coordinate of the counter	Any unsigned integer between 0 and 65535 (inclusive)
`y`	Integer	The y coordinate of the counter	Any unsigned integer between 0 and 65535 (inclusive)
`increment_inputs`	Array	The array of observables which will increment `count` when an they trigger an event	Any array of unsigned integers, can be an empty array
`decrement_inputs`	Array	The array of observables which will decrement `count` when an they trigger an event	Any array of unsigned integers, can be an empty array
`occupancy_inputs`	Array	The array of observables which will add to `count` the number of objects which are triggering the observable	Any array of unsigned integers, can be an empty array

For a list of properties common to all observables, please see the General Concepts section on Observables.

Please note that as this observable is a Conditional Rule, for this observable to function correctly, the three inputs and the dependents property must be kept in-sync. As an example, if the observables has 2, and 6 are `increment_inputs`, 7 is a `decrement_inputs` and 8 is a `occupancy_inputs` its `dependents` property should be the list:

```
[
    2,
    7,
    8,
    6
```

```
]
```

4.4.4 Or Observable

The Or observable is a representation of the logical 'OR' operation. Please use the method mentioned earlier to add this element. A sample 'Or' observable is shown below:

```
{
    "typename": "vca.observable.Or",
    "channel": 0,
    "inputa": 2,
    "inputb": 7,
    "constrain_target": true,
    "name": "Or 4",
    "dependents": [
        2,
        7
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`inputa`	Integer	The index of another observable, which becomes the first input	Any unsigned integer
`inputb`	Integer	The index of another observable, which becomes the second input	Any unsigned integer
`constrain_target`	Boolean	A boolean specifying whether this observable generates events per-target	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables.

Please note that as this observable is a Conditional Rule, for this observable to function correctly, the two inputs and the dependents property must be kept in-sync. As an example, if the observable has 3 as `inputa`, and 4 as `inputb`, its `dependents` property should be the list:

```
[
    3,
    4
]
```

4.4.5 Previous Observable

The 'Previous' observable generates an event when another event has occurred previously, within a certain amount of time. The time parameter is user-specified. Please use the method mentioned earlier to add this element. A sample 'Previous' observable is shown below:

```
{
    "typename": "vca.observable.Previous",
    "interval": 1000,
    "channel": 0,
    "input": 5,
    "constrain_target": true,
    "name": "Previous 6",
    "dependents": [
      5
    ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`channel`	Integer	The index of the channel this observable is associated with	Any unsigned integer
`input`	Integer	The index of another observable, which becomes the input	Any unsigned integer
`interval`	Unsigned Integer	The interval value of this observable, in milliseconds	Any unsigned integer
`constrain_target`	Boolean	A boolean specifying whether this observable generates events per-target	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables.

Please note that as this observable is a Conditional Rule, for this observable to function correctly, the two inputs and the dependents property must be kept in-sync. As an example, if the observable has 7 as `input`, its `dependents` property should be the list:

```
[
    7
]
```

4.5 Other Sources

4.5.1 Armed

The 'Armed' observable is a channel-independent observable which generates an event when the VCA-core system is Armed. Please use the method mentioned earlier to add this element. A sample 'Armed' observable is shown below:

```
{
    "typename": "vca.observable.Armed",
    "name": "Armed Source",
    "dependents": [ ],
    "triggers_action": true
}
```

The 'Armed' observable does not have any specific properties. For a list of properties common to all observables, please see the General Concepts section on Observables.

4.5.2 Disarmed

The 'Disarmed' observable is a channel-independent observable which generates an event when the VCAcore system is Disarmed. Please use the method mentioned earlier to add this element. A sample 'Disarmed' observable is shown below:

```
{
    "typename": "vca.observable.Disarmed",
    "name": "Disarmed Source",
    "dependents": [ ],
    "triggers_action": true
}
```

The 'Disarmed' observable does not have any specific properties. For a list of properties common to all observables, please see the General Concepts section on Observables.

4.5.3 HTTP

The 'HTTP' observable is a channel-independent observable which generates an event each time the `state` value is changed to `true`. Please use the method mentioned earlier to add this element. A sample 'HTTP' observable is shown below:

```
{
    "typename": "vca.observable.Http",
    "state": true,
    "name": "Http Source",
    "dependents": [ ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`state`	Boolean	A boolean specifying whether this observable is on or off	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables.

4.5.4 Interval

The 'Interval' observable is a channel-independent observable which generates an event each time the `interval` period passes. Please use the method mentioned earlier to add this element. A sample 'Interval' observable is shown below:

```
{
    "typename": "vca.observable.Interval",
    "interval": 1000,
    "name": "Interval Source",
    "dependents": [ ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`interval`	Unsigned Integer	The interval value of this observable, in milliseconds	Any unsigned integer

For a list of properties common to all observables, please see the General Concepts section on Observables.

4.5.5 Schedule

The 'Schedule' observable is a channel-independent observable which generates an event when the system clock coincides with a scheduled 'on' period. Events are generated once per `on` period (if VCAcore is started during an `on` period a single event is fired as soon as possible). When `set_arm_disarm` is `true`, VCAcore will be armed and disarmed according to the periods of `on` / `off` defined by the `schedule`. Please use the method mentioned earlier to add this element. A sample 'Schedule' observable is shown below:

```
{
    "typename": "vca.observable.Schedule",
    "schedule": [
```

```
            "000000000000000000000000000000000000000000000000",
            "000000000000000000000000000000000000000000000000",
            "000000000000001111111111000000000000000000000000",
            "000000000000000000001111111111110000000000000000",
            "000011111111111111111100000000000000000000000000",
            "000000000000000000000000000000000000000000000000",
            "000000000000000000000000000000000000000000000000"
        ],
        "set_arm_disarm": false,
        "name": "Schedule Source",
        "dependents": [ ],
        "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`schedule`	String Array	Array of seven strings, forty-eight characters in size. Each character is a binary digit	Array of 7, 48 binary character strings
`set_arm_disarm`	Boolean	A boolean specifying whether this observable also sets the armed state of VCAcore	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables. I

4.5.6 System

The 'System' observable is a channel-independent observable which generates an event when the specified `resource_type` passes a set `threshold`. Events will continue to send whilst the set `threshold` is met each time the `min_interval` duration has passed if `repeat_events` is `true`. Please use the method mentioned earlier to add this element. A sample 'System' observable is shown below:

```
{
    "typename": "vca.observable.System",
    "resource_type": "Gpu Utilisation",
    "threshold": 0,
    "min_interval": 60000,
    "repeat_events": true,
    "name": "System Alarm Source",
    "dependents": [ ],
    "triggers_action": true
}
```

In addition to the common properties described above, below is a list of properties specific to this observable:

Property	Type	Description	Possible values
`resource_type`	String	String defining the system resource to monitor	Only `Gpu Utilisation`
`threshold`	Unsigned Integer	Percentage threshold that must be reached to trigger an event	Unsigned integer between 0 and 1
`min_interval`	Unsigned Integer	The interval value between each triggered event, in milliseconds	Any unsigned integer
`repeat_events`	Boolean	A boolean specifying whether this observable can repeatedly trigger events	`true` or `false`

For a list of properties common to all observables, please see the General Concepts section on Observables.

Chapter 5

REST API Zones, Actions and VCAcore Status

5.1 Zones

5.1.1 Adding a Zone

To add a zone, send a POST request to the `/api/zones` endpoint. Unlike other elements, a zone does not have a `typename` property, so a zone may be added by sending a single POST request to the endpoint mentioned above with the correct payload. A sample zone is given below:

```
{
    "name": "Zone 0",
    "channel": 4,
    "points": [
        {
            "x": 20585,
            "y": 17374
        },
        {
            "x": 23368,
            "y": 51893
        }
    ],
    "colour": {
        "r": 252,
        "g": 175,
        "b": 62
    },
    "polygon": false,
    "detection": true
}
```

The properties of a zone object are given below:

Property	Type	Description	Possible values
`name`	String	A user-defined name for this zone	Any string, can be empty

Property	Type	Description	Possible values
`channel`	Unsigned integer	The identifier of the channel this zone is associated with	Any unsigned integer
`points`	Array of objects	An array of point objects	A point object array that has a minimum of two points
`colour`	Object	A colour object specifying the colour of the zone, without the alpha(`"a"`) property	Any valid colour object, with no alpha property
`polygon`	Boolean	An boolean specifying whether this zone should be treated as a polygon (`true`) or a line (`false`)	`true` or `false`
`detectOn`	Boolean	An boolean specifying whether detection is enabled on this zone	`true` or `false`

5.2 Actions

Actions are objects that represent an operation that can be performed by the application. Actions can be linked to observables so that when an observable fires an event, the event causes the action to be triggered.

5.2.1 General Concepts

Actions can be added by sending a POST request to the `/api/actions` endpoint. Please use the method mentioned earlier to add this element. A sample TCP action is shown below:

```
{
    "typename": "vca.action.Tcp",
    "uri": "192.168.5.3",
    "port": 0,
    "body": "{ "event_name":{{name}} } ",
    "name": "TcpActionAddTest",
    "observables": [
        5,
        4
    ],
    "always_trigger": false
}
```

The following is a list of the properties common to all actions:

Property	Type	Description	Possible values
`name`	String	A user-defined name for this action	Any string, can be empty
`observables`	Array of indices	An array of indexes of observables to associated with this action	Any array of unsigned integers (can be empty)

Property	Type	Description	Possible values
`always_trigger`	Boolean	A boolean specifying whether this action triggers irrespective of the armed state of the device	`true` or `false`

What follows is a list of all actions supported in VCAcore.

5.2.2 TCP Action

A TCP action sends data to a user-specified endpoint. Please use the method mentioned earlier to add this element. A sample TCP action is shown below:

```
{
    "typename": "vca.action.Tcp",
    "uri": "192.168.5.3",
    "port": 0,
    "body": "{ "event_name":{{name}} } ",
    "name": "TcpActionAddTest",
    "observables": [],
    "always_trigger": false
}
```

The following is a list of properties specific to the TCP action:

Property	Type	Description	Possible values
`uri`	String	The URI of the TCP server to send the data to	Any string, can be empty
`port`	Unsigned integer	The port to connect to on the TCP server	Any unsigned integer between 0 and 63353 (inclusive)
`body`	String	Content of the action body, e.g VCAcore metadata tokens, XML, JSON etc as required	Any string, can be empty

For a list of properties common to all actions, please refer to the General Concepts section on actions.

5.2.3 HTTP Action

An HTTP action sends an HTTP request to a user-specified endpoint. Please use the method mentioned earlier to add this element. A sample HTTP action is shown below:

```
{
    "typename": "vca.action.Http",
    "method": "GET",
    "uri": "http://192.168.1.60",
```

```
    "port": 0,
    "headers": "Content-Type: application/json",
    "body": "{"event_name":{{name}}, }",
    "authentication": false,
    "username": "",
    "password": "",
    "send_snapshot": false,
    "pre_snapshots": 1,
    "post_snapshots": 1,
    "jpeg_quality": "average",
    "interval": 10,
    "multipart_name": "vca",
    "name": "",
    "observables": [],
    "always_trigger": false
}
```

The following is a list of properties specific to the HTTP action:

Property	Type	Description	Possible values
`method`	String	The HTTP verb to use when sending the request	One of the following: `"GET"`, `"POST"`, `"PUT"`, `"DELETE"`, `"HEAD"`
`uri`	String	The URI of the HTTP Request	Any string, can be empty
`port`	Unsigned integer	The port to connect to on the TCP server	Any unsigned integer between 0 and 63353 (inclusive)
`headers`	String	Content of the HTTP action header	Any string, can be empty
`body`	String	Content of the action body, e.g VCAcore metadata tokens, XML, JSON etc as required	Any string, can be empty
`authentication`	Boolean	A boolean specifying whether to enable authentication	`true` or `false`
`username`	String	The username to use when authentication is enabled	Any string, can be empty
`password`	String	The password to use when authentication is enabled	Any string, can be empty
`send_snapshot`	Boolean	A boolean value specifying whether to send snapshots with the request	`true` or `false`
`pre_snapshots`	Unsigned integer	The number of pre-event snapshots to send with the request	Any unsigned integer between 0 and 10 (inclusive)
`post_snapshots`	Unsigned integer	The number of post-event snapshots to send with the request	Any unsigned integer between 0 and 5 (inclusive)
`jpeg_quality`	String	The quality of the JPEG snapshots that are sent with the request	One of the following: `"worst"`, `"low"`, `"average"`, `"good"`, `"best"`
`interval`	Unsigned Integer	The time interval between snapshots (in milliseconds)	Any unsigned integer between 0 and 1000 (inclusive)

Property	Type	Description	Possible values
multipart_name	String	The text assigned as multipart name in the request	Letters, Numbers, Dashes, Underscores and Square Brackets only, cannot be empty

Please note multipart_name will need to be reflected in any scripts that handle this request, for example in php; this would be by using $_FILES['vca'] where vca is the string set in the multipart_name field

For a list of properties common to all actions, please refer to the General Concepts section on actions.

5.2.4 Email Action

An Email action sends an email in a user-specified format. Please use the method mentioned earlier to add this element. A sample Email action is shown below:

```
{
    "typename": "vca.action.Email",
    "server": "",
    "port": 0,
    "encryption": "none",
    "username": "",
    "password": "",
    "enable_authentication": false,
    "verify_host_certificate": false,
    "to": "",
    "cc": "",
    "bcc": "",
    "from": "",
    "subject": "{{type.string}} Cam ID {{#Channel}}{{id}}{{/Channel}}",
    "format": "custom",
    "body": "{{name}} triggered with at {{start.iso8601}}",
    "send_snapshot": false,
    "pre_snapshots": 0,
    "post_snapshots": 0,
    "jpeg_quality": "average",
    "interval": 0,
    "name": "",
    "observables": [],
    "always_trigger": false
}
```

Property	Type	Description	Possible values
server	String	The URI of the SMTP Server to use for sending the email	Any string, can be empty

Property	Type	Description	Possible values
`port`	Unsigned integer	The port to connect to on the TCP server	Any unsigned integer between 0 and 63353 (inclusive)
`encryption`	String	The encryption method to use when connecting to the server. Both unencrypted and STARTTLS methods are supported	Either `"none"` or `"start_tls"`
`username`	String	The username to use when authentication is enabled	Any string, can be empty
`password`	String	The password to use when authentication is enabled	Any string, can be empty
`enable_authentication`	Boolean	A boolean value specifying whether authentication should be enabled	`true` or `false`
`verify_host_certificate`	Boolean	A boolean value specifying whether the server's SSL certificate should be verified	`true` or `false`
`to`	String	The 'To' field of the email	Any string, can be empty
`cc`	String	The 'CC' field of the email	Any string, can be empty
`bcc`	String	The 'BCC' field of the email	Any string, can be empty
`from`	String	The 'From' field of the email	Any string, can be empty
`subject`	String	Content of the email subject. Can use VCAcore metadata tokens	Any string, can be empty
`format`	String	The string specifying the format of the email body	Must always be set to `custom`
`body`	String	Content of the action body, e.g VCAcore metadata tokens, XML, JSON etc as required	Any string, can be empty
`send_snapshot`	Boolean	A boolean value specifying whether to send snapshots with the request	`true` or `false`
`pre_snapshots`	Unsigned integer	The number of pre-event snapshots to send with the request	Any unsigned integer between 0 and 10 (inclusive)
`post_snapshots`	Unsigned integer	The number of post-event snapshots to send with the request	Any unsigned integer between 0 and 5 (inclusive)
`jpeg_quality`	String	The quality of the JPEG snapshots that are sent with the request	One of the following: `"worst"`, `"low"`, `"average"`, `"good"`, `"best"`
`interval`	Unsigned Integer	The time interval between snapshots (in milliseconds)	Any unsigned integer between 0 and 1000 (inclusive)

For a list of properties common to all actions, please refer to the General Concepts section on actions.

5.2.5 Arm Action

An 'Arm' action sets the state of the application to 'armed' When the application is armed, all actions fire normally. Please use the method mentioned earlier to add this element. A sample 'Arm' action is

shown below:

```
{
    "typename": "vca.action.Arm",
    "name": "",
    "observables": [],
    "always_trigger": false
}
```

The 'Arm' action does not have any specific properties. For a list of properties common to all actions, please refer to the General Concepts section on actions.

5.2.6 Disarm Action

A 'Disarm' action sets the state of the application to 'disarmed' When the application is disarmed, only actions with `always_trigger` set to `true` will fire. Other actions will be prevented from firing. Please use the method mentioned earlier to add this element. A sample 'Disarm' action is shown below:

```
{
    "typename": "vca.action.Disarm",
    "name": "",
    "observables": [],
    "always_trigger": false
}
```

The 'Disarm' action does not have any specific properties. For a list of properties common to all actions, please refer to the General Concepts section on actions.

5.3 Status

5.3.1 User Credentials

To change the current password, a `POST` request must be sent to `/api/auth/user/admin` with the following data:

```
{
    "current": "CURRENT_PASSWORD_MD5_HASH",
    "password": "NEW_PASSWORD_MD5_HASH"
}
```

The password hashes are computed as follows:

```
MD5("admin:vcatechnology.com:" + password)
```

Note that all following HTTP requests will need to be made with the updated password.

5.3.2 Armed State

The current armed state of VCAcore can be retrieved by sending a `GET` request to `/api/arm`

The armed state of VCAcore can also be set via the API by sending a POST request to `/api/arm` with a payload containing either:

```
{
    true
}
```

to arm VCAcore or:

```
{
    false
}
```

to disarm VCAcore.

Chapter 6

Metadata APIs

VCAcore supports two methods to access metadata produced by the various algorithms running on a channel.

- Server-Sent Events (SSE) stream
- RTSP Metadata stream

Both methods expose VCAcore's metadata in JSON format, a detailed description of the data format is outlined below.

6.1 Server-Sent Events (SSE) Endpoints

6.1.1 Channel

The SSE metadata API endpoint for a channel is split into two categories, `objects` and `events` each of which generates a separate message. For a comprehensive breakdown of the returned data formats please see Metadata Format.

It is possible to filter which category of message is sent by adding query parameters to the SSE endpoint.

For `event` messages only:

`http://SERVER_IP:PORT/metadata/CHANNEL_ID?events=1`

For `object` messages only:

`http://SERVER_IP:PORT/metadata/CHANNEL_ID?objects=1`

If neither parameter is specified, all messages will be generated and sent.

6.1.2 System Statistics

The SSE metadata API endpoint to retrieve system statistics is:

`http://SERVER_IP:PORT/api/system-stats`

System information covering system uptime, current processor load, graphics card information and load and memory load can be retrieved. An example response is given below:


```
{
    "cpu": {
        "process": 0.0506757,
        "processes": [
            0.0506757
        ],
        "temperature": 0,
        "temperatures": [],
        "total": 0.0473899,
        "totals": [
            0.0675676,
            0.0945946,
            0,
            0.0273973
        ]
    },
    "gpus": [
        {
            "device": {
                "address_id": 0,
                "bus_id": 1,
                "product": "Device 1f02",
                "vendor": "NVIDIA Corporation"
            },
            "memory": {
                "available": 8149336064,
                "total": 8366784512,
                "used": 217448448
            },
            "temperature": 49,
            "utilisation": 0
        }
    ],
    "memory": {
        "physical": {
            "in_use": 4253986816,
            "process": 216981504,
            "total": 16673214464
        },
        "virtual": {
            "in_use": 4253986816,
            "process": 1548251136,
            "total": 18820694016
        }
    },
    "time": {
```

```
        "now": "2020-07-20T14:12:25.246948405+01:00"
    },
    "uptime": {
        "process": 2741172,
        "system": 277573990
    }
}
```

Please note the `process` and `processes` values belonging to `cpu` will not be populated when VCAcore is run on Windows and will be set to 0.

6.1.3 Uptime

VCAcore has an uptime service which will return the time (ms) that the VCAcore process has been active (the counter is reset each time the VCAcore process is reset) and the total time the system has been active.

`http://SERVER_IP:PORT/api/uptime`

Example response below:

```
{
    "process": 1987591,
    "system": 276820400
}
```

Please note this endpoint is now depreciated and replaced by `system-stats` which includes both uptime information as well as system performance information.

6.1.4 SSE Code Sample

Below is example Python code demonstrating how the channel SSE metadata stream, can be consumed:

```
#!/usr/bin/python
# The user must install the sseclient and requests packages using pip

from sseclient import SSEClient
import json
import requests

def do_something_useful(message):
    metadata = json.loads(message.data)
    print('Received metadata event')
    print(json.dumps(metadata, indent=4, sort_keys=True))

if __name__ == '__main__':
    SERVER_IP = '192.168.1.99'
    PORT = '80'
```

```
CHANNEL_ID = 0
messages = SSEClient('http://' + SERVER_IP + ':' + PORT + '/metadata/' + str(CHAN
                     auth=requests.auth.HTTPDigestAuth('admin', 'admin'))
for msg in messages:
    do_something_useful(msg)
```

6.2 RTSP Metadata Stream

In addition to a channel's RTSP video stream, the metadata for that channel is also encoded into an RTSP metadata stream.

The RTSP metadata endpoint for a channel is the same as the RTSP URL:

`rtsp://SERVER_IP:RTSP_PORT/channels/CHANNEL_ID`

6.2.1 RTSP Metadata Stream Code Sample

Example code in Python demonstrating how the RTSP metadata stream, can be consumed is available for download here:

RTSP Metadata Python Example

Chapter 7

Metadata Format

7.1 SSE Metadata Format

For a particular frame for a given channel, SSE metadata messages can be created in two categories. A message can either contain data on the `events` generated by the rules configured on the channel, example response below:


```
{
    "2020-10-30T12:43:42.035830016Z": [
        {
            "typename":"vca.meta.data.Event",
            "id":12841,
            "name":"Deep Learning Presence 24",
            "type":"Presence",
            "category":"analytics",
            "start":"2020-10-30T12:43:42.035830016Z",
            "end":"2020-10-30T12:43:42.035830016Z",
            "duplicate":true,
            "objects":[
                {
                    "typename":"vca.meta.data.Channel",
                    "id":2
                },
                {
                    "typename":"vca.meta.data.Zone",
                    "id":4,
                    "name":"Car Park",
                    "channel":2,
                    "colour":{
                        "r":114,
                        "g":159,
                        "b":207,
                        "a":255
                    },
                    "detection":"on",
```

```
                "type":"polygon",
                "outline":[
                    {
                        "x":0,
                        "y":26634
                    },
                    {
                        "x":34476,
                        "y":26634
                    },
                    {
                        "x":34476,
                        "y":23035
                    },
                    {
                        "x":0,
                        "y":23035
                    }
                ]
            }
        ]
    },
    {
        "typename": "vca.meta.data.Event",
        "category": "analytics",
        "type": "count",
        "duplicate": false,
        "start": "2020-10-30T12:42:59.835830016Z",
        "end": "2020-10-30T12:43:42.035830016Z",
        "id": 1247782,
        "name": "South to North",
        "objects": [
            {
                "id": 79,
                "name": "South to North",
                "position": {
                    "x": 54195,
                    "y": 9934
                },
                "typename": "vca.meta.data.count.Value",
                "value": 105
            },
            {
                "id": 79,
                "typename": "vca.meta.data.Observable"
            },
            {
                "id": 0,
```

```
                    "typename": "vca.meta.data.Channel"
                }
            ]
        }
    ]
}
```

Alternatively, a message can contain data on the `objects` tracked in the channel, example response below:

```
{
    "2018-10-02T16:51:55.782845060+01:00":[
        {
            "typename":"vca.meta.data.Object",
            "id":2128,
            "outline":[
                {
                    "x":12910,
                    "y":33733
                },
                {
                    "x":27169,
                    "y":33733
                },
                {
                    "x":12910,
                    "y":65535
                },
                {
                    "x":27169,
                    "y":65535
                }
            ],
            "width":14259,
            "height":31802,
            "meta":[
                {
                    "typename":"vca.meta.data.object.GroundPoint",
                    "value":{
                        "x":20040,
                        "y":65535
                    }
                }
            ]
        }
    ]
}
```

All messages contain a JSON object and each message type will only be generated if there is data to send, e.g. if there are no tracked objects in the scene then there will be no message containing object

data.

Each JSON object has the ISO8601 timestamp of that particular frame as its only property. The value associated with that property is an array of objects, details of which are outlined in Metadata Format.

7.2 RTSP Metadata Stream Format

Each RTSP metadata stream message is a JSON object for a particular frame for a given channel.

Each JSON object contains a **timestamp** and **objects** property. The **timestamp** property has an ISO8601 timestamp value for the given frame. The **objects** property contains an array of objects, details of which are outlined in Metadata Format..


```
{
    "timestamp":"2020-09-08T16:52:34.011858944+01:00",
    "objects":[
        {
            "typename":"vca.meta.data.Object",
            "id":1380,
            "outline":[
                {
                    "x":10646,
                    "y":23724
                },
                {
                    "x":19021,
                    "y":23724
                },
                {
                    "x":10646,
                    "y":29627
                },
                {
                    "x":19021,
                    "y":29627
                }
            ],
            "width":8375,
            "height":5903,
            "meta":[
                {
                    "typename":"vca.meta.data.object.GroundPoint",
                    "value":{
                        "x":14833,
                        "y":29627
                    }
                },
                {
```

```
                "typename":"vca.meta.data.classification.Confidence",
                "class":"vehicle",
                "confidence":0.8999999761581421,
                "object_id":1380
            }
        ]
    },
    {
        "typename":"vca.meta.data.Event",
        "id":12841,
        "name":"Deep Learning Presence 24",
        "type":"Presence",
        "category":"analytics",
        "start":"2020-09-08T16:52:31.677858944+01:00",
        "end":"2020-09-08T16:52:34.011858944+01:00",
        "duplicate":true,
        "objects":[
            {
                "typename":"vca.meta.data.Channel",
                "id":2
            },
            {
                "typename":"vca.meta.data.Zone",
                "id":4,
                "name":"Car Park",
                "channel":2,
                "colour":{
                    "r":114,
                    "g":159,
                    "b":207,
                    "a":255
                },
                "detection":"on",
                "type":"polygon",
                "outline":[
                    {
                        "x":0,
                        "y":26634
                    },
                    {
                        "x":34476,
                        "y":26634
                    },
                    {
                        "x":34476,
                        "y":23035
                    },
                    {
```

```
                        "x":0,
                        "y":23035
                    }
                ]
            }
        ]
    }
]
}
```

7.3 Metadata Objects

Each metadata object has a `typename` string property enabling its identification. Below is a list of object types that may be found in the metadata API.

Note: Where coordinate data is present, VCAcore's default is to provide this data as a 16-bit integer, with 0-65535 representing the range from 0-1 in the frame. However, this upper limit is customisable.

7.3.1 Counter Value

The object representation of a line counter event. This object represents a change in the value of the counter.

Example:

```
{
    "typename": "vca.meta.data.count.Value",
    "id": 5,
    "name": "my counter",
    "value": 0
}
```

Property	Type	Description	Possible values
`id`	Number	The id of the counter this count value is associated with	An unsigned integer
`name`	String	The name of the counter this count value is associated with	Any string, can be empty
`value`	Number	The value of the counter	A signed integer

7.3.2 Counting Line

The object representation of a line counter event. This object contains data pertaining to the object which has crossed the line.

Example:

```
{
    "typename": "vca.meta.data.count.Line",
    "rule_id": 4,
    "width": 20,
    "position": 3,
    "count": 2,
    "direction": false
}
```

Property	Type	Description	Possible values
`rule_id`	Number	The rule id of this counting line associated with this event	An unsigned integer
`width`	Number	The width of the object which crossed the line	An unsigned, 16-bit integer
`position`	Number	The position of object on the line	An unsigned, 16-bit integer
`count`	Number	The number of objects crossing the line in this event	An unsigned integer
`direction`	Boolean	The direction in which the object has crossed the line, with Left = false and Right = true	true/false

7.3.3 Scene Learning

An object indicating scene learning is in progress.

Example:

```
{
    "typename": "vca.meta.data.Learning"
}
```

7.3.4 Tamper

An object indicating tamper is in progress.

Example:

```
{
    "typename": "vca.meta.data.Tampered"
}
```

7.3.5 Tracked Objects

An object representing a tracked object.

```
{
    "id": 56,
```

```
"width": 4726,
"height": 4914,
"meta": [
    {
        "typename": "vca.meta.data.object.GroundPoint",
        "value": {
            "x": 45686,
            "y": 25667
        }
    },
    {
        "class": "person",
        "confidence": 0.875215470790863,
        "object_id": 56,
        "typename": "vca.meta.data.classification.Confidence"
    },
    {
        "typename": "vca.meta.data.ColourSignature",
        "colours": [
            {
                "colour_name": "Black",
                "colour_value": {
                    "r": 0,
                    "g": 0,
                    "b": 0
                },
                "proportion": 0.95555555820465088
            },
            {
                "colour_name": "..."
            }
        ]
    }
],
"outline": [
    {
        "x": 43323,
        "y": 20753
    },
    {
        "x": 48049,
        "y": 20753
    },
    {
        "x": 43323,
        "y": 25667
    },
    {
```

```
            "x": 48049,
            "y": 25667
        }
    ],
    "typename": "vca.meta.data.Object"
}
```

Property	Type	Description	Possible values
`id`	Number	The id of this tracked object	An unsigned integer
`height`	Number	The height of the objects bounding box outline	An unsigned integer
`width`	Number	The width of the objects bounding box outline	An unsigned integer
`meta`	Array	An array of additional metadata objects	An array of valid metadata objects
`outline`	Array	An array of objects with x/y coordinates defining the bounding box for the object	For both x and y, valid values are any unsigned 16-bit integer

7.3.6 Ground Point

The ground point of a tracked object.

Example:

```
{
    "typename": "vca.meta.data.object.GroundPoint",
    "value": {
        "x": 45686,
        "y": 25667
    }
}
```

Property	Type	Description	Possible values
`value`	Object	An object with x/y coordinates of the ground point	For both x and y, valid values are any unsigned 16-bit integer

7.3.7 Channel

The source channel of an object.

Example:

```
{
    "typename": "vca.meta.data.Channel",
    "id": 1
}
```

Property	Type	Description	Possible values
id	Number	The id of this channel	An unsigned integer

7.3.8 Observable

The source observable related to an object.

Example:

```
{
    "typename": "vca.meta.data.Observable",
    "id": 1
}
```

Property	Type	Description	Possible values
id	Number	The id of this observable	An unsigned integer

7.3.9 Zone

The zone data object.

Example:

```
{
    "typename":"vca.meta.data.Zone",
    "id":4,
    "name":"Car Park",
    "channel":2,
    "colour":{
        "r":114,
        "g":159,
        "b":207,
        "a":255
    },
    "detection":"on",
    "type":"polygon",
    "outline":[
        {
            "x":0,
            "y":26634
        },
        {
            "x":34476,
            "y":26634
        },
        {
```

```
            "x":34476,
            "y":23035
        },
        {
            "x":0,
            "y":23035
        }
    ]
}
```

Property	Type	Description	Possible values
`id`	Number	The id of this zone in the configuration	An unsigned integer
`name`	String	The name of the zone	Any string, can be empty
`channel`	Number	The id of this channel the zone is configured on	An unsigned integer
`colour`	Object	A colour object specifying the colour of the zone, without the alpha("a") property	Any valid colour object, with no alpha property
`detection`	Boolean	An boolean specifying whether detection is enabled on this zone	`true` or `false`
`type`	String	A string specifying whether this zone is a `polygon` or a `line`	`polygon` or `line`
`outline`	Array of objects	An array of point objects	A point object array that has a minimum of two points

7.3.10 Confidence Classification

The class name (e.g. `person`) and confidence provided by a classification algorithm (i.e. the deep learning filter). The available class names are defined by the classification algorithm. The confidence value indicates now likely that classification is to be correct. Confidence classification metadata will be added to the `meta` array of a tracked object when available.

Class string	Description
`person`	A person, or tracked object with a person present (e.g bicycle)
`vehicle`	A Car, Van, Bus or Truck
`background`	Any object which is not included in the previous classes

Example:

```
{
    "class": "person",
    "confidence": 0.875215470790863,
    "object_id": 22,
    "typename": "vca.meta.data.classification.Confidence"
```

```
}
```

Property	Type	Description	Possible values
`class`	String	Classification name	One of a set list of names as defined by the classification algorithm
`confidence`	Float	The model's confidence that the object class is correct	0 - 1
`object_id`	Number	The id of this tracked object	An unsigned integer

7.3.11 Colour Signature

The break down of pixel colours found in a given tracked object's bounding box. The number of colours a pixel can be grouped into is fixed, however the number of colours retuned in the colour signature metadata object may change. Colour signature metadata will be added to the `meta` array of a tracked object when available.

Example:


```
{
    "typename": "vca.meta.data.ColourSignature",
    "colours": [
        {
            "colour_name": "Black",
            "colour_value": {
                "r": 0,
                "g": 0,
                "b": 0
            },
            "proportion": 0.95555555820465088
        },
        {
            "colour_name": "Brown",
            "colour_value": {
                "r": 150,
                "g": 75,
                "b": 0
            },
            "proportion": 0
        },
        {
            "colour_name": "Grey",
            "colour_value": {
                "r": 100,
                "g": 100,
                "b": 100
            },
            "proportion": 0.029468599706888199
```

```
},
{
    "colour_name": "Blue",
    "colour_value": {
        "r": 0,
        "g": 0,
        "b": 200
    },
    "proportion": 0
},
{
    "colour_name": "Green",
    "colour_value": {
        "r": 0,
        "g": 150,
        "b": 0
    },
    "proportion": 0
},
{
    "colour_name": "Cyan",
    "colour_value": {
       "r": 0,
       "g": 255,
       "b": 255
    },
    "proportion": 0
},
{
    "colour_name": "Red",
    "colour_value": {
        "r": 255,
        "g": 0,
        "b": 0
    },
    "proportion": 0
},
{
    "colour_name": "Magenta",
    "colour_value": {
        "r": 200,
        "g": 0,
        "b": 200
    },
    "proportion": 0
},
{
    "colour_name": "Yellow",
```

```
            "colour_value": {
                "r": 255,
                "g": 255,
                "b": 0
            },
            "proportion": 0
        },
        {
            "colour_name": "White",
            "colour_value": {
                "r": 255,
                "g": 255,
                "b": 255
            },
            "proportion": 0.014975845813751221
        }
    ]
}
```


Property	Type	Description	Possible values
colours	Array	An array of colour names, their RGB values and the proportion of all pixels that have been grouped into this colour category for this object	A fixed array of colour metadata objects

7.3.12 Body Part

Describes a body part (skeletal joint) position and the detection confidence of the algorithm.

Example:


```
{
    "position": {
        "x": 43386,
        "y": 3561
    },
    "confidence": 0.8080910444259644,
    "typename": "vca.meta.data.pose.BodyPart"
}
```


Property	Type	Description	Possible values
position	Object	Position in x and y of the detected body part	single point in terms of x and y
confidence	Number	Detection and classification confidence of the detection algorithm	0 - 1

7.3.13 Deep Learning People Tracker Skeleton Data

A map of detected body parts (skeleton joints). The `map` property will return all body parts detected for the attached object. Current implementation provides 18 possible body part types. In any given `map` there could be less than 18. The list of detectable body parts is subject to change.

Below is a list of possible body part body parts:

Body Part key string
nose
left_eye
right_eye
left_ear
right_ear
neck
left_shoulder
right_shoulder
left_elbow
right_elbow
left_hand
right_hand
left_hip
right_hip
left_knee
right_knee
left_ankle
right_ankle

Example:


```
{
    "map": [
        {
            "key": "nose",
            "value": {
                "position": {
                    "x": 43386,
                    "y": 3561
                },
                "score": 0.8080910444259644,
                "typename": "vca.meta.data.pose.BodyPart"
            }
        },
        {
            "key": "neck",
            "value": {
                "position": {
                    "x": 43690,
```

```
                "y": 5698
            },
            "score": 0.7894839644432068,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_shoulder",
        "value": {
            "position": {
                "x": 41262,
                "y": 4986
            },
            "score": 0.7354756593704224,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_elbow",
        "value": {
            "position": {
                "x": 39745,
                "y": 12109
            },
            "score": 0.7663553953170776,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_wrist",
        "value": {
            "position": {
                "x": 38228,
                "y": 18520
            },
            "score": 0.8059698939323425,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "left_shoulder",
        "value": {
            "position": {
                "x": 46724,
                "y": 6054
            },
            "score": 0.7997663617134094,
            "typename": "vca.meta.data.pose.BodyPart"
```

```
        }
    },
    {
        "key": "left_elbow",
        "value": {
            "position": {
                "x": 48241,
                "y": 12822
            },
            "score": 0.7496799826622009,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "left_wrist",
        "value": {
            "position": {
                "x": 48241,
                "y": 18520
            },
            "score": 0.7150058746337891,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_hip",
        "value": {
            "position": {
                "x": 41262,
                "y": 18164
            },
            "score": 0.6697762608528137,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_knee",
        "value": {
            "position": {
                "x": 41262,
                "y": 26000
            },
            "score": 0.7108161449432373,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_ankle",
```

```
        "value": {
            "position": {
                "x": 40049,
                "y": 32767
            },
            "score": 0.5868609547615051,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "left_hip",
        "value": {
            "position": {
                "x": 45207,
                "y": 18520
            },
            "score": 0.689977765083313,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "left_knee",
        "value": {
            "position": {
                "x": 44600,
                "y": 25644
            },
            "score": 0.7125816345214844,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "left_ankle",
        "value": {
            "position": {
                "x": 42476,
                "y": 31698
            },
            "score": 0.6361902952194214,
            "typename": "vca.meta.data.pose.BodyPart"
        }
    },
    {
        "key": "right_eye",
        "value": {
            "position": {
                "x": 42779,
                "y": 2849
```

```
                },
                "score": 0.7985154986381531,
                "typename": "vca.meta.data.pose.BodyPart"
            }
        },
        {
            "key": "left_eye",
            "value": {
                "position": {
                    "x": 43993,
                    "y": 2849
                },
                "score": 0.8068280220031738,
                "typename": "vca.meta.data.pose.BodyPart"
            }
        },
        {
            "key": "right_ear",
            "value": {
                "position": {
                    "x": 42172,
                    "y": 2493
                },
                "score": 0.33054694533348083,
                "typename": "vca.meta.data.pose.BodyPart"
            }
        },
        {
            "key": "left_ear",
            "value": {
                "position": {
                    "x": 44903,
                    "y": 2849
                },
                "score": 0.6982870697975159,
                "typename": "vca.meta.data.pose.BodyPart"
            }
        }
    ],
    "score": 1.5487611293792725,
    "typename": "vca.meta.data.Pose"
}
```


Property	Type	Description	Possible values
`map`	Array	An array of body parts keyed against a fixed list of body part types	An array of body part metadata objects
`confidence`	Number	Mean confidence value from all body part objects in `map`	0 - 1

7.3.14 Object History

An object containing the tracking history of a tracked object in the form of the ground points keyed by timestamp.

Example:


```
{
    "map": [
        {
            "key": "1970-01-01T01:00:00.000500000+01:00",
            "value": {
                "typename": "vca.meta.data.object.GroundPoint",
                "value": {
                    "x": 17900,
                    "y": 33646
                }
            }
        },
        {
            "key": "1970-01-01T01:00:01.000000000+01:00",
            "value": {
                "typename": "vca.meta.data.object.GroundPoint",
                "value": {
                    "x": 17506,
                    "y": 33658
                }
            }
        }
    ],
    "object_id": 147,
    "typename": "vca.meta.data.object.History"
},
```


Property	Type	Description	Possible values
`map`	Array	An array of tracked objects	An array of valid tracked objects
`object_id`	Number	The id of this tracked object	An unsigned integer

Please note the `object_id` will match the `id` value of the host `"vca.meta.data.Object"`.

7.3.15 Event

An object representing a metadata event


```
{
    "typename":"vca.meta.data.Event",
    "id":12841,
    "name":"Deep Learning Presence 24",
```

```
"type":"Presence",
"category":"analytics",
"start":"2020-09-08T16:52:31.677858944+01:00",
"end":"2020-09-08T16:52:34.011858944+01:00",
"duplicate":true,
"objects":[
    {
        "typename":"vca.meta.data.Channel",
        "id":2
    },
    {
        "typename":"vca.meta.data.Zone",
        "id":4,
        "name":"Car Park",
        "channel":2,
        "colour":{
            "r":114,
            "g":159,
            "b":207,
            "a":255
        },
        "detection":"on",
        "type":"polygon",
        "outline":[
            {
                "x":0,
                "y":26634
            },
            {
                "x":34476,
                "y":26634
            },
            {
                "x":34476,
                "y":23035
            },
            {
                "x":0,
                "y":23035
            }
        ]
    },
    {
        "id": 105,
        "width": 728,
        "height": 3268,
        "meta": [
            {
```

```
                    "typename": "vca.meta.data.object.GroundPoint",
                    "value": {
                        "x": 44229,
                        "y": 24298
                    }
                }
            ],
            "outline": [
                {
                    "x": 43865,
                    "y": 21030
                },
                {
                    "x": 44593,
                    "y": 21030
                },
                {
                    "x": 43865,
                    "y": 24298
                },
                {
                    "x": 44593,
                    "y": 24298
                }
            ],
            "typename": "vca.meta.data.Object"
        }
    ]
}
```


Property	Type	Description	Possible values
`id`	Number	The id of this event	An unsigned integer
`name`	String	The name of the rule that triggered this event	Any string (can be empty)
`type`	String	The type of this event	A valid event type. See below for a list of types
`category`	String	The category of this event	A valid event category. See below for a list of categories
`start`	String	The start timestamp of this event	A valid ISO8601 timestamp
`end`	String	The end timestamp of this event	A valid ISO8601 timestamp
`duplicate`	Boolean	Indicates a persistent event has fired before	true/false
`objects`	Array	The tracked objects associated with this event	A valid array of tracked objects

Below is a list of possible event categories:

Event category string	Description
`analytics`	An event generated by the VCA5 Analytics engine
`loss-of-signal`	An event indicating the loss of video signal from a camera

Below is a list of possible event types:

Event type string	Description
`presence`	A presence event
`enter`	An enter event
`exit`	An exit event
`appear`	An appear event
`abandoned`	An abandoned event
`disappear`	A disappear event
`stopped`	A stopped event
`dwell`	A dwell event
`direction`	A 'direction' event
`speed`	A speed event
`tailgating`	A tailgating event
`linecountera`	A line counter crossing event (left direction)
`linecounterb`	A line counter crossing event (right direction)

Chapter 8

ONVIF

VCAcore supports a number of ONVIF features. Note that the same ONVIF features are present in VCAserver and VCAbridge.

Device Endpoints:

Endpoint	Description
`GetServices`	Get the list of supported services
`GetServiceCapabilities`	Get the capabilities for all supported services
`GetDeviceInformation`	Get basic information such as manufacturer and version number
`GetScopes`	Get the list of scopes the application supports
`GetUsers`	Get the list of users. Returns 'admin' which is the only permitted user
`SetUser`	Set the password of a specific user. Please note the limitation above
`GetCapabilities`	Get the capabilities of the application

It is important to note that even though the `GetUsers` and `SetUser` endpoints are supported, neither user addition (`CreateUsers`) nor deletion (`DeleteUsers`) are supported.

Event endpoints:

Endpoint	Description
`GetEventProperties`	Get the properties of all events
`CreatePullPointSubscription`	Create a subscription for pulling events. Returns a subscription reference
`PullMessages`	Pull the list of events, providing a subscription reference
`Unsubscribe`	Unsubscribe a specific subscription reference

Please note that the `PullMessages` endpoint returns a list of the most recent events generated by the application. The following fields are currently included for each event.

Property	Description
`start_time`	The start time of the event
`end_time`	The end time of the event
`id`	The id of the event
`name`	The user-specified name of the event
`type`	The type of the event
`category`	The category of the event

Note: * The `start_time` and `end_time` properties of the events are specified as nanoseconds since the Unix epoch. * The remaining properties are identical to the ones specified for events in the Metadata API section.

Please refer to the ONVIF documentation for more information.