

Bosch Video Management System
Client/ Server SDK 1.9 Help
Client Enterprise SDK 1.3 Help, Cameo SDK 1.3 Help



BOSCH

Content

Bosch VMS SDKs: Interfaces to Bosch VMS 4.0 or later.....	3
Introduction	3
What's New?.....	3
SDKs purpose and overview.....	4
Server SDK.....	4
Client SDK.....	4
Client Enterprise SDK	4
Cameo SDK	4
Server SDK, Client SDK and Client Enterprise SDK.....	4
Client Scripts, Client Enterprise and Server Scripts.....	5
Creating a scriptlet	7
Starting an Client scriptlet on accepting a specific alarm	7
Starting a scriptlet on startup of a workstation (Client (Enterprise) AND Server scripts)	8
Starting a server scriptlet in response to an event.....	8
Structure of a script file.....	9
Quick start (Client Enterprise SDK)	10
Quick Reference (Client Enterprise SDK).....	11
Cameo SDK.....	13
Installation	13
Quick start.....	13
Quick Reference.....	14
Examples in C#	16
Example scriptlet 1: Using a Hardware decoder and showing an OSD-Message.....	16
Example scriptlet 2: Open and close a relay.....	17
Example scriptlet 3: Toggle a set of relays	17
Example scriptlet 4: A scriptlet with a switch case block.....	18
Example scriptlet 5: Fill a monitor with cameras 1-9.....	19
Example scriptlet 6: If an input is "on" then reposition a dome camera.....	19
Example scriptlet 7: Remote connection with Exception Handler	20
Example executable 8: Control operator client in response to a Virtual Input Data event and read out the data	21
Question and Answers.....	21
A command summary of the SDKs	23

Bosch VMS SDKs: Interfaces to Bosch VMS 4.0 or later

Introduction

Here you find a basic explanation and simple examples of the interfacing (scripting, calling functions by external applications) functionality in Bosch Video Management System. The interfacing functionality is based on Bosch VMS SDKs. The Bosch VMS SDKs contain all interfaces, object types and methods that are needed for automation and integration to and of Bosch VMS.

The system supports executables in any .NET language and script files in C# and VB.net but only one or the other. When you start the scripting editor the first time you have to decide which programming language you want to use.

The SDKs provide:

- Server API
- Client API
- Client Enterprise API
- Cameo SDK

Of course you can also use the basic commands that are provided by the programming language (for example .Net Framework). When connecting to Bosch VMS by a 3rd party application you can use those APIs to remotely control Bosch VMS. Thereby the SDK provides a powerful integration into other systems.

What's New?

Now IMediaPlayer is disposable. Added support for IDispose.

Added IMediaPlayer.Dispose method sample into CameoSDK sample application.

Added MOV video export. IExporterFactory.CreateMovExporter();

Fixed memory leaks and bugs.

Here is a code sample for Cameo SDK:

```
CameoSdkParameters parameters = new CameoSdkParameters(uniqueId, pathToCameoSdkBinaries,
useInternalLoggingSettings, mainForm);
Sdk = EntryPoint.GetSdk(parameters);
Sdk.ConnectionStateChanged += SdkConnectionStateChanged;
Sdk.ScheduleLogoffRequired += SdkScheduleLogoffRequired;
Sdk.ConfigurationStateChanged += SdkConfigurationStateChanged;

StartupResult result = Sdk.Startup(serverAddress, userName, userPassword,
StartupMode.TryUpdateConfiguration);

if (result == StartupResult.Ok)
{
    // user code
}

Sdk.Shutdown();
```

The ServerScript class which contains the server scripts now supports the Dispose pattern that allows explicitly cleanup of resources. For this, the ServerScript class implements the IDisposable interface (see C# sample below). The Dispose() method of the ServerScript class is called by the Management Server when a new configuration gets activated or when the Management Server stops. This enables stopping threads and timers at a defined point in time.

```
[BvmsScriptClass()]
```

```
public class ServerScript : IDisposable
{
    private readonly IServerApi Api;
    private readonly ILog Logger;

    public ServerScript(IServerApi api)
    {
        this.Logger = LogManager.GetLogger("ServerScript");
        this.Api = api;
    }

    public void Dispose()
    {
        // Use this method to cleanup any objects here (consider fully
        // implementing the Dispose pattern).
        // For example, stop and dispose any started timers. Ensure that
        // all threads that were started are stopped here.
        // DO NOT BLOCK in this method for a very long time, as this may
        // block the activation process of a changed configuration.
    }
}
```

The Dispose pattern is also supported for VB.net server scripts.

SDKs purpose and overview

Server SDK

The Server SDK is meant to control and monitor Bosch VMS by scripts and external applications. You can use those interfaces with a valid administrator account.

Client SDK

The Client SDK is meant to control and monitor Bosch VMS Operator Client by external applications and scripts (these scripts depend on the connected server).

Client Enterprise SDK

The Client Enterprise SDK is meant to control and monitor the behavior of Bosch VMS Enterprise Operator Client of a multisite system by external applications. The SDK allows browsing devices that are accessible by the running, connected Operator Client and to control some UI functionalities.

Cameo SDK

The Cameo SDK is meant to embed Bosch VMS live and playback cameos into your external third-party application. The cameos will follow the Bosch VMS based user permissions.

The Cameo SDK provides a subset of the Bosch VMS Operator Client functionalities that enables you to create applications similar to the Operator Client.

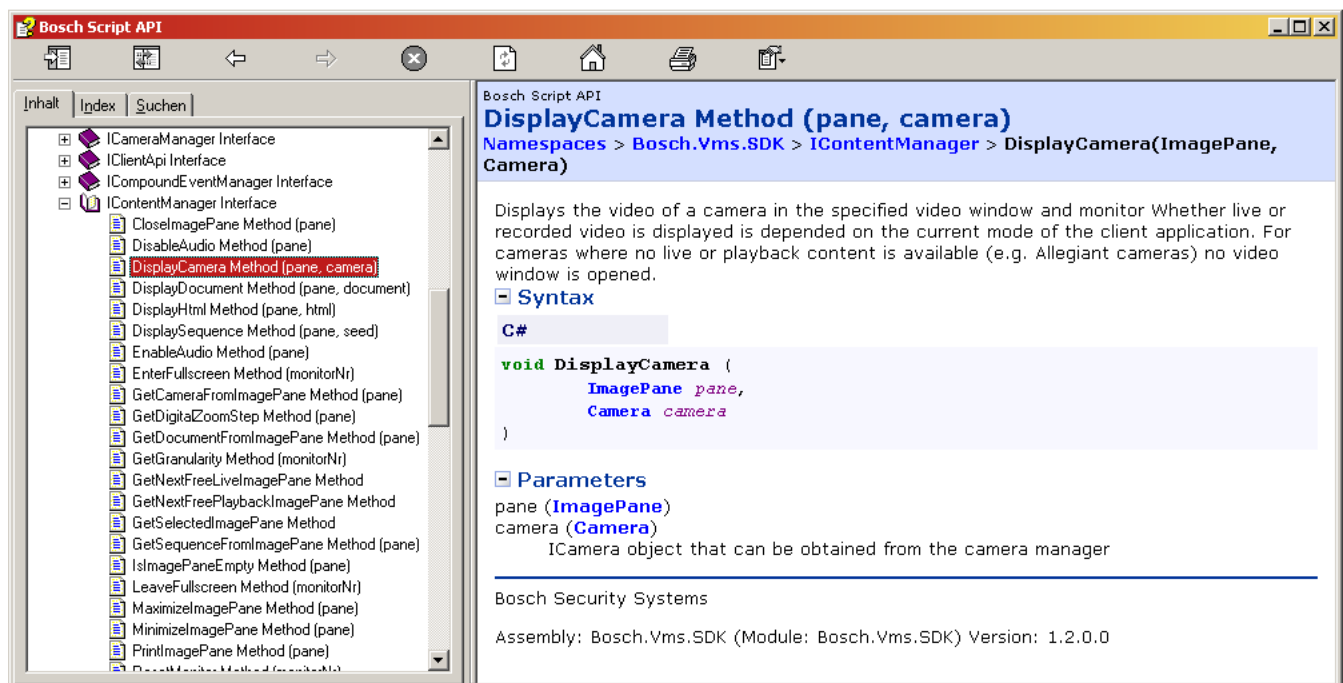
Server SDK, Client SDK and Client Enterprise SDK

The following command set is offered:

Command set	Example	Valid for
1) Basic language commands	If-else constructs, for-/while-loops, arrays, arithmetical operations, creating variables, etc.	Client / Server script / executable

2) Server API commands	All device functions such as switching relays, getting the state of any devices state, setting the camera recording mode, switching a video stream on a hardware decoder, etc. of a single, dedicated VMS server. 16 Managers	Client script Server script Connected applications
3) Client API commands	Several client functions such as setting the granularity of the image window, loading a camera or a map in an image pane, showing a message on the client interface etc. . Offered to control Operator client applications that are connected to a single VMS server. 7 Managers	Client script and client connected applications. Server script only by remoting
4) Client Enterprise API commands	Several client functions such as setting the granularity of the image window, loading a camera or a map in an image pane, showing a message on the client interface etc. . Offered to control Operator client applications that are connected to multiple VMS servers (via Enterprise Management server). 6 Managers	Enterprise client script Enterprise client connected applications

For the complete Server-, Client- and Client Enterprise-command set please refer to the SDK-help manuals.



Example: SDK-help for method “DisplayCamera”

Client Scripts, Client Enterprise and Server Scripts

There are two general types of script files; the server script and the client (Enterprise) script. Each one serves different purposes:

The client (Enterprise) script file contains subroutines (so-called scriptlets) which can be added to the logical tree to be started manually from the client machine by a user. E.g. the user double-clicks a scriptlet on a map and automatically a camera is loaded in full-screen on the second monitor. The client script can also be added to an alarm workflow (i.e. a client script can be triggered when an alarm is accepted).

The server script file contains scriptlets which are triggered automatically on the server by an event. Server scripts can be selected in the Event table in the “Script” column. E.g. in response to an event the server automatically switches a relay.

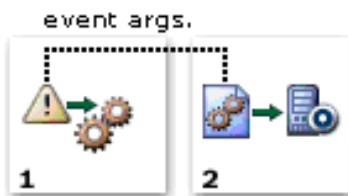
NOTE: In a configured and running system only the scriptlets are called up, never the whole script file. The constructor and static classes will be called up as soon as the scripting engine is initialized.

Client scriptlet function:



- 1) A Client (Enterprise) scriptlet is started from a client workstation by a user action or automatically (startup script). The script can be located in the logical tree or on a map. The scriptlet is started by double click or context menu on that icon. A Client scriptlet can also be started when the operator accepts an alarm.
- 2) The client scriptlet code is processed on the client workstation immediately. It can contain client commands and (Client SDK only) server commands because the Client SDK is always bound to a specific Management server. A Client Enterprise scriptlet may NOT use server commands because the Enterprise client is connected to multiple management servers. Use the Server SDK to control VMS servers explicitly.
- 3) Client commands invoke client actions (e.g. loading a camera in the image window) whereas server commands invoke actions controlled by a dedicated Management Server (e.g. switching relays).

Server scriptlet function:



- 1) A server scriptlet is initiated by an event. This can be any event, e.g. the activation of a schedule, a digital input, a user button, etc.
- 2) The scriptlet code is processed on the Management server which also takes care of all the events. This happens automatically. No user action is required. The scriptlet can also use several arguments that come with each event such as event time, devices name, device state, etc. This way, you can build the system behaviour dependent on the values of the arguments.
- 3) Server commands invoke actions controlled by the Management server (e.g. switching relays).

You can use the same scriptlet for different events. Each event carries data in the Event argument. To get to the individual event data/information (e.g. the device name) you have to look into it.

Creating a scriptlet

Step by step introduction:

- 1) Open command script editor.
- 2) Choose language.
- 3) Right click client script node or server script node.
- 4) Select menu item **New Scriptlet**.
- 5) Name the scriptlet in a reasonable way so that you can find it later.
- 6) Start programming where the automatically created code says “//Insert code here”.

Each scriptlet gets a scriptlet GUID (Globally Unique Identifier) from the system when it is created. It is important to always create a new scriptlet in the script tree on the left side of the command script editor. It is not sufficient to write the code manually to the script file. Also be careful when you use other scriptlets by copying and pasting the code.

NOTE: New scriptlets must be added with the command script editor function “Add scriptlet”.

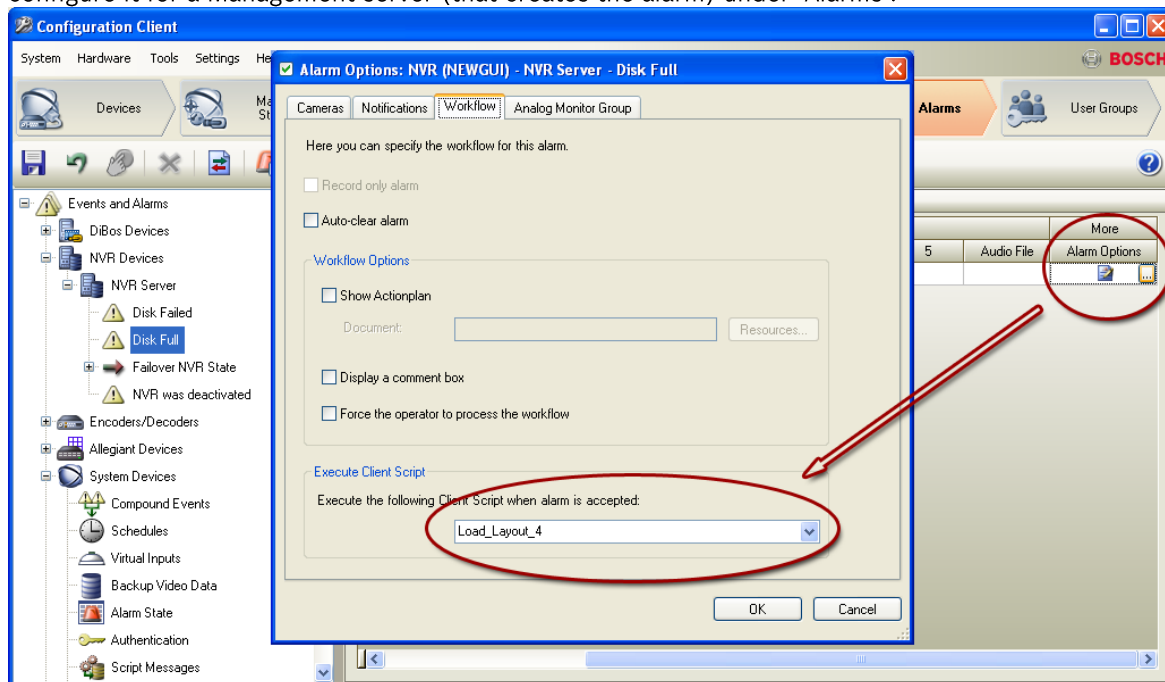
Changing the programming language

The system supports two languages, C# and VB.net. You have to decide once per System in which language you want to program your script files.

NOTE: If you want to change the language afterwards, all scriptlet code will be deleted (method bodies will be replaced by empty bodies) so that you will lose the functionality behind. All parts of the configuration where scripts are used will NOT be lost.

Starting an Client scriptlet on accepting a specific alarm

When one of the operators accepts a certain alarm you can execute a client scriptlet automatically. You can configure it for a Management server (that creates the alarm) under ‘Alarms’:

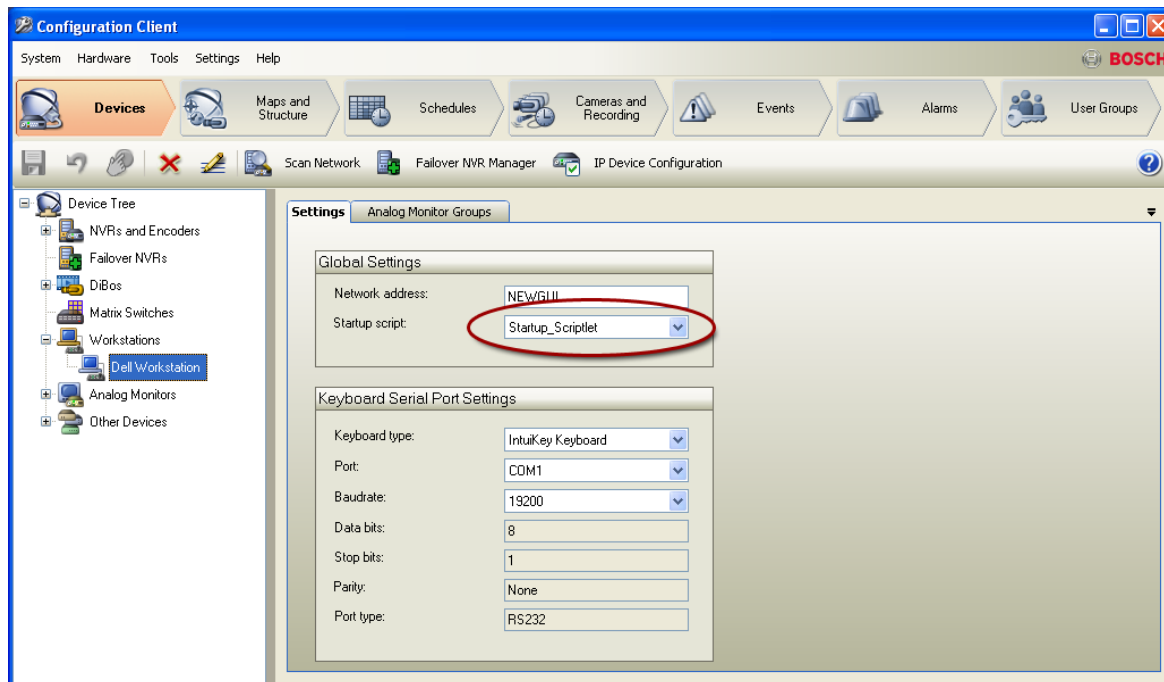


Example:

When the operator accepts an alarm his workstation enters Playback mode, loads a specific camera and sets the time to 1 minute before accepting the alarm.

Starting a scriptlet on startup of a workstation

Right after an operator has logged in on a specific workstation Bosch VMS can start a Client (Enterprise) script automatically. The startup script can be configured on the configuration page of a workstation:

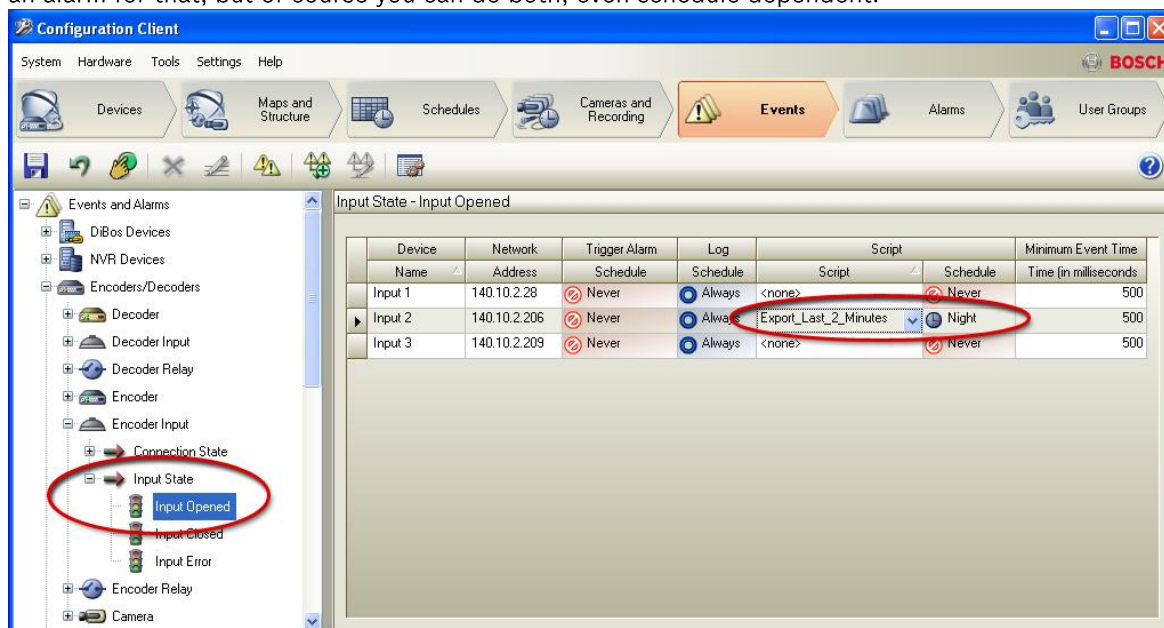


Example:

With the Start-up script you can load a default layout on a specific workstation each time an operator logs in to it.

Starting a server scriptlet in response to an event

In response to each event that happens in Bosch VMS you can execute a server scriptlet. You don't have to trigger an alarm for that, but of course you can do both, even schedule dependent.



Example:

When the door Input opens (Input 2) the server automatically exports the last 2 minutes of the door camera recordings to a Network drive, but only during the night.

Structure of a script file

Client scripts

C# - Client script

```
// ScriptType: ClientScript
// ScriptLanguage: CS

using System;
using System.Diagnostics;
using System.Collections.Generic;
using log4net;
using Bosch.Vms.Core;
using Bosch.Vms.SDK;

[BvmsScriptClass()]
public class ClientScript
{
    private readonly IClientApi Api;
    private readonly ILog Logger;

    public ClientScript(IClientApi api)
    {
        this.Logger =
            LogManager.GetLogger("ClientScript");
        this.Api = api;
    }

    [Scriptlet("computer generated GUID")]
    public void MyScriptlet()
    {
        Camera camera =
            Api.CameraManager.GetCameraByName("Cam01");
        Api.ContentManager.DisplayCamera(new
            ImagePane(1, 1), camera);
    }
}
```

VB.net - Client script

```
' ScriptType: ClientScript
' ScriptLanguage: VB

Imports System
Imports System.Diagnostics
Imports System.Collections.Generic
Imports log4net
Imports Bosch.Vms.Core
Imports Bosch.Vms.SDK

<BvmsScriptClass()> _
Public Class ClientScript

    Private Shared Logger As ILog
    Private Shared Api As IClientApi

    Sub New(api As IClientApi)

        Logger =
            LogManager.GetLogger("ClientScript")
        Me.Api = api
    End Sub

    <Scriptlet("computer generated GUID")> _
    Public Sub MyScriptlet()

        Camera camera =
            Api.CameraManager.GetCameraByName("Cam01")
        Api.ContentManager.DisplayCamera(new
            ImagePane(1, 1), camera)
    End Sub

End Class
```

The “using”/“Imports” declarations bring in different sets of commands that are provided by so called “name spaces”.

Comments can be entered by putting “//” (C#) or “'” (VB.net) at the beginning of a line.

The “public class” is the body of the script file. Between the first “{” and the last “}” of the file you can add different subroutines (C#).

New scriptlets will be added after the last scriptlet in the file

The computer generated GUID (Globally Unique Identifier) has to be in the headline of each scriptlet. Never enter that line manually. A typical GUID looks like "24af66fc-5015-45ee-b27b-a60597f4196d".

Server scripts

C# - Server script

```
// ScriptType: ServerScript
// ScriptLanguage: CS

using System;
using System.Diagnostics;
using System.Collections.Generic;
using log4net;
using Bosch.Vms.Core;
using Bosch.Vms.SDK;

[BvmsScriptClass()]
public class ServerScript : IDisposable
{
    private readonly IServerApi Api;
    private readonly ILog Logger;

    public ServerScript(IServerApi api)
    {
        this.Logger =
            LogManager.GetLogger("ServerScript");
        this.Api = api;
    }

    public void Dispose()
    {
        // Use this method to cleanup any objects here
        (consider fully implementing the Dispose pattern).
        // For example, stop and dispose any started
        timers. Ensure that all threads that were started
        are stopped here.
        // DO NOT BLOCK in this method for a very long
        time, as this may block the activation process of
        a changed configuration.
    }

    [Scriptlet("5e1ledb2-cf45-4f3e-aa2a-15f6a00dd02a")]
    public void ServerScriptlet(EventData e)
    {
        // Insert code here
    }
}
```

VB.net – Server script

```
' ScriptType: ServerScript
' ScriptLanguage: VB

Imports System
Imports System.Diagnostics
Imports System.Collections.Generic
Imports log4net
Imports Bosch.Vms.Core
Imports Bosch.Vms.SDK

<BvmsScriptClass()> _
Public Class IDisposable

    Private Shared Logger As ILog
    Private Shared Api As IServerApi

    Sub New(api As IServerApi)
        Logger =
            LogManager.GetLogger("ServerScript")
        Me.Api = api
    End Sub

    Public Sub Dispose() Implements
        IDisposable.Dispose
        ' Use this method to cleanup any objects here
        (consider fully implementing the Dispose pattern).
        ' For example, stop and dispose any started
        timers. Ensure that all threads that were started
        are stopped here.
        ' DO NOT BLOCK in this method for a very long
        time, as this may block the activation process of
        a changed configuration.
    End Sub

    <Scriptlet("5e1ledb2-cf45-4f3e-aa2a-15f6a00dd02a")> _
    Public Sub ServerScriptlet(e As EventData)
        ' Insert code here
    End Sub
End Class
```

Quick start (Client Enterprise SDK)

The `RemoteClientApiEnterprise` class is the entry point of the SDK. Create an object of the class `RemoteClientApiEnterprise` by passing the client address and the user credentials to the constructor.

RemoteClientApiEnterprise class

Create an object of this class by passing the client address and the user credentials.

The created object provides properties to access different kind of functionalities:

AlarmManager: gives access to alarm management as accept, unaccept, clear alarms...

ApplicationManager: gives access to application status (logged in user, group, live or playback status,...)

CameraManager: gives access to all cameras and helper methods for camera identification

ContentManager: gives access to methods that control the UI of the Operator Client (display a camera, document, change to full-screen mode, control cameo space, ...)

DeviceManager: gives access to all devices and helper methods for device identification **DocumentManager:** gives access to all documents and helper methods for document identification

DomeCameraManager: give access to all dome cameras and helper methods to control the cameras (PanTiltZoom, prepositions,...)

TimelineManager: gives access to methods that control the playback mode of the Operator Client (change playback speed, move to a specific position, control playback direction,...)

The class `RemoteClientApiEnterprise` provides an event `ConnectionLostEvent` that is triggered when the remote Api has lost the connection to the Operator Client.

Quick Reference (Client Enterprise SDK)

IAAlarmManagerEnterprise

To be able to receive Alarms a class derived from `AlarmReceiverEnterprise` must be defined and an object created.

`OnAlarm(IEnumerable<AlarmData> alarmData)` is called by the SDK to submit alarms and their state changes to the 3rd party client application.

`IAAlarmManagerEnterprise` provides methods to manage alarms (accept, clear, unaccept), to get a list of current alarms and to register/unregister to receive alarms with the above described callback method `OnAlarm(...)`.

IAApplicationManagerEnterprise

The application manager interface `IAApplicationManagerEnterprise` provides methods to get the following information:

- logged in user: user name. user group
- Operator Client mode: live or playback
- Server information: list of server Info objects that contain the display name, the network name and the serverId of each server

The application manager interface `IAApplicationManagerEnterprise` provides methods to control the following:

- Change the Operator Client mode to live or playback
- Fire user events
- Post a message that is shown on the Operator Client system
- Speak a submitted text on the Operator Client system

IDeviceManagerEnterprise

The device manager interface provides methods to get the following:

- Device object by providing an Instance Id or a name
- Name of a Device object

ICameraManagerEnterprise

The camera manager interface is inherited from the `IDeviceManagerEnterprise` providing methods to get the following:

- List of Camera objects that are added to the Logical Tree of the Operator Client
- Camera object by providing an `InstanceId`, a name or a local or a logical number

IContentManagerEnterprise

The content manager interface provides methods to control the UI of the Operator Client and to get status information about the content of the Image window.

IDocumentManagerEnterprise

The document manager interface provides methods to browse all documents (html documents and maps) and to get a document object by passing an id or name.

IDomeCameraManagerEnterprise

The dome camera manager interface provides methods to browse all dome cameras, to get a dome camera object by passing an ID or name and to control a dome camera (PTZ, focus, Aux, Iris, ...)

ITimelineManagerEnterprise

The timeline manager interface provides methods to remote control the playback mode of the connected Operator Client. For example the current timeline position can be set, the playback direction and speed can be changed.

Cameo SDK

This chapter provides information on how to get started with the Cameo SDK documentation and the Cameo SDK Test Application including a complete C# project.

NOTICE!

You must have selected the Cameo SDK Manuals checkbox during setup to install the manuals and the test application on your computer.

To display the Cameo SDK reference help:

- Click **Start**, then select **Bosch VMS**, select **Cameo SDK**, and then click **Cameo SDK Help**.

The Cameo SDK reference help is started.

To start the Cameo SDK test application:

- Click **Start**, then select **Bosch VMS**, select **Cameo SDK**, and then click **Cameo SDK Test Application C# Project**.

The test application is started.

To start the Cameo SDK test project:

- Click **Start**, then select **Bosch VMS**, select **Cameo SDK**, and then click **Cameo SDK Test Project**.

A C# project is started in your development environment containing the source code of the Cameo SDK Test Application C# Project.

Installation

This chapter provides information on the installation of CameoSDK and Client Enterprise SDK.

NOTICE!

You can install CameoSDK as a Bosch VMS component using the Bosch VMS Setup.

Quick start

This chapter provides information on how to get started with CameoSDK and with Client Enterprise SDK.

Bosch VMS Cameo SDK is represented by `Bosch.Vms.CameoSdk.dll` file that is available on the target computer after CameoSDK installation (`\CameoSdk\` folder).

To use this assembly just add it to the referenced assemblies list in your C# project.

Here are a few tips that should be kept in mind for successful Cameo SDK assembly usage:

- all DiBos components should be properly registered (by installation of compatible Bosch VMS Operator or Config Client)
- the folder which contains core components (`Bosch.Vms.CameoSdk.Core.dll`) should contain all VideoSDK required files (folders: "Bosch.VideoSDK5.BVIP", "Bosch.VideoSDK5.Core", "Bosch.VideoSDK5.DiBos", "Bosch.VideoSDK5.Divar700", "Bosch.VideoSDK5.Experimental", "Bosch.VideoSDK5.ONVIF", "Bosch.VideoSDK5.RTSP"; files: "Bosch.VideoSDK5.manifest").

To start using the CameoSDK create a `CameoSdkParameters` object and an `EntryPoint` object, then call the `GetSdk (...)` method of the `EntryPoint` class by passing the object `CameoSdkParameters` to get an `ICameoSdk` interface.

EntryPoint class

Create an object of this class to start using the CameoSDK. Call the method `GetSdk` and pass a `CameoSdkParameters` object to get an `ICameoSDK` interface.

CameoSdkParameters class

Create an object of this class to be passed to the `EntryPoint` method `GetSdk(...)`. In the constructor of this class, pass a unique host id that identifies the hosting application, a path to the binary folder of the CameoSDK and the `Form` object of the hosting application.

ICameoSdk

This interface is returned by the `EntryPoint` constructor and provides functionality of the CameoSdk, like Startup/shutdown, connection state, configuration state. Other features of the CameoSdk are exposed via properties of type `IDeviceManager`, `ControlFactory`, `MediaPlayerFactory` and `ExporterFactory` and are described in the following chapter.

Quick Reference

IDeviceManager

This interface provides functionality to discover devices. The `RemoteServerApi` (from Bosch VMS SDK) has to be used to receive events of device states.

IDevice

This interface provides methods to get the name, the id and the network address of the device.

ICamera

This interface represents a camera object. It provides methods to get camera information (name, id, network address, ...) and a PTZ property if the camera is a PTZ camera.

IPtz, IPtzControl, IPtzSettings

These interfaces provide methods to control PTZ cameras.

IControlFactory

Provides methods to create live or playback cameo controls, which can be placed in the 3rd party application UI. In case a playback control has to be created then a `MediaPlayer` object must be passed.

IVideoCameoControl

This interface is returned by the methods provided by the `IControlFactory` to create Image pane controls. This interface provides methods to assign a camera to the control, to get a snapshot of the currently displayed image (live or playback), to check the connection state to the camera and to get a UI control that can be placed in the 3rd party application.

IMediaPlayerFactory

This interface provides functionality to create media player controls (playback control). A single media player can be created and associated with several `VideoCameoControls`, in this case all associated controls will be synchronized (by `CurrentPosition`).

IMediaPlayer

This interface provides functionality to create a media player control (play, stop change playback direction or speed,)

Note:

If the same `MediaPlayer` object is passed to multiple Image pane controls, their playback is synchronized. This case is similar to the playback mode of the Operator Client application. A `MediaPlayer` control can also be created for each Image pane control. In this case the playback is not synchronized and the playback can be controlled differently for each control.

This case is similar to instant playback in the Live Mode of Operator Client.

IExporterFactory

This interface allows creating ASF exporter objects of the interface type `IExporter`.

IExporter

Provides methods for adding cameras to the export, for starting/canceling the export and tracking the progress.

Examples in C#

Example scriptlet 1: Using a Hardware decoder and showing an OSD-Message

When creating a new scriptlet in the Bosch VMS Command Script editor, the identifier and the method frame of the scriptlet are written automatically into the script file. You only have to enter the code between the curly brackets. Read the comments for further explanation of what is programmed in the code.

```
1: [Scriptlet("system generated script ID")]
2: public void ShowCameraOnDecoder()
3: {
4:     Camera c = Api.CameraManager.GetCameraByName("Camera Name");
5:     Decoder d = Api.DecoderManager.GetDecoderByName("Decoder Name");
6:     Api.DecoderManager.DisplayCamera(d, c);
7:     System.Threading.Thread.Sleep(100);
8:     Api.DecoderManager.SetOsd(d, "Hello out there!");
9: }
```

Comments (line-wise):

- 1: The first attribute is the automatically generated identifier, do not change manually!
- 2: Initializing the subroutine with the Name of the scriptlet. You can change that name in the code.
- 4: Initialize a camera object by name, use the system name as you see it the logical tree (hint: you may also identify cameras by their logical number)
- 5: Initialize a decoder object by name, use the system name as you see it in the device tree (hint: you may also identify cameras by their logical number)
- 6: Call up method from decoder manager with camera and decoder as parameters
- 7: Hold the thread for 100 milliseconds. This is inserted because the decoder tends to drop commands when they arrive almost immediately. In this case the OSD Message might get lost when the sleep command is deactivated.
- 8: Display a string on the decoder using the decoder manager again.

NOTE: You should ONLY use decoders that are not part of any Analog Monitor Group (AMG). Otherwise the AMG control will interfere with the script actions.

Example scriptlet 2: Open and close a relay

This scriptlet opens and closes a specific relay.

This scriptlet is meant to be used in a server script.

```
[Scriptlet("system generated script ID")]
public void OpenAndCloseRelays()
{
1: Relay r = Api.RelayManager.GetRelayByName("Relay 1");
2: Api.RelayManager.Open(r);
3: System.Threading.Thread.Sleep(1000);
4: Api.RelayManager.Close(r);
}
```

Comments (line-wise):

- 1: Get the relay as specified in Bosch VMS (by name)
- 2: Open the relay
- 3: Wait for 1000 ms = 1 second
- 4: Close the relay

Example scriptlet 3: Toggle a set of relays

This scriptlet toggles a set of relays based on each relay state. Open relays are closed and vice versa.

Therefore it uses a simple „foreach“ loop.

In each case there is an „if – else“ decision if the relay should be opened or closed, depending on the current state of the relay. This scriptlet is meant to be used in a server script.

```
[Scriptlet("system generated script ID")]
public void ToggleRelays()
{
1: Relay[] relay_array = new Relay[3]
  { Api.RelayManager.GetRelayByName("Relay 1"),
    Api.RelayManager.GetRelayByName("Relay 2"),
    Api.RelayManager.GetRelayByName("Relay 3")
  };

2: foreach (Relay r in relay_array)
  {
    if (Api.RelayManager.GetState(r) == RelayState.On)
    {
      Api.RelayManager.Open(r);
    }
    else
    {
      Api.RelayManager.Close(r);
    }
  }
}
```

Comments (line-wise):

- 1: Open a new array of relays
- 2: Initializing the foreach loop which checks each relay state and inverts it.

Example scriptlet 4: A scriptlet with a switch case block

In this example a server scriptlet is presented which switches different sets of relays in dependency of an argument string. In this special case we use the event arguments ("e") which provide e.g. the device name ("e.DeviceName"). Take motion detection as the event that triggers this script.

If "Camera 1" triggers then the script does that, if "Camera 2" triggers it does something else, and so on...

In the switch case block you define the different behaviors per option. The default behavior can be set in the default block, e.g. in the default case don't do anything.

```
[Scriptlet("system generated script ID")]
public void OnCameraEventSwitchRelays(Event e)
{
    List<Relay> RelayList = new List<Relay>();
    Boolean Toggle = true;
    switch(e.DeviceName) //
    {
        case "Camera 1":
            RelayList.Add(Api.RelayManager.GetRelayByName("TR01"));
            break;
        case "Camera 2":
            RelayList.Add(Api.RelayManager.GetRelayByName("TR02"));
            break;
        case "Camera 3":
            RelayList.Add(Api.RelayManager.GetRelayByName("TR03"));
            RelayList.Add(Api.RelayManager.GetRelayByName("TR04"));
            RelayList.Add(Api.RelayManager.GetRelayByName("TR05"));
            break;
        default:
            Toggle = false;
            Logger.Info("unknown Devicename");
            break;
    }

    if (Toggle) // closing all relays in the relay list
    {
        foreach (Relay r in RelayList)
        {
            Api.RelayManager.Close(r);
        }
    }
}
```

Example scriptlet 5: Fill a monitor with cameras 1-9

This example uses the client script API. It changes the layout on the client and fills the image panes with different cameras.

```
1: Api.ContentManager.ResetMonitor(1); // clears monitor 1 and sets it to a standard 3x3 grid
2: for (int i = 1; i <= 9; i++) // counts i from 1 to 9
3: {
4:     Api.ContentManager.DisplayCamera(new ImagePane(1, i),
    Api.CameraManager.GetCameraByLogicalNumber(i)); // opens camera i in imagepane i
5: }
```

Comments (line-wise):

- 1: Resets the monitor 1. That means that the 1st monitor is cleared and set to a regular 3x3 grid.
- 2: A “for”-loop that counts the variable integer variable “i” from 1 to 9
- 4: for each loop camera i (logical camera number) is opened in image pane i, so that the whole image window is finally filled up with different cameras

Example scriptlet 6: If an input is “on” then reposition a dome camera

This example is just one command, using an if-construct and the command to reposition a dome camera. Here the camera is found by logical number.

```
public void DomeCameraInRespondToInput()
{
    if (Api.InputManager.GetState(Api.InputManager.GetInputByName("Input Test 1")) == InputState.On)
    {
        Camera cam= Api.DomeCameraManager.GetDomeCameraByLogicalNumber(42);
        Api.DomeCameraManager.MoveToPredefinedPosition (cam, 3);
    }
}
```

Example scriptlet 7: Remote connection with Exception Handler

Use “ConnectToServerApi()” to connect remotely to a Bosch VMS client or server.

The connection commands are protected with try catch blocks so that the application does not crash if the connection to the Remote API fails.

```
IServerApi ServerApi;
IClientApi ClientApi;

void ConnectToServerApi()
{
    try
    {
        ServerApi = new RemoteServerApi(serveraddress, username, password);
        // "localhost:5390", "Admin", ""; // Important: right port number
    }
    catch (Exception mye)
    {
        MessageBox.Show(mye.Message, "Connection to server failed!");
    }
}

void ConnectToClientApi()
{
    try
    {
        ClientApi = new RemoteClientApi(serveraddress, username, password);
        // "localhost:5394", "Admin", ""; // Important: right port number
    }
    catch (Exception mye)
    {
        MessageBox.Show(mye.Message, "Connection to client failed");
        ClientConnected = false;
    }
}
```

Example executable 8: Control Operator Client in response to a Virtual Input Data event and read out the data

This example shows you how to react within the Operator client on a certain event (e.g. by opening a dedicated cameo)..

An executable reads out the EventData and switches through the Virtual Input Data field one. Different data causes different reactions. The default block defines the behavior if the value is not matching or unknown.

```
string serverIpAdr = "192.168.2.3:5390"; // server IP address, followed by port number 5390
string clientIpAdr = "192.168.2.17:5394"; // client IP address, followed by port number 5394
string loginName = <login name>; // login name
string loginPassword = <password>; // login password
RemoteServerApi ServerApi = new RemoteServerApi(serverIpAdr, loginName, loginPassword);
RemoteClientApi ClientApi = new RemoteClientApi(clientIpAdr, loginName, loginPassword);
MyReceiver er = new MyReceiver();
ServerApi.EventManager.Register(er);

public class MyReceiver : EventReceiver
{
    public override void OnEvent(EventData e)
    {
        //Do something here.
        switch (e["Data1"]) //reads out the EventData for the 1st Virtual Input Data String (Data1)
        {
            case "camera01": //If it is "camera01" it will execute this case
                // open a certain cameo, e.g. for camera1
                break;
            case "camera02": //If it is "camera02" it will execute this case
                // open a certain cameo, e.g. for camera1
                break;
            default: //If no cases fit it will execute the default case
                Logger.InfoFormat("Content of Data1 unknown - event: {0}", e.ToString());
                break;
        }
    }
}
```

Question and Answers

Q: Do I have to be a .NET developer to create a command script?

A: No, you don't. Using the SDK only requires a basic programming understanding. It uses the .net languages VB.net or C# to be 100% compatible with the application code which is also .NET based.

Q: Why did you choose VB.net or C# as the language for the SDK and for the command scripts?

A: The top 3 reasons for that are:

- 1) .NET languages are the most modern languages you may use. Especially C# was invented to get the advantages of C and Java into a new language, being code reducing, restrictive and straight.
- 2) The complete application runs in a .NET environment. Using the same language makes command scripts and SKD actions 100% compatible to the main software. This way we benefit from a higher stability and performance.
- 3) C# is not C++, it is much easier, plus we hide the mystery of object orientated languages to the user. All you need to do is program a scriptlet (like a macro) which anyway contains more or less the SDK commands only. VB.net might be the preferred language for those of you who have already experience in Visual Basic.

Q: Can I use the same script file on another Bosch VMS?

A: Yes, you can. Easiest you would export the script file in the command script editor and import it on the other Bosch VMS. Note: In this case you would overwrite the existing script file on the other Bosch VMS (if there is one).

Q: How can I use the SDK from a 3rd party application?

A: See example scriptlet 6. You need to instantiate the Client (Enterprise) and/ or the Server API in the 3rd party context. And you need to reference the Bosch.Vms.SDK.dll in your development environment.

To control an operator client depending on server states or alarms you shall access the server SDK from the dedicated operator client.

Q: What SDK commands can I use in the client script?

A: You can use ALL commands that are provided by the Bosch VMS Client SDK and the Bosch VMS Server SDK.

Q: What SDK commands can I use in the client Enterprise script?

A: You can use ALL commands that are provided by the Bosch VMS Client Enterprise SDK.

Q: What SDK commands can I use in the server script?

A: You can use all server commands (look into the interface IServerApi in the Bosch VMS SDK help). Please do not use the client SDK from within a server script!

Q: Why does my camera not load in response to a command script?**AKA: Why does my relay not switch in response to a command script?**

A: The command script editor is very restrictive. It allows saving the script file only if it does not contain any errors. So in 95% of the cases (raw estimation) a typo in the name argument of the “Get...ByName” method is the cause of a malfunction. Please check it very thoroughly: Mind case sensitivity and spaces inside device names.

A command summary of the SDKs

If you browse through the SDK help (e.g. by pressing the SDK help button in the Command script editor) you can explore the whole SDK namespace. All classes, methods, interfaces and other members are described.

For a better understanding here are the major functions listed. All those things you can do with the Bosch VMS SDK:

Client Commands	API interface containing the commands
Get the currently logged in user/user group Get the currently used mode (live or playback) Show a text message on the client (operator has to click OK) Text to speak on the audio speakers of the client Switch to live mode/playback mode	ApplicationManager
Close an image pane Enable/disable audio on the client Display a Camera/Map/Document/Sequence in an image pane Enter/Leave Full-screen mode (no tree on left side) Get the currently displayed camera/document/sequence of an image pane Get/set the currently displayed digital zoom step Get/set the granularity of an image window of a monitor (full-screen – 5x5 granularity) Get the next free live/playback image pane Get the selected image pane Check if an image pane is empty Maximize/minimize an image pane Print an image of an image pane Save a camera image of an image pane to a JPEG file Reset monitor (erases all image panes of a monitor and sets this monitor to 3x3 granularity)	ContentManager
Get/set the position of the timeline Get/set the play speed Play forward/backward Position at first/last available recording Play one step forward/backward Stop playback	TimelineManager
Pause Sequence Play Sequence forward/backward Next step forward/backward Unload the sequence	SequenceManager

Client Enterprise Commands	API interface containing the commands
Get info about connected servers Get the currently logged in user/user group Get the currently used mode (live or playback) Show a text message on the client (operator has to click OK) Text to speak on the audio speakers of the client Switch to live mode/playback mode Fire a user event	ApplicationManagerEnterprise
Close an image pane Enable/disable audio on the client Display a Camera/Map/Document/sequence in an image pane Enter/Leave Full-screen mode (no tree on left side) Get the currently displayed camera/document/Sequence of an image pane Get/set the currently displayed digital zoom step Get/set the granularity of an image window of a monitor (full-screen – 5x5 granularity) Get the next free live/playback image pane Get the selected image pane Check if an image pane is empty Maximize/minimize an image pane Print an image of an image pane Save a camera image of an image pane to a JPEG file Reset monitor (erases all image panes of a monitor and sets this monitor to 3x3 granularity) Switch Image pane bars on/ off	ContentManagerEnterprise
Get/set the position of the timeline Get/set the play speed Play forward/backward Position at first/last available recording Play one step forward/backward Stop playback	TimelineManagerEnterprise
Get/ register/ unregister/ accept/ unaccept/ clear alarms Check if alarm list is empty	AlarmManagerEnterprise
Move PTZ camera to predefined position Send AUX command to PTZ camera AUX commands on/ off Focus/ iris commands Pan/ Tilt/ Zoom Save position	DomeCameraManagerEnterprise

ServerCommands	API interface containing the commands
Start/stop manual alarm recording Get camera state (e.g. too noisy, too dark, too bright, video loss...) Get the current recording mode (normal, motion, alarm) Export an array of cameras (1-n) to a drive as ASF/Native format	CameraManager
Move PTZ camera to predefined position Send AUX command to PTZ camera	DomeCameraManager
Get the connection state of an IP device (network cable unplugged?)	DeviceManager
Get the state of a hardware input (open or closed)	InputManager
Get the state of a hardware relay(open or closed) Open/close a relay	RelayManager
Switch on/off a virtual input Send Data (10 x 120 characters + 1 alarm ID) to a virtual input	VirtualInputManager
Get all recording schedules Get all task schedules Check if a schedule is active	ScheduleManager
Send an Email Send an SMS	SendManager
Write something into the Bosch VMS logbook	LogbookManager
Check if a compound event is true or false	CompoundEventManager
Display matrix camera on an matrix output (perform matrix switch) Get all cameras that are connected to an analog matrix Send a command to an analog matrix Load an analog matrix sequence Control an analog matrix sequence (run, forward, backward, pause) Set an OSD text to a matrix output	MatrixManager
Clear Picture on a decoder Display a camera on a decoder Get the currently displayed camera from a decoder Get the currently set volume of a decoder Set the volume of a decoder Display a OSD text on the decoder picture	AnalogMonitorManager