Bosch Security Systems

Technical Note
VCA Task Script Language FW 6.30

BOSCH
Invented for life

June 14th, 2016

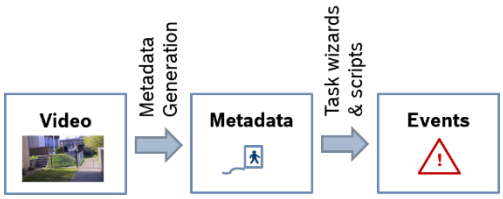
VCA Task Script Language

This technical note gives a short introduction into the VCA task script language as well as script examples explained in detail.

What is the VCA task script language?

VCA stands for video content analysis. The VCA task script language

- describes every predefined and configured Intelligent Video Analytics, Essential Video Analytics and MOTION+ task and task wizard
- can combine tasks to form more complex ones
- can NOT configure the metadata generation



When to use the VCA task scripts?

The VCA task scripts are used implicitly whenever an Intelligent Video Analytics, Essential Video Analytics or MOTION+ task is configured via the GUI. However, manual configuration of the VCA task scripts is also possible and advised whenever the predefined tasks are not enough:

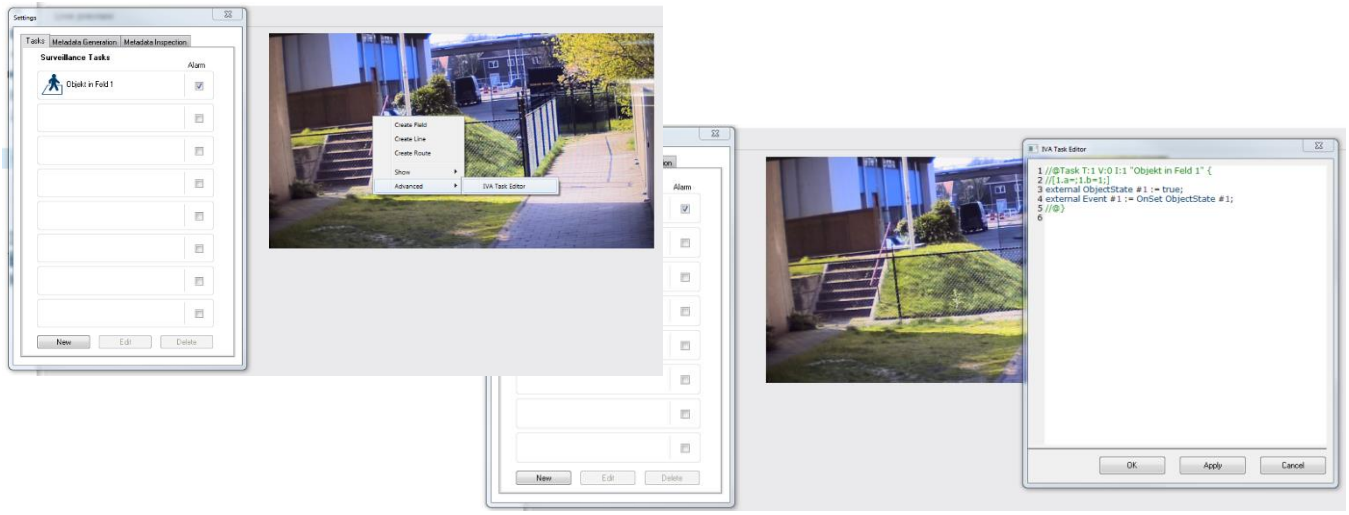
- For backup and exchange of the configured tasks, the script can be copied from and pasted into the task script editor.
- When more than 8 alarm tasks are needed: 16 external alarm tasks are configurable via VCA task scripts.
- Fine tuning the position of lines and fields.
- Line combinations for counting with FW < 6.30.
- Don't touch (museum mode) with FW < 6.10: Alarm needs to be triggered by any part of bounding box, not only by the object center.
- Logical combinations of predefined events are needed.

How to use VCA task scripts?

(1) Define all tasks via task wizards as far as possible.
(2) Change to the VCA task script editor. The already defined tasks can be found there with all details.
(3) Make your modifications.
(4) Change the defined tasks into task type scripted so accidental use of the task wizards will not overwrite your changes.

How to access the VCA task script language?

Go to the Intelligent Video Analytics, Essential Video Analytics or MOTION+ configuration and open the task page. Right-click on the video and select Advanced -> VCA Task Editor. A separate popup with the current VCA task script will appear:



Task & object filter overview

A task consists of

- A task primitive
- An object state or interaction with a task primitive
- A filter on the object properties if task is based on objects

Note that not all tasks and filters are available in every FW version.

MOTION+:

Task Primitives	Tasks
Field	Motion in Field
Image	

Intelligent Video Analytics & Essential Video Analytics (the latter w/o flow tasks):

Task Primitives	Tasks	Flow Tasks	Object Filter
Line	Detect any object	Flow in field	Object area
Field	Crossing line	Counterflow in field	Aspect ratio
Route	Object in field	Crowd detection	Speed
Image	Entering field	Tampering	Direction
	Leaving field		Color
	Following route		Head
	Loitering		Object class
	Removed object		
	Idle object		
	Condition change		
	Similarity search		
	Counter		
	BEV people counter		
	Occupancy		
	Crowd detection		
	Tampering		

Object state and event overview

From the task primitives, the tasks and the object filter as defined in the GUI via the task wizards, target object states and events are derived. These can also be defined directly in the VCA task script language.

Events are always temporal relations

- Up to 8 external events can be shown in GUI
- Up to 16 external events can be defined in total
- Up to 32 events can be defined in total

States can be, amongst others, spatial relations, object properties, tamper states, counter values

- Up to 16 external states can be defined
- Up to 32 states in total can be defined

Events and states can be internal or external. Only external events and states are outputted.

Simple states are be used whenever a property has no corresponding objects. Examples are Motion+, Flow, counter values and tamper states.

Object Properties	Object States	Object events	TamperStates	Other SimpleStates
Direction	InsideField	CrossedLine	SignalTooNoisy	DetectedMotion
Velocity	ObjectsInField	EnteredField	SignalTooDark	DetectedFlow
AspectRatio	IsLoitering	LeftField	SignalTooBright	EstimatedCrowdDensity
ObjecSize	SimilarToColor	FollowedRoute	SignalLoss	Counter
FaceWidth	HasDirection	Appeared	GlobalChange	ObjectsOnScreen
MaxFaceWidth	HasVelocity	Disappeared	RefImageCheckFailed	ObjectsInField
	HasAspectRatio	Idle		ObjectsInState
	HasObjectSize	Removed		
	HasColor			
	HasFace			
	HadFace			
	HasClass			
	HadClass			

Combining & converting object states and events

The combinations of object states and events listed below are possible. Object states can be also converted into events, but only the onset or leaving of an object state is an actual event, not the actual duration of the object in this state.

Logical combination of states / conditions:
- and
- or

Event conditions:
where

Event combination via temporal relations:
- before
- before (<from>,<to>)
- not before

State changes as events:
- OnChange
- OnSet
- OnClear

Understanding the VCA task script language: Line Crossing, GUI

```
//Definition of task primitives
Line #1 := { Point(103, 30) Point(159, 77)};
Line #2 := { Point(26, 65) Point(82, 122)
             DebounceTime(0.50) Direction(1) };

//Definition of alarm task shown in GUI
//@Task T:2 V:0 I:1 "Crossing line 1" {
//[1.a=s1:1;1.b=1;1.c=32;1.d=31;4.a=i:1;]
external Event#1 :={CrossedLine#1};
//@}

//Definition of self-defined task shown in GUI
//@Task T:0 V:0 I:2 "Crossing line 2" {
external Event#2 :={CrossedLine#2};
//@}

//Definition of alarm task not shown in GUI
external Event#3 :={CrossedLine#2};
```

Line: 2 end points

Direction of Line :
// any
Direction(1) //forward
Direction(2) //backward

DebounceTime of Line/Field is optional

CrossedLine #x is an event that triggers when an object crosses Line #x in the specified way

external is keyword for alarms / statistics

Task wizard definition:
//@Task T:x V:y I:z
T:x describes the task number (Object in Field, Line Crossing,... see icon for correct task!)
T:0 describes a self-defined task. Use this for your own scripts to avoid the task wizards overwriting it
V:0 is the version number, currently always 0 I:z is the slot in the tasks page
I:1 describes the occupied slot in the GUI task list. For the slot to change to red in case of alarms, the external event / state defined in the task needs to have the same number as the task
[1.a=s1:1;....] describes the task wizard values

Understanding the VCA task script language: Line crossing with object filters

```
//Definition of task primitives
Line #1 := { Point(103, 30) Point(159, 77)};

//@Task T:2 V:0 I:1 "Crossing line 1" {
//[1.a=s1:1;1.b=1;1.c=32;1.d=31;3.o=(b1:1,
b2:0.1,b3:500, c1:1,c2:0.02,c3:34.00,d1:1,
d2:0,d3:27.78,e1:1,e2:315, e3:45,f1:1,f2:135,
f3:225);4.a=(c:2b19,i:1,pr:2);5.a=(c:1,p:1);
6.a=(d1:8,d2:33,r:2,u:1);]
ColorHistogram #1:= {HSV(60,38,100,25)
      Similarity(75) Outlier(75) };
external Event#1:={CrossedLine#1
      where ObjectSize within(0.1,500)
      and AspectRatio within(0.02,34.00)
      and Velocity within(0,27.78)
      and (Direction within(315,405) or Direction within(135,225))
      and SimilarToColor #1
      and HadFace and MaxFaceWidth within(8,33)};
//@}
```

Object properties can be modeled via object states or via event restrictions

where restricts events with regards to object properties
and / or combine properties
within restricts to value range

The wildcard * describes undefined values

Example for modeling via object state:
ObjectState#1 := Velocity within (30.0,*);

Understanding the VCA task script language: Fields

```
//Definition of task primitives
Field #1 := { Point(56, 24) Point(106, 24)
              Point(106, 74)};
Field #2 := { Point(56, 24) Point(106, 24)
              Point(106, 74) Point(56, 74)
              DebounceTime(0.50)
              ObjectSet(BoundingBox)
              SetRelation(Covering)};

//@Task T:1 V:0 I:1 "Object in field 1" {
//[1.a=1;1.b=1;3.a=i:1;]
external ObjectState #1 := InsideField #1;
//@}
```

Field: 3-16 vertices

DebounceTime of Field is optional

ObjectSet of Field:
ObjectSet(BaryCenter) #default
ObjectSet(BoundingBox)

SetRelation for BountingBox:
SetRelation(Intersection) #default
SetRelation(Covering)

States of Field:
InsideField#x
ObjectsInField#x

Events of Field:
EnteredField#x
LeftField#x

Understanding the VCA task script language: Routes

```
//Definition of task primitives
Route #1 := { Point(34, 111) Distance(5)
              Point(92, 125) Distance(5)
              Point(140, 104) Distance(5)
              Point(143, 72) Distance(5)
              MinPercentage(80) MaxGap(10) };

//@Task T:6 V:0 I:1 "Follow route 1"
{//[1.a=id:1;1.b=1;3.a=i:1;]
external Event #1 := { FollowedRoute #1 };
//@}
```

Route: 2-8 vertices
+distance

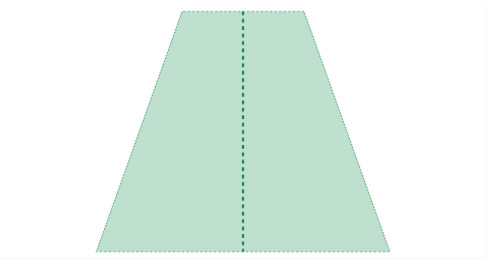
Direction of Line / Route:
any
Direction(1) #forward
Direction(2) #backward

FollowedRoute #x is an event that triggers when an object follows Route #x in the specified way

Example: Alarm if object enters field and afterwards crosses the line

```
//Definition of task primitives
Resolution := { Min(-1, -1) Max(1, 1) };
Field #1 := { Point(-0.6, 0.95) Point(-0.25, -0.95)
              Point(0.25, -0.95) Point(0.6, 0.95)
              DebounceTime(0.50) };
Line #1 := { Point(0.0, -0.95) Point(0.0, 0.95)
              DebounceTime(0.50) };
//@Task T:0 V:0 I:1 "Enter Field and Line" {
external Event#1:={EnteredField #1
                   before (*,30) CrossedLine #1
                   where first.oid == second.oid };
//@}
```

Application: Reduction of false alarms, especially for insects attracted by infrared illumination



before(*,30) means the object needs to cross the line 0-30 seconds after entering the field. As the object needs to enter the field in order to cross the line, the other temporal direction need not be checked.

The same object has to trigger both events. Thus using where first.oid==second.oid

Example: Combining lines for counting

```
//Definition of task primitives
Line #1 := { Point(0, 90) Point(60, 90)
             DebounceTime(0.10) Direction(1) };
Line #2 := { Point(60, 90) Point(90, 60)
             DebounceTime(0.10) Direction(1) };
Line #3 := { Point(90, 60) Point(90, 0)
             DebounceTime(0.10) Direction(1) };

//@Task T:0 V:0 I:1 "Counter 1" {
Event#1 :=CrossedLine#1 or CrossedLine#2
          or CrossedLine#3;
external Counter#1 := { Event#1 Text("Corner Count1:")
                        TopLeft(4,4) Mode(Wraparound)
                        within(0,999999999) };
//@}
```

Make sure that one lines end point is the start of the next line to leave no gaps

Set the same debounce time the same for all lines

Combining line crossing line events via or

Counter defines the counter task. The event on which it counts is configured as well as the placement and label for the count text. The mode selected here says that when reaching max count, the counter then starts from 0 again.

Example: Crossing line1 with 60km/h and line 2 with 20km/h

```
//Definition of task primitives
Line #1 := {Point(50,0) Point(50,144)};
Line #2 := {Point(120,0) Point(120,144)};

//Definition of line crossing events
Event#11 :={CrossedLine#1 where
            Velocity within(13.89,19.44) };
Event#12 :={CrossedLine#2 where
            Velocity within(2.778,5.556) };

//Combination of line crossing events
//@Task T:0 V:0 I:1 "Two Line Speed Check" {
external Event#1 :={Event#11 before Event#12
                    where first.oid==second.oid};
//@}
```

Actually alarming when velocity on first line between 50 km/h and 70 km/h and on second line between 10 km/h and 30 km/h.

The velocity range is not in km/h here, therefore the conditions were generated with GUI default line crossing tasks

Approach:
(1) Defined two line crossings via GUI with velocity filters
(2) Removed all external keywords and task definitions, keeping task primitives and event definitions
(3) Added combination of both line crossing events as user-defined task

The same object has to trigger both line crossings. Thus using where first.oid==second.oid

Example: Stopping in area after crossing line

```
//Definition of task primitives
Line #1 := { Point(130, 0) Point(130, 144) };
Field #2 := { Point(50, 0) Point(115, 0)
              Point(115, 144) Point(50, 144)};

//@Task T:0 V:0 I:1 "Loitering after Line Crossing" {
Event#11 :={CrossedLine #1};
Loitering #13 := { Radius (15) Time (10) };
ObjectState #14 := InsideField #2 and IsLoitering #13;
external Event #1 := {Event #11 before OnSet ObjectState #14
                      where first.oid==second.oid};
//@}
```

Using loitering instead of idle for demonstration purposes

Note that only Event #4 is external and thus generating alarms!

The same object has to trigger both the line crossing and the loitering. Thus using where first.oid==second.oid

Example: Alarm When Object Touches Area

```
//Definition of task primitives
Field #1 := { Point(100, 0) Point(175, 0)
              Point(175, 144) Point(100, 144)
              ObjectSet(BoundingBox)};

//@Task T:0 V:0 I:1 "Object Touches Field" {
ObjectState #1 := InsideField #1;
external Event #1 := OnSet ObjectState #1;
//@}

//Alternative, objects have to come from the outside of the field:
//@Task T:0 V:0 I:2 "Object Touches Field" {
external Event #2 := EnteredField #1;
//@}
```

Alarm on object touch instead of center of gravity of the object via ObjectSet(BoundingBox)

Events and States are enumerated separately, thus both ObjectState and Event may have #1

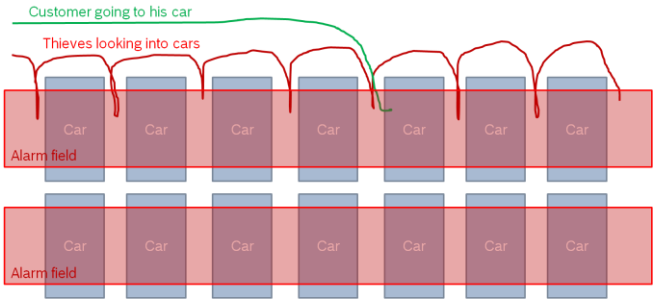
Using state InsideField when objects appearing in the field shall also trigger the alarm

Using event EnteredField when alarming only on objects which have been outside before

Example: Alarm when object enters an area at least 4 times

```
//Definition of task primitives
Resolution := { Min(-1, -1) Max(1, 1) };
Field #1 := { Point(-0.244, -0.922) Point(0.325, -0.944)
              Point(0.350, 0.944) Point(-0.231, 0.944)
              DebounceTime(0.50) };

//@Task T:0 V:0 I:1 "Thieve detection" {
Event #11 :={EnteredField #1 before EnteredField #1
                          where first.oid==second.oid};
Event #12 :={Event #11 before EnteredField #1
                          where first.oid==second.oid};
external Event#1 :={Event #12 before EnteredField #1
                          where first.oid==second.oid};
//@}
```



Application: protecting parked cars

Using before to concatenate events

Using an alternative way to define coordinates by setting a resolution range

Example: Alarm when a second object crosses a line within 3 seconds of the first

```
//Definition of task primitives
Resolution := { Min(-1, -1) Max(1, 1) };
Line #1 := { Point(-0.187, -0.967) Point(-0.156, 0.911)
             DebounceTime(0.50) Direction(2) TriggerPoint(FootPoint) };
//@Task T:0 V:0 I:1 "Tailgaiting" {
Event#21:={CrossedLine#1};
Event#22:={CrossedLine#1};
external Event#1:={Event#21 before(0,3) Event#22
                   where first.oid!=second.oid};
//@}
```

Using before(0,3) for “within 3 seconds”

Two different objects have to trigger the line crossing. Thus using where first.oid!=second.oid

Example: Alarm when at least 2 object are in an area

```
//Definition of task primitives
Resolution := { Min(-1, -1) Max(1, 1) };
Field #1 := { Point(-0.656, -0.700) Point(0.350, -0.767)
              Point(0.469, 0.644) Point(-0.694, 0.644)
              DebounceTime(0.10) };

//@Task T:0 V:0 I:1 "Alarm on more than two objects " {
external ObjectState #1:= ObjectsInField#1 within (2,*);
//@}
```

ObjectsInField#1 returns the number of objects in field 1. Here, the range for the alarm is set from 2 to infinity.

Example: Count the number of objects in an area

```
//Definition of task primitives
Resolution := { Min(-1, -1) Max(1, 1) };
Field #1 := { Point(-0.656, -0.700) Point(0.350, -0.767)
              Point(0.469, 0.644) Point(-0.694, 0.644)
              DebounceTime(0.10) };

//@Task T:0 V:0 I:1 "Count of objects in field " {
external Counter#1 := { ObjectsInField#1 Text("Counter:")
                        TopLeft(-0.975,-0.844) };
//@}
```

ObjectsInField can be used as input for a counter as well

Example: Alarm if one queue is empty and the other has at least 3 persons

```
//Definition of task primitives
Resolution:= { Min(-1, -1) Max(1, 1) };
Field #1:= { Point(-0.150, -0.700) Point(0.250, -0.700)
             Point(0.250, 0.600) Point(-0.150, 0.600)
             DebounceTime(0.10) };
Field #2:= { Point(-0.600, -0.700) Point(-0.200, -0.700)
             Point(-0.200, 0.600) Point(-0.600, 0.600)
             DebounceTime(0.10) };

//@Task T:0 V:0 I:1 "Only one queue" {
ObjectState #1:= ObjectsInField#1 within (3,*);
external Event #1:= {OnSet ObjectState #1
                     where ObjectsInField#2 within (0,0)};
//@}

// To see the results of the no object check, use the following:
//@Task T:0 V:0 I:2 "No object" {
external SimpleState #2:= ObjectsInField#2 within (0,0);
//@}
```

ObjectsInField#2 within (0,0) is a simple state as it refers to an amount of objects, not to an object itself.

Example: Virtual room for counting

```
//Definition of task primitives
Resolution:= { Min(-1, -1) Max(1, 1) };
Line #1 := { Point(-0.7, -0.7) Point(0.7, -0.7) DebounceTime(0.50) Direction(1) };
Line #2 := { Point(-0.7, -0.7) Point(-0.7, 0.7) DebounceTime(0.50) Direction(1) };
Line #3 := { Point(-0.7, 0.7) Point(0.7, 0.7) DebounceTime(0.50) Direction(1) };
//@Task T:0 V:0 I:1 "Virtual Room" {
Event#31:={CrossedLine#1};
Event#30:={CrossedLine#2};
Event#29:={CrossedLine#3};
external Counter#1 := { Event#31 Text("Linie 1:") TopLeft(4,4) };
external Counter#2 := { Event#30 Text("Linie 2:") TopLeft(4,14) };
external Counter#3 := { Event#29 Text("Linie 3:") TopLeft(4,24) };
Counter#32 := { Counter#1 + Counter#2};
external Counter#4 := { Counter#32 - Counter#3 Text("Virtual Room:") TopLeft(4,34) };
//@}
```

Bosch Sicherheitssysteme GmbH
Robert-Bosch-Ring 5
85630 Grasbrunn
Germany
www.boschsecurity.com