Advanced Integration Package

Bosch Video IP

BOSCH

en Developer's Guide

Table of Contents

1 General settings

1.1 Device Identification

1.1.1 Device Product Type Identification

With the Device Product Type Identification you can get unique and not changeable device settings such as name, type and software version.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the IDeviceInfo Property from a initialized IDeviceProxy.
	`Bosch.VideoSDK.Device.DeviceInfo deviceInfo = m_deviceProxy.Info;`
2	Retrieve the device Name as shown in the web browser.
	`OEM_Name = deviceInfo.Description;`
3	Retrieve the unique device type.
	`DeviceType = deviceInfo.Type.ToString();`
4	Retrieve the device subtype to identify the variation of the device (transmitter or receiver).
	`DeviceSubType = deviceInfo.SubType;`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Dynamic Capability Detection at Runtime**

– Class List > **IDeviceInfo**

RCP+ over CGI

Steps and codes	
1	Get the current software version with the RCP+ command CONF_SOFTWARE_VERSION.
	`http://<Device IP>/rcp.xml?command=0x002f&type=P_STRING&direction=READ`
2	Get the unique device ID with the RCP+ command CONF_DEVICE_TYPE_IDS.
	`http://<Device IP>/rcp.xml?command=0x0b07&type=P_OCTET&direction=READ`
3	Get the device name as shown in the web browser with the RCP+ command CONF_OEM_DEVICE_NAME.
	`http://<Device IP>/rcp.xml?command=0x097c&type=P_STRING&direction=READ`

RCP+ SDK

Steps and codes	
1	Get the current software version with the RCP+ command CONF_SOFTWARE_VERSION.
	`client.sendRequest(0, 0x002f1030, response);`
2	Get the unique device ID with the RCP+ command CONF_DEVICE_TYPE_IDS.
	`client.sendRequest(0, 0x0b070C30, response);`
3	Get the device name as shown in the web browser with the RCP+ command CONF_OEM_DEVICE_NAME.
	`client.sendRequest(0, 0x097c1030, response);`

1.1.2 Changeable Device Names

Camera name and unit name can be adjusted to their needs.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the IDeviceInfo Property from a initialized IDeviceProxy.
	`Bosch.VideoSDK.Device.DeviceInfo deviceInfo = m_deviceProxy.Info;`
2	Retrieve the camera name. To change the camera name, other integration methods must be used.
	`state.Description = m_deviceInfo.Name;`

Further information

Video SDK Interface Definition.chm:

– Class List > **IDeviceInfo**

– Examples > **VSDKDeviceProxy.cs**

RCP+ over CGI

Steps and codes	
1	Get the unit name with the RCP+ command CONF_UNIT_NAME.
	`http://<Device IP>/rcp.xml?command=0x0024` `&type=P_UNICODE&direction=READ`
2	Set the unit name to "Tower Camera".
	`http://<Device IP>/rcp.xml?command=0x0024` `&type=P_UNICODE&direction=WRITE` `&payload=0x0054006f0077006500720020004300610 06d006500720061 0000`
3	Get the camera name for the first camera with the RCP+ command CONF_CAMNAME.
	`http://<Device IP>/rcp.xml?command=0x0019` `&type=P_UNICODE&num=1&direction=READ`
4	Set the camera name for the first camera to "Tower Cam1".
	`http://<Device IP>/rcp.xml?command=0x0019` `&type=P_UNICODE&num=1&direction=WRITE` `&payload=0x0054006f00770065007200200043006100 6d00310000`

RCP+ SDK

Steps and codes	
1	Get the unit name with the RCP+ command CONF_UNIT_NAME.
	`client.sendRequest(0, 0x00241430, response);`
2	Set the unit name to "Tower Camera".

	`payload.writeWString("Tower Camera");`
	`client.sendRequest(0, 0x00241431, payload, response);`

3	Get the camera name for the first camera.
	`client.sendRequest(0, 0x00191430, response);`
4	Set the camera name for the first camera to "Tower Cam1".

	`payload.writeWString("Tower Cam1");`
	`client.sendRequest(0, 0x00191431, payload, response);`

1.2 Device Capabilities

1.2.1 Video Capabilities

This section describes how to get the video capabilities like the number of inputs and outputs supported coding and stream capabilities: resolution and frame rate settings.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the number of video inputs from a initialized IDeviceProxy.
	`int nbrOfVideoInputs = m_deviceProxy.VideoInputs.Count;`
2	Get the number of video outputs from a initialized IDeviceProxy.
	`int nbrOfVideoOutputs = m_deviceProxy.VideoOutputs.Count;`

RCP+ over CGI

Steps and codes	
1	Get the number of video inputs with the RCP+ command CONF_NBR_OF_VIDEO_IN.
	`http://<Device IP>/rcp.xml?command=0x01d6&type=T_DWORD&direction=READ`
2	Get the number of video outputs with the RCP+ command CONF_ NBR_OF_VIDEO_OUT.
	`http://<Device IP>/rcp.xml?command=0x01d7&type=T_DWORD&direction=READ`
3	Get the video coding type, the supported resolutions and the number of streams. **Note:** This command returns the whole capability list. How to get the video coding type please see **CONF_CAPABILITY_LIST** in the documentation **RCP+ Commands for Advanced Integration Package**.
	`http://<Device IP>/rcp.xml?command=0xff10&type=P_OCTET&direction=READ`
4	Get the supported frame rate modes with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX_OPTIONS. **Note:** This command is only applicable for CPP3 cameras and encoders and CPP4 HD cameras with changeable frame rate 25fps/30fps and 30fps/60fps. For other devices, this parameter has no influence.

	`http://<Device IP>/rcp.xml?command=0x0b9b`
	`&type=P_OCTET&direction=READ&num=1`

Steps and codes	
5	Get all possible H.264 stream combinations, with the RCP+ command CONF_VIDEO_H264_ENC_BASE_OPERATION_MODE_CAPS. **Note:** This command is only applicable for CPP3 cameras and encoders and CPP4 HD cameras and HD resolutions. For SD resolutions like H.264 MPSD or H.264 MP SD ROI you have to define the resolution in the profile itself. For CPP-ENC MPEG4/H.264 cameras this parameter has no influence, because for those devices everything is set in the profile itself (see **CONF_MPEG4_RESOLUTION** in the documentation **RCP+ Commands for Advanced Integration Package**).
	`http://<Device IP>/rcp.xml?command=0x0af9` `&type=P_OCTET&direction=READ&num=1`

RCP+ SDK

Steps and codes	
1	Get the number of video inputs.

	`client.sendRequest(0, 0x01d60830, response);`
	`response.wait();`
	`NbrVideoIn = response.readULong();`

2	Get the number of video outputs.

	`unsigned long NbrVideoOut;`
	`client.sendRequest(0, 0x01d70830, response);`
	`response.wait();`
	`NbrVideoOut = response.readULong();`

3	Get the video coding type, the supported resolutions and the number of streams. **Note:** This command returns the whole capability list. How to get the video coding type please see **CONF_CAPABILITY_LIST** in the documentation **RCP+ Commands for Advanced Integration Package**.

```
client.sendRequest(0, 0xff100c30, response);
response.wait();
response.readUShort(); //BABA
response.skip(2);//Version
unsigned short nbrOfSections = response.readUShort();
for (int i=0; i<nbrOfSections; i++)
   {
   unsigned short sectionType = response.readUShort();
   unsigned short sectionSize = response.readUShort();
   unsigned short sectionElements = response.readUShort();
   switch (sectionType)
      {
      case 0x0001://video
      printf("Video Section\n");
      for (int j=0; j<sectionElements; j++)
         {
         unsigned short type = response.readUShort();
         unsigned short identifier = response.readUShort();
         unsigned short compression = response.readUShort();
         unsigned short inputNbr = response.readUShort();
         unsigned short resolution = response.readUShort();
         }
   break;
```

Steps and codes

4 Get the supported frame rate modes with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX_OPTIONS.

Note: This command is only applicable for CPP3 cameras and encoders and CPP4 HD cameras with changeable frame rate 25fps/30fps and 30fps/60fps. For other devices, this parameter has no influence.

```
client.setNum(1);
client.sendRequest(0, 0x0b9b0C30, response);
response.wait();
nbrOfVideoFormats = response.readOctet();
for (int i=0; i<nbrOfVideoFormats, i++)
   {
   videoFormat[i] = response.readOctet();
   }
```

5 Get all possible H.264 stream combinations, with the RCP+ command CONF_VIDEO_H264_ENC_BASE_OPERATION_MODE_CAPS.

Note: This command is only applicable for CPP3 cameras and encoders and CPP4 HD cameras and HD resolutions. For SD resolutions like H.264 MPSD or H.264 MP SD ROI you have to define the resolution in the profile itself. For CPP-ENC MPEG4/H.264 cameras this parameter has no influence, because for those devices everything is set in the profile itself (see **CONF_MPEG4_RESOLUTION** in the documentation **RCP+ Commands for Advanced Integration Package**).

```
client.setNum(1);
client.sendRequest(0, 0x0af90C30, response);
response.wait();
NbrOfStreams = response.readULong();
while (ris.available())
   {
   streamCombination++;
   for (int i=0; i<NbrOfStreams, i++)
      {
      operationMode[streamCombination][i] = response.readULong();
      }
   }
```

1.2.2 PTZ Capabilities

On Bosch devices, following PTZ modes are possible:

– No PTZ	No PTZ available on this device.
– ePTZ (electronic PTZ)	The camera is not physically moving, a region of interest (ROI) within the video is selected and the user can navigate the viewable area.
– Physical PTZ	The device itself has a motor to physically move and change the viewable area.
– Connected PTZ	The device itself is not a PTZ device, but it is connected via serial connection to a physical PTZ device.

Integration methods:

☐ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

RCP+ over CGI

Steps and codes

1 The PTZ capabilities can be received with the RCP+ command CONF_DEVICE_CAPABILITIES.

With the tags 2, 10 and 14 in the payload, the different PTZ modes can be retrieved:

PTZ mode	Tag 2 (PTZ camera)	Tag 10 (ePTZ)	Tag 14 (BICOM dome)
No PTZ	0	0	0
ePTZ	0	1	0
Physical PTZ	1	0	1
Connected PTZ	1	0	0

```
http://<Device IP>/rcp.xml?command=0x0b60&type=P_OCTET&direction=READ
```

RCP+ SDK

Steps and codes	
1	The PTZ capabilities can be received with the RCP+ command CONF_DEVICE_CAPABILITIES. Please see the RCP+ command reference for payload syntax.

	`RcpInputStream response;`
	`client.sendRequest(0, 0x0b600c30, response);`
	`parseCapabilitiesTags(response);`

1.2.3 Audio Capabilities

This section describes how to get the audio capabilities like the number of inputs and outputs and the used types of audio.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the number of audio inputs from a initialized IDeviceProxy.
	`int nbrOfAudioInputs = m_deviceProxy.AudioInputs.Count;`
2	Get the number of audio outputs from a initialized IDeviceProxy.
	`int nbrOfAudioOutputs = m_deviceProxy.AudioOutputs.Count;`

RCP+ over CGI

Steps and codes	
1	Get the number of audio inputs with the RCP+ command CONF_NBR_OF_AUDIO_IN.
	`http://<Device IP>/rcp.xml?command=0x01d8&type=T_DWORD&direction=READ`
2	Get the number of audio outputs with the RCP+ command CONF_ NBR_OF_AUDIO_OUT.
	`http://<Device IP>/rcp.xml?command=0x01d9&type= T_DWORD&direction=READ`
3	To get the supported audio options (microphone, loudspeaker, etc. support) for audio line 1 use the CONF_AUDIO_OPTIONS command.
	`http://<Device IP>/rcp.xml?command=0x09bf` `&type= T_DWORD&direction=READ&num=1`
4	Get the audio coding type. **Note:** This command returns the whole capability list. How to get the audio coding type please see **CONF_CAPABILITY_LIST** in the documentation **RCP+ Commands for Advanced Integration Package**.
	`http://<Device IP>/rcp.xml?command=0xff10&type=P_OCTET&direction=READ`

RCP+ SDK

Steps and codes

1 Get the number of audio inputs.

```
client.sendRequest(0, 0x01d80830, response);
```

2 Get the number of audio outputs.

```
client.sendRequest(0, 0x01d90830, response);
```

3 To get the supported audio options (microphone, loudspeaker, etc. support) for audio line 1 use the CONF_AUDIO_OPTIONS command.

```
ulong audioOptions;
client.setNum(1);
RcpInputStream response;
client.sendRequest(0, 0x09bf0830, response);
response.wait();
audioOptions = response.readULong();
LineIn = audioOptions&0x1;
LineIn = audioOptions&0x2;
Mic = audioOptions&0x4;
Loudspeaker = audioOptions&0x8;
```

4 Get the audio coding type.

```
client.sendRequest(0, 0xff100c30, response);
response.wait();
response.readUShort(); //BABA
response.skip(2);//Version
unsigned short nbrOfSections = response.readUShort();
for (int i=0; i<nbrOfSections; i++)
    {
    unsigned short sectionType = response.readUShort();
    unsigned short sectionSize = response.readUShort();
    unsigned short sectionElements = response.readUShort();
    switch (sectionType)
    {
    case 0x0002://audio
        printf("Audio Section\n");
        for (int j=0; j<sectionElements; j++)
            {
            unsigned short type = ris.readUShort();
            unsigned short identifier = ris.readUShort();
            unsigned short compression = ris.readUShort();
            }
    break;
```

1.2.4 Recording Status and Capabilities

This section describes how to get supported storage types from a BVIP device, actual information about current recording type, and its status.

Integration methods:

☐ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

RCP+ over CGI

Steps and codes	
1	Get the available storage medium types with the RCP+ command CONF_STORAGE_MEDIUM_AVAIL.
	`http://<Device IP>/rcp.xml?command=0x09d4&type=P_OCTET&direction=READ`
2	Get the current recording mode (locally or centrally managed) with the RCP+ command CONF_REC_MGNT.
	`http://<Device IP>/rcp.xml?command=0x0a89&type=T_OCTET&direction=READ`
3	Optional: In case of centrally managed, get the VRM IP address with the RCP+ command CONF_MANAGING_VRM.
	`http://<Device IP>/rcp.xml?command=0x0aeb` `&type=P_OCTET&direction=READ&num=1`
4	Optional: In case of locally managed, get the managed storage media with the RCP+ command CONF_REC_SPAN_MGR.
	`http://<Device IP>/rcp.xml?command=0x0a36` `&type=P_OCTET&direction=READ&num=1`
5	Get the current recording status with the RCP+ commands CONF_HD_MGR_REC_STATUS/CONF_HD_MGR_REC_STATUS_SECONDARY for the primary/secondary recording.

	`http://<Device IP>/rcp.xml?command=0x0aae` `&type=P_OCTET&direction=READ&num=1`
	
	 `http://<Device IP>/rcp.xml?command=0x0aaf` `&type=P_OCTET&direction=READ&num=1`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_21_RCP+**

RCP+ SDK

Steps and codes	
1	Get the available storage medium types with the RCP+ command CONF_STORAGE_MEDIUM_AVAIL.

```
client.sendRequest(0, 0x0ae90130, response);
response.wait();
while (response.available())
   {
   mediumType[i++] = response.readULong();
   }
```

2	Get the current recording mode (locally or centrally managed) with the RCP+ command CONF_REC_MGNT.

```
client.sendRequest(0, 0x0a890130, response);
response.wait();
recordingMode= response.readOctet();
if (recordingMode>0)
   centrallyManaged = true;
```

3	Optional: In case of centrally managed, get the VRM IP address with the RCP+ command CONF_MANAGING_VRM.

```
client.sendRequest(0, 0x0aeb0C30, response);
response.wait();
vrmip = response.readULong(); //VRM IP
vrmport = response.readUShort();// VRM Port
flags = response.readOctet();//1= SSL
response.readOctet(); //skip the reserved
for (int i=0; i<32; i++)
   username[i] = response.readOctet();
for (int i=0; i<32; i++)
   password[i] = response.readOctet();

backupip = response.readULong();
backupport = response.readUShort();

convertIP(vrmip, *vrmipstr);
convertIP(backupip, *backupipstr);

printf("\n%04x --> VRM IP: %s\n", vrmip, vrmipstr);
printf("%02d --> VRM Port: %d\n", vrmport, vrmport);
printf("%01x --> Flags: %d\n", flags, flags);

printf("Username: %s\n", username);
printf("Password: %s\n", password);

printf("%04x --> Backup VRM IP: %d\n", backupip, backupip);
printf("%02d --> Backup VRM Port: %d\n", backupport, backupport);
```

Steps and codes	
4	Optional: In case of locally managed, get the managed storage media with the RCP+ command CONF_REC_SPAN_MGR.

```
client.sendRequest(0, 0x0a360C30, response);
response.wait();

unsigned long spanMgrIP;
char spanMgrIPstr[15];
unsigned long storageIP;
char storageIPstr[15];
char targetIndex;
char lun;
unsigned short priority;

while (response.available())
   {
   spanMgrIP = response.readULong();
   storageIP = response.readULong();
   targetIndex = response.readOctet();
   lun = response.readOctet();
   response.skip(2);
   priority = response.readUShort();
   response.skip(2);

   convertIP(spanMgrIP, *spanMgrIPstr);
   convertIP(storageIP, *storageIPstr);

   printf("\n%04x --> Span Manager IP: %s\n", spanMgrIP, spanMgrIPstr);
   printf("%04x --> Storage IP: %s\n", storageIP, storageIPstr);
   printf("%01d --> Target Index\n", targetIndex);
   printf("%01d --> Lun\n", lun);
   printf("%01d --> Priority\n", priority);
   }
```

Steps and codes	
5	Get the current recording status with the RCP+ commands CONF_HD_MGR_REC_STATUS/CONF_HD_MGR_REC_STATUS_SECONDARY for the primary/secondary recording.

	`client.setNum(VideoLine);`
	`client.sendRequest(2000, (OpCode)0x0aae0c30, m_RcpInputStream);`
	`response.wait();`
	`byte RecState = response.readOctet();`
	`byte RecProfile = response.readOctet();`
	`byte VideoProfile = response.readOctet();`
	`byte Flags = response.readOctet();`
	`string []RecordingStateList = {"Off", "No Recording", "Stand By", "Pre Alarm Recording", "Alarm Recording", "Post Alarm Recording"};`
	`Console.WriteLine("Recording State: "+RecordingStateList[RecState]+"`
	`   , used Recording Profile: "+RecProfile+"`
	`   , used Video Profile: "+VideoProfile);`
	`   if (Flags>0)`
	`   {`
	`   Console.WriteLine("Alarm Recording Configuration: ");`
	`   if ((byte)(Flags & 0x01) > 0)`
	`      Console.WriteLine("Alarm recording mode");`
	`   if ((byte)(Flags & 0x02) > 0)`
	`      Console.WriteLine("Input Alarm Recording");`
	`   if ((byte)(Flags & 0x04) > 0)`
	`      Console.WriteLine("Motion Alarm Recording");`
	`   if ((byte)(Flags & 0x08) > 0)`
	`      Console.WriteLine("Video Loss Alarm Recording");`
	`   if ((byte)(Flags & 0x10) > 0)`
	`      Console.WriteLine("Virtual Alarm Recording");`
	`   }`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_21_RCP+**

1.2.5 I/O Capabilities

This section describes how to get the capabilities of alarm inputs and outputs (relays).

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the number of alarm inputs from a initialized IDeviceProxy.
	`int nbrOfAlarmInputs = m_deviceProxy.AlarmInputs.Count;`
2	Get the number of alarm outputs (relays) from a initialized IDeviceProxy.
	`int nbrOfAlarmOutputs = m_deviceProxy.Relays.Count;`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Relay Control**

– Class List > **IRelay**

– Class List > **IAlarmInput**

– Examples > **VSDKRelay.cs**

– Examples > **VSDKAlarmInput.cs**

Related sections in this document:

– **General Event Handling**, see page 89

RCP+ over CGI

Steps and codes	
1	Get the number of alarm inputs with the RCP+ command CONF_NBR_OF_ALARM_IN.
	`http://<Device IP>/rcp.xml?command=0x01db&type=T_DWORD&direction=READ`
2	Get the number of alarm outputs (relays) with the RCP+ command CONF_NBR_OF_ALARM_OUT.
	`http://<Device IP>/rcp.xml?command=0x01dc&type= T_DWORD&direction=READ`
3	Get the number of virtual alarms with the RCP+ command CONF_NBR_OF_VIRTUAL_ALARMS.
	`http://<Device IP>/rcp.xml?command=0x0aed&type= T_DWORD&direction=READ`
4	**Note:** To get the number of alarm inputs, alarm outputs and virtual alarms, you can also use the RCP+ command CONF_CAPABILITY_LIST.

RCP+ SDK

Steps and codes	
1	Get the number of alarm inputs.

	`client.sendRequest(0, 0x01db0830, response);`
	`response.wait();`
	`NbrAlarmIn = response.readULong();`

2	Get the number of alarm outputs (relays).

	`client.sendRequest(0, 0x01dc0830, response);`
	`response.wait();`
	`NbrAlarmOut = response.readULong();`

3	Get the number of virtual alarms.

	`client.sendRequest(0, 0x0aed0830, response);`
	`response.wait();`
	`NbrVirtualAlarm = response.readULong();`

4	**Note:** To get the number of alarm inputs, alarm outputs and virtual alarms, you can also use the RCP+ command CONF_CAPABILITY_LIST.

1.3 Network Settings

This section describes how to read and set the basic network settings. These include the MAC address, IP, subnet, gateway and DHCP settings.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

1.3.1 Reading the Network Parameters

VideoSDK

Steps and codes	
1	Get the IDeviceInfo property from a initialized IDeviceProxy.
	`Bosch.VideoSDK.Device.DeviceInfo deviceInfo = m_deviceProxy.Info;`
2	Retrieve the IP and MAC address from IDeviceInfo properties.

	`IPAddress = m_deviceInfo.IPAddress;`
	`MacAddress = m_deviceInfo.MacAddress;`

Further information

Video SDK Interface Definition.chm:

– Class List > **IDeviceInfo**

– Examples > **VSDKDeviceProxy.cs**

RCP+ over CGI

Steps and codes	
1	Get device DHCP status.
	`http://<Device IP>/rcp.xml?command=0x00af&type=T_OCTET&direction=READ`
2	Get device IP.
	`http://<Device IP>/rcp.xml?command=0x007c&type=P_STRING&direction=READ`
3	Get device IP mask.
	`http://<Device IP>/rcp.xml?command=0x007d&type=P_STRING&direction=READ`
4	Get device IP gateway.
	`http://<Device IP>/rcp.xml?command=0x007f&type=P_STRING&direction=READ`
5	Get device MAC address.
	`http://<Device IP>/rcp.xml?command=0x00bc&type=P_OCTET&direction=READ`

RCP+ SDK

Steps and codes	
1	Get device DHCP status.
	`client.sendRequest(0, 0x00af0130, response);`
2	Get device IP.
	`client.sendRequest(0, 0x007c1030, response);`
3	Get device IP mask.
	`client.sendRequest(0, 0x007d1030, response);`
4	Get device IP gateway.
	`client.sendRequest(0, 0x007f1030, response);`
5	Get device MAC address.
	`client.sendRequest(0, 0x00bc0C30, response);`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_06_RCP+**

1.3.2 Setting the Network Parameters

RCP+ over CGI

Steps and codes	
1	Set DHCP to off.
	`http://<Device IP>/rcp.xml?command=0x00af &type=T_OCTET&direction=WRITE&payload=0`
2	Set device IP.
	`http://<Device IP>/rcp.xml?command=0x007c &type=P_STRING&direction=WRITE&payload=192.168.1.100`
3	Set device IP mask.
	`http://<Device IP>/rcp.xml?command=0x007d &type=P_STRING&direction=WRITE&payload=255.255.255.0 (as string)` `http://<Device IP>/rcp.xml?command=0x0002 &type=T_DWORD&direction=WRITE&payload=0xffff0000`
4	Set device IP gateway.
	`http://<Device IP>/rcp.xml?command=0x007f &type=P_STRING&direction=WRITE&payload=160.10.5.0`

RCP+ SDK

Steps and codes	
1	Set DHCP on or off.
	`payload.writeOctet(useDHCP ? 1 : 0);` `client.sendRequest(0, 0x00af0131, payload, response);`
2	Set device IP.
	`payload.writeString(newIP);` `client.sendRequest(0, 0x007c1031, payload, response);`
3	Set device IP mask.
	`payload.writeString(newMask);` `client.sendRequest(0, 0x007d1031, payload, response);`
4	Set device IP gateway.
	`payload.writeString(newGateway);` `client.sendRequest(0, 0x007f1031, payload, response);`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_06_RCP+**

1.4 Network Scan

Network scan offers you an easy way to detect Bosch video over IP devices on your network automatically. Devices generate and transmit their replies to the request within two seconds.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Initialize the device finder.
	`m_deviceFinder = new Bosch.VideoSDK.Device.DeviceFinderClass();`
2	Add event handlers for the device finder's events.

	`m_deviceFinder.DeviceDetected += new Bosch.VideoSDK.GCALib.`
	`_IDeviceFinderEvents_DeviceDetectedEventHandler(OnDeviceDetected);`
	`m_deviceFinder.DeviceRemoved += new Bosch.VideoSDK.GCALib.`
	`_IDeviceFinderEvents_DeviceRemovedEventHandler(OnDeviceRemoved);`

3	Select the discovery mode.
	`m_deviceFinder.DiscoveryMode = m_eDeviceDiscoveryModeEnum;`
4	Start the device scan.
	`m_deviceFinder.StartDetect(0);`
5	Collect the results in via event handlers.

	`private void OnDeviceDetected(Bosch.VideoSDK.Device.DeviceInfo`
	`deviceInfo)`
	`{/*do something with the results*/}`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Device Detection**
– Class List > **IDeviceFinder**
– Examples > **VSDKDeviceFinder.cs**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_03_VSDK**

RCP+ over CGI

Steps and codes	
1	Each Bosch Video over IP device with Firmware 5.60 or newer is able to send/receive an auto-detect request/replay to detect other Bosch Video over IP devices on the network.
	`http://<Device IP>/autodetect.xml`
2	The response carries at least the information about MAC Address, IP Address, Device Name, Unit Name and Serial Number. Please see below for an example response.

```
<AutodetectResult>
   <ScanGroup>255.255.255.255</ScanGroup>
   <ScanPort>1800</ScanPort>
   <NumDevices>349</NumDevices>
   <Result>
      <Mac>00-07-5f-71-d8-1e</Mac>
      <IP>160.10.0.150</IP>
      <Link>http://160.10.0.150</Link>
      <IPv6/>
      <LinkIPv6/>
      <Name>VIP-X1600-XFMD</Name>
      <unitName>160.10.0.150 - Unit</unitName>
      <HttpPort>80</HttpPort>
      <HttpsPort>443</HttpsPort>
      <serialNumber>f1001341:30500410</serialNumber>
```

RCP+ SDK

Steps and codes	
1	Trigger the scan for all Bosch devices with the keep details method.
	`NetworkScanner::triggerScan(2, 0xff, 0, CB_ScanResult, 0);`
2	Collect the scan results in the callback function.

```
void CB_ScanResult(void *ptr, RcpHeader &hdr, RcpInputStream &ris)
{
const unsigned long *devs = rcpplus::NetworkScanner::getDeviceList();
   for (int i=0; devs[i]; i++)
   {
   printf("%d.%d.%d.%d\n",
      ((unsigned char*)&devs[i])[0],
      ((unsigned char*)&devs[i])[1],
      ((unsigned char*)&devs[i])[2],
      ((unsigned char*)&devs[i])[3]);
   }
}
```

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_01_RCP+**

2 Video

2.1 Video Settings

2.1.1 Change Video Profile

There are two methods to change the video profile on a device, depending on the product group where the device belongs to. All device product groups can be found at

http://ipp.boschsecurty.com > Tools > Bosch firmware-product matrix

To change the streaming behavior of a BVIP device, several aspects need to be taken into account. This simplified explanation covers the most common use cases.

First, the video input format determines the frame rate. It is used to switch between 60 fps and 30 fps format. Next, you select the basic operating mode, which sets the resolution. All other settings like bit rate, reduced fps or expert settings (GOP structure, I-frame distance, or quantizer) are handled by profiles. Once a profile is set to the customer‘s needs, it can then be assigned to a video stream.

Integration methods:

☐ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

RCP+ over CGI

Change H.264 profile for CPP 3 and CPP 4 devices

Steps and codes	
1	Set the base frame rate with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX. Supported input formats can be retrieved with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX_OPTIONS. In the code below, the base frame-rate is set to 60 fps (0x0a = 720P**60**) for the first video input (num=1).

	`http://<Device IP>/rcp.xml?command=0x0b10`
	`&type=P_OCTET&direction=WRITE&num=1`
	`&payload=0x000a000000000000000000000000000000000`

2	Choose a stream 1 and stream 2 mode with the RCP+ command CONF_VIDEO_H264_ENC_BASE_OPERATION_MODE. In the code below, the base operation mode of the H.264 encoder for the first video input is changed. Stream 1 is set to H.264 MP fixed 1080p and stream 2 to 0: copy stream (0x6).

	`http://<Device IP>/rcp.xml?command=0x0ad3`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x0000000600000000`

Steps and codes	
3	Settings like bit rate, reduced fps or expert settings (GOP-structure, I-frame distance or quantizer) are handled by the profiles. Get the current profile for the first stream on video input 1 with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER. The other payload parameters can be ignored on read request.

	`http://<Device IP>/rcp.xml?command=0x061c`
	`&type=P_OCTET&direction=READ&payload=0x0101000000000000`

4	Change the target and maximum data rate in the profile with the RCP+ commands CONF_MPEG4_BANDWIDTH_KBPS and CONF_MPEG4_BANDWIDTH_KBPS_SOFT_LIMIT. In the code below, the target bit rate of the selected profile is set. The profile is 1 and the target bit rate is 5000 kbps.

	`http://<Device IP>/rcp.xml?command=0x0607`
	`&type=T_DWORD&direction=WRITE&num=1&payload=5000`

	In the code below, the maximum bit rate of the selected profile is set. The profile is 1, the bit rate is 10000 kbps.

	`http://<Device IP>/rcp.xml?command=0x0612`
	`&type=T_DWORD&direction=WRITE&num=1&payload=10000`

5	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In example below, the skip ratio is set to 3, which means the frame rate is 10 fps on a device where the max frame rate is 30 fps.

	`http://<Device IP>/rcp.xml?command=0x0606`
	`&type=T_DWORD&direction=WRITE&num=1&payload=3`

6	Assign the profile to the encoder with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER. In the code below, the preset 1 is set to the encoder 1.

	`http://<Device IP>/rcp.xml?command=0x061c`
	`&type=P_OCTET&direction=WRITE&payload=0x0101000001000000`

Change H.264 profile for CPP ENC devices

Steps and codes	
1	Get the current stream configuration for the first stream on video input 1 with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER. The other payload parameters can be ignored on read request.
	`http://<Device IP>/rcp.xml?command=0x061c &type=P_OCTET&direction=READ&payload=0x0101000000000000`
2	Change the resolution with the RCP+ command CONF_MPEG4_RESOLUTION.
	`http://<Device IP>/rcp.xml?command=0x0608 &type=T_DWORD&direction=WRITE&num=1&payload=7`
3	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In the example below, the skip ratio is set to 3, which means the frame rate is 10 fps on a device where the max frame rate is 30 fps.
	`http://<Device IP>/rcp.xml?command=0x0606 &type=T_DWORD&direction=WRITE&num=1&payload=3`
4	Change the target and maximum data rate in the profile with the RCP+ commands CONF_MPEG4_BANDWIDTH_KBPS and CONF_MPEG4_BANDWIDTH_KBPS_SOFT_LIMIT. In the code below, the target bit rate of the selected profile is set. The profile is 1 and the target bit rate is 5000 kbps.
	`http://<Device IP>rcp.xml?command=0x0607 &type=T_DWORD&direction=WRITE&num=1&payload=5000`
	In the code below, the maximum bit rate of the selected profile is set. The profile is 1, the bit rate is 10000 kbps.
	`http://<Device IP>/rcp.xml?command=0x0612 &type=T_DWORD&direction=WRITE&num=1&payload=10000`
5	Assign the profile to the encoder with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER. In the code below, the preset 1 is set to the encoder 1 on line 1.
	`http://<Device IP>/rcp.xml?command=0x061c &type=P_OCTET&direction=WRITE&payload=0x0101000001000000`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_05_CGI**

Change JPEG profile for CPP 3 and CPP 4 devices

Steps and codes	
1	On CPP3 and CPP4 devices, the JPEG stream settings such as resolution, frame rate and quality are adjusted with the RCP+ command CONF_JPEG_STREAM_SETUP. In the code below, the resolution is set to 720p (0x12) with full 30 fps (0x7530) and medium quality settings (0x32).

	`http://<Device IP>/rcp.xml?command=0x0ad5`
	`&type=T_OCTET&direction=WRITE&payload=0x000000120000753000000032`

Change JPEG profile for CPP ENC devices

Steps and codes	
1	On CPP-ENC devices, the JPEG settings are adjusted with video profiles. Get the current profile for the third stream, which is the JPEG stream, on video input 1. The other payload parameters can be ignored on read request.

	`http://<Device IP>/rcp.xml?command=0x061c`
	`&type=P_OCTET&direction=READ&payload=0x0103000000000000`

2	Change the resolution with the RCP+ command CONF_MPEG4_RESOLUTION. In the example below, VGA resolution (7) is assigned for profile 5.

	`http://<Device IP>/rcp.xml?command=0x0608`
	`&type=T_DWORD&direction=WRITE&num=5&payload=7`

3	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In the example below, the frame rate is set to 10 fps (the device base frame rate is 30 fps) with frame skip value 3 on the 5th profile.

	`http://<Device IP>/rcp.xml?command=0x0606`
	`&type=T_DWORD&direction=WRITE&num=5&payload=3`

4	Change the JPEG bit rate with the RCP+ command CONF_JPEG_BANDWIDTH_KBPS for the 5th video profile to the value 1500.

	`http://<Device IP>/rcp.xml?command=0x061d`
	`&type=T_DWORD&direction=WRITE&num=5&payload=1500`

5	Assign the 5th profile to the JPEG encoder.

	`http://<Device IP>/rcp.xml?command=0x061c1`
	`&type=P_OCTET&direction=WRITE&payload=0x0103000005000000`

RCP+ SDK

Change H.264 profile for CPP 3 and CPP 4 devices

Steps and codes

1 Set the base frame rate with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX according to the RCP+ documentation.

Supported Input formats can be retrieved with the RCP+ command CONF_VIDEO_INPUT_FORMAT_EX_OPTIONS.

In the code below, the base frame-rate is set to 60fps (0x0a = 720P**60**) for the first video input (num=1).

```
payload.writeUShort(000a);        // 720P60
payload.write(zerobuffer, 18);    // 18 bytes reserved
client.setNum(1);
RcpInputStream response;
client.sendRequest(0, 0x0b100C31, payload, response);
```

2 Choose a stream 1 and stream 2 mode with the RCP+ command CONF_VIDEO_H264_ENC_BASE_OPERATION_MODE.

In the code below, the base operation mode of the H.264 encoder for the first video input is changed.

```
payload.writeULong(Stream1Mode);
payload.writeULong(Stream2Mode);
client.setNum(1);
RcpInputStream response;
client.sendRequest(0, 0x0ad30C31, payload, response);
```

3 Settings like bit rate, reduced fps or expert settings (GOP-structure, I-frame distance or quantizer) are handled by the profiles.

Get the current profile for the first stream on video input 1 with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER.

The other payload parameters can be ignored on read request.

```
payload.writeOctet(LineNbr);
payload.writeOctet(encoderNbr);
payload.write(zeroBuffer, 6);
RcpInputStream response;
client.sendRequest(0, 0x061c0C30, payload, response);
```

4 Change the target and maximum data rate in the profile with the RCP+ commands CONF_MPEG4_BANDWIDTH_KBPS and CONF_MPEG4_BANDWIDTH_KBPS_SOFT_LIMIT.

```
client.setNum(profileNbr);
payload.writeUShort(bandwidthTarget);
RcpInputStream response;
client.sendRequest(0, 0x06070831, payload, response);
payload.writeULong(bandwidthMax);
RcpInputStream response;
client.sendRequest(0, 0x06120831, payload, response);
```

Steps and codes	
5	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In example below, the skip ratio is set to 3, which means the frame rate is 10 fps on a device where the max frame rate is 30 fps.

	`m_RcpOutputStream.writeULong(skipRatio);`
	`client.setNum(profileNbr);`
	`client.sendRequest(2000, (OpCode)0x06060831, payload, response);`

6	Assign the profile to the encoder with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER.

	`payload.writeOctet(LineNbr);`
	`payload.writeOctet(encoderNbr);`
	`payload.writeUShort(0);`
	`payload.writeOctet(profileNbr);`
	`payload.write(zeroBuffer, 3);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061c0c31, payload, response);`

Change H.264 profile for CPP ENC devices

Steps and codes	
1	Get the current stream configuration with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER. Only line and coder parameters needs to be specified on a read request.

	`payload.writeOctet(LineNbr);`
	`payload.writeOctet(encoderNbr);`
	`payload.write(zeroBuffer, 6);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061c0C30, payload, response);`

2	Change the resolution with the RCP+ command CONF_MPEG4_RESOLUTION.

	`m_RcpOutputStream.writeULong(resolution);`
	`client.setNum(profileNbr);`
	`client.sendRequest(2000, (OpCode)0x06080831, payload, response);`

3	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In example below, the skip ratio is set to 3, which means the frame rate is 10 fps on a device where the max frame rate is 30 fps.

	`m_RcpOutputStream.writeULong(skipRatio);`
	`client.setNum(profileNbr);`
	`client.sendRequest(2000, (OpCode)0x06060831, payload, response);`

Steps and codes	
4	Change the target and maximum data rate in the profile with the RCP+ commands CONF_MPEG4_BANDWIDTH_KBPS and CONF_MPEG4_BANDWIDTH_KBPS_SOFT_LIMIT.

	`client.setNum(profileNbr);`
	`payload.writeUShort(bandwidthTarget);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x06070831, payload, response);`
	`payload.writeULong(bandwidthMax);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x06120831, payload, response);`

5	Assign the profile to the encoder with the RCP+ command CONF_MPEG4_CURRENT_PARAMS_REL_CODER.

	`payload.writeOctet(LineNbr);`
	`payload.writeOctet(encoderNbr);`
	`payload.writeUShort(0);`
	`payload.writeOctet(profileNbr);`
	`payload.write(zeroBuffer, 3);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061c0c31, payload, response);`

Change JPEG profile for CPP 3 and CPP 4 devices

Steps and codes	
1	On CPP3 and CPP4 devices, the JPEG stream settings such as resolution, frame rate and quality are adjusted with the RCP+ command CONF_JPEG_STREAM_SETUP.

	`payload.writeULong(Resolution);`
	`payload.writeULong(FPS * 1000);`
	`payload.writeULong(Quality);`
	`client.sendRequest(2000, (OpCode)0x0ad50c31, payload, response);`

Change JPEG profile for CPP ENC devices

Steps and codes	
1	On CPP ENC devices, the JPEG settings are adjusted with video profiles. Get the current profile for the third stream, which is the JPEG stream. The other payload parameters can be ignored on read request.

	`payload.writeOctet(LineNbr);`
	`payload.writeOctet(3);               // jpeg encoder`
	`payload.write(zeroBuffer, 6);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061c0C30, payload, response);`

2	Change the resolution with the RCP+ command CONF_MPEG4_RESOLUTION.

	`m_RcpOutputStream.writeULong(resolution);`
	`client.setNum(profileNbr);`
	`client.sendRequest(2000, (OpCode)0x06080831, payload, response);`

3	Change the frame rate with the RCP+ command CONF_MPEG4_FRAME_SKIP_RATIO. In the example below, the frame rate is set to 10 fps (the device base frame rate is 30 fps) with frame skip value 3 on the 5th profile.

	`m_RcpOutputStream.writeULong(skipRatio);`
	`client.setNum(profileNbr);`
	`client.sendRequest(2000, (OpCode)0x06060831, payload, response);`

4	Change the JPEG bit rate with the RCP+ command CONF_JPEG_BANDWIDTH_KBPS.

	`client.setNum(profileNbr);`
	`payload.writeUShort(bandwidth);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061d0831, payload, response);`

5	Assign the profile to the JPEG encoder.

	`payload.writeOctet(LineNbr);`
	`payload.writeOctet(3);               // jpeg encoder`
	`payload.writeUShort(0);`
	`payload.writeOctet(profileNbr);`
	`payload.write(zeroBuffer, 3);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x061c0c31, payload, response);`

2.1.2 Camera and Alarm Stamping

This section describes how to enable/disable the camera and alarm stamping video overlays in the device.

Integration methods:

- ☐ VideoSDK
- ☑ RCP+ over CGI
- ☐ RTSP
- ☑ RCP+ SDK

RCP+ over CGI

Steps and codes	
1	The camera name stamping can be changed with the RCP+ command CONF_NAME_STAMP_VAL. Following options are available for the camera name stamping: 0 = camera name stamping is disabled 1 = camera name stamping is on the bottom 2 = camera name stamping is on the top 3 = custom camera name stamping position In the code below, the camera stamping is set to custom stamping position.

	`http://<Device IP>/rcp.xml?command=0x0084`
	`&type=T_OCTET&direction=WRITE&num=1&payload=3`

2	Optional: Only necessary if option 3 (custom camera name stamping position) is chosen. The position can be set with the RCP+ command CONF_STAMP_ATTR_NAME. The first byte sets the x position (0-255) and the second byte sets the y position (0-255). The following 10 bytes are reserved and not used right now. In the code below, the camera stamping is set to left (x=5) bottom (y=245) corner.

	`http://<Device IP>/rcp.xml?command=0x0936`
	`&type=T_OCTET&direction=WRITE&num=1&payload=0x05f500000000000000000000`

3	The alarm text stamping can be changed with the RCP+ command CONF_ALARM_DISP_VAL. Following options are available for the camera name stamping: 1 = alarm stamping is disabled 2 = alarm stamping is on 3 = alarm name stamping position In the code below, the alarm stamping is set to custom stamping position.

	`http://<Device IP>/rcp.xml?command=0x008e`
	`&type=T_OCTET&direction=WRITE&num=1&payload=3`

Steps and codes	
4	Optional: Only necessary if option 2 (custom alarm stamping position) is chosen. The position can be set with the RCP+ command CONF_STAMP_ATTR_ALARM. The first byte sets the x position (0-255) and the second byte sets the y position (0-255). The following 10 bytes are reserved and not used right now. In the code below, the alarm stamping is set to center (x=120) top (y=10) position.

	`http://<Device IP>/rcp.xml?command=0x0938`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x780a00000000000000000000`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_12_RCPP_CS_AlarmStamping**

RCP+ SDK

Steps and codes	
1	The camera name stamping can be changed with the RCP+ command CONF_NAME_STAMP_VAL. Following options are available for the camera name stamping: 0 = camera name stamping is disabled 1 = camera name stamping is on the bottom 2 = camera name stamping is on the top 3 = custom camera name stamping position In the code below, the camera stamping is set to custom stamping position.

	`payload.writeOctet(3);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x00840131, payload, response);`

2	Optional: Only necessary if option 3 (custom camera name stamping position) is chosen. The position can be set with the RCP+ command CONF_STAMP_ATTR_NAME. The first byte sets the x position (0-255) and the second byte sets the y position (0-255). The following 10 bytes are reserved and not used right now. In the code below, the camera stamping is set to left (x=5) bottom (y=245) corner.

	`payload.writeOctet(5);`
	`payload.writeOctet(245);`
	`payload.write(zeroBuffer, 9)`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09360C31, payload, response);`

Steps and codes	
3	The alarm text stamping can be changed with the RCP+ command CONF_ALARM_DISP_VAL. Following options are available for the camera name stamping: 1 = alarm stamping is disabled 2 = alarm stamping is on 3 = alarm name stamping position In the code below, the alarm stamping is set to custom stamping position.

	`payload.writeOctet(3);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x008e0131, payload, response);`

4	Optional: Only necessary if option 2 (custom alarm stamping position) is chosen. The position can be set with the RCP+ command CONF_STAMP_ATTR_ALARM. The first byte sets the x position (0-255) and the second byte sets the y position (0-255). The following 10 bytes are reserved and not used right now. In the code below, the alarm stamping is set to center (x=120) top (y=10) position.

	`payload.writeOctet(120);`
	`payload.writeOctet(10);`
	`payload.write(zeroBuffer, 9)`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09360C31, payload, response);`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_01_RCP+**

2.1.3 Picture Settings

This section describes how to get/change picture settings such as contrast, brightness, and saturation.

Integration methods:

☐ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

RCP+ over CGI

Steps and codes	
1	Read the contrast value with the RCP+ command CONF_VID_IN_CONTRAST.
	`http://<Device IP>/rcp.xml?command=0x092b &type=T_OCTET&num=1&direction=READ`
2	Read the brightness value with the RCP+ command CONF_VID_IN_BRIGHTNESS.
	`http://<Device IP>/rcp.xml?command=0x092a &type=T_OCTET&num=1&direction=READ`
3	Read the saturation value with the RCP+ command CONF_VID_IN_SATURATION.
	`http://<Device IP>/rcp.xml?command=0x092c &type=T_OCTET&num=1&direction=READ`
4	Set the contrast value with the RCP+ command CONF_VID_IN_CONTRAST.
	`http://<Device IP>/rcp.xml?command=0x092b &type=T_OCTET&num=1&direction=WRITE&payload=0x80`
5	Set the brightness value with the RCP+ command CONF_VID_IN_BRIGHTNESS.
	`http://<Device IP>/rcp.xml?command=0x092a &type=T_OCTET&num=1&direction=WRITE&payload=0x80`
6	Set the saturation value with the RCP+ command CONF_VID_IN_SATURATION.
	`http://<Device IP>/rcp.xml?command=0x092c &type=T_OCTET&num=1&direction=WRITE&payload=0x80`

RCP+ SDK

Steps and codes	
1	Read the contrast value with the RCP+ command CONF_VID_IN_CONTRAST.

	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092b0130, response);`
	`response.wait();`
	`contrast = response.readOctet();`

2	Read the brightness value with the RCP+ command CONF_VID_IN_BRIGHTNESS.

	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092a0130, response);`
	`response.wait();`
	`brightness = response.readOctet();`

3	Read the saturation value with the RCP+ command CONF_VID_IN_SATURATION.

	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092c0130, response);`
	`response.wait();`
	`saturation = response.readOctet();`

4	Set the contrast value with the RCP+ command CONF_VID_IN_CONTRAST.

	`payload.writeOctet(contrast);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092b0131, payload, response);`
	`response.wait();`

5	Set the brightness value with the RCP+ command CONF_VID_IN_BRIGHTNESS.

	`payload.writeOctet(brightness);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092a0131, payload, response);`
	`response.wait();`

6	Set the saturation value with the RCP+ command CONF_VID_IN_SATURATION.

	`payload.writeOctet(saturation);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x092c0131, payload, response);`
	`response.wait();`

2.2 Video Streaming

2.2.1 Display Video

Displaying video covers:

- Receiving video from the device
- Decoding the video
- Rendering and displaying the video

Integration methods:

☑ VideoSDK

☐ RCP+ over CGI

☐ RTSP

☐ RCP+ SDK

VideoSDK

Steps and codes	
1	Create a device connector.
	`private Bosch.VideoSDK.Device.DeviceConnector m_deviceConnector = new Bosch.VideoSDK.Device.DeviceConnector();`
2	Create a cameo to display the video.

	`/*creation of the Cameo ActiveX control*/`
	`private Bosch.VideoSDK.AxCameoLib.AxCameo m_axCameo = new Bosch.VideoSDK.AxCameoLib.AxCameo();`
	
	`/*configuration of the created control and addition to the child controls list of a GUI component*/`
	`PanelCameo.Controls.Add(m_axCameo);`
	`m_axCameo.Dock = DockStyle.Fill;`
	
	`/*retrieve the ICameo interface once the Cameo ActiveX control was visible for the first time*/`
	`private Bosch.VideoSDK.CameoLib.Cameo m_cameo = (Bosch.VideoSDK.CameoLib.Cameo)m_axCameo.GetOcx();`

3	Register to the connection event.
	`m_deviceConnector.ConnectResult += new Bosch.VideoSDK.GCALib._IDeviceConnectorEvents_ConnectResultEventHandler (DeviceConnector_ConnectResult);`
4	Connect to the BVIP device.
	`m_deviceConnector.ConnectAsync(string URL, string ProgID);`

Steps and codes	
5	When the connection is successfully established, the video stream can be assigned to the cameo to display the video.
	`private void DeviceConnector_ConnectResult(Bosch.VideoSDK.Device.ConnectResultEnum connectResult, string url, Bosch.VideoSDK.Device.DeviceProxy deviceProxy) { if (connectResult == Bosch.VideoSDK.Device.ConnectResultEnum.creInitialized) { m_cameo.SetVideoStream(deviceProxy.VideoInputs[1].Stream); } }`

```
private void
DeviceConnector_ConnectResult(Bosch.VideoSDK.Device.ConnectResultEnum
connectResult, string url, Bosch.VideoSDK.Device.DeviceProxy
deviceProxy)
{
   if (connectResult ==
      Bosch.VideoSDK.Device.ConnectResultEnum.creInitialized)
   {
      m_cameo.SetVideoStream(deviceProxy.VideoInputs[1].Stream);
   }
}
```

Further information

Video SDK Interface Definition.chm:

– Concepts > **Device Connection**
– Concepts > **Cameo ActiveX Control**
– Concepts > **Live Video**
– Class List > **ICameo**
– Class List > **IDeviceConnector**

Sample applications in the Video SDK installation directory:

– Sample Applications – Simple – **CSharpDesignerCameo**
– Sample Applications – Simple – **CSharpRuntimeCameo**
– Sample Applications – Simple – **JavaScript**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_07_VSDK**

Related sections in this document:

– **Get Stream**, see page 44

2.2.2 Get Stream

This section describes how to get the raw video stream. The stream can then be post processed (adding some overlay) or decoded using the customers own decoder.

Integration methods:

☑ VideoSDK

☐ RCP+ over CGI

☑ RTSP

☑ RCP+ SDK

VideoSDK

	Steps and codes
1	Implement the virtual StreamItemProcessor class, mainly the ProcessFrame method, which is called when a new frame is delivered from the device.

	`//Very simple implementation of the virtual StreamItemProcessor class.`
	`//It counts the I frames.`
	`   public partial class VsdkStreamItemProcessor`
	`      : Bosch.VideoSDK.StreamItemsLib.IStreamItemProcessor`
	`   {`
	`      public int m_pframes, m_iframes, m_bframes;`
	`      public VsdkStreamItemProcessor()`
	`      {`
	`         m_iframes =0;`
	`      }`
	`      public bool AcceptFrame(Guid logicalType){return true;}`
	`      public void ProcessFrame(Guid logicalType,`
	`Bosch.VideoSDK.StreamItemsLib.IStreamItem streamItem)`
	`      {`
	`         //do something with the raw stream here`
	`         if (streamItem.Flags==0x00000001)`
	`            m_iframes++;`
	`      }`
	`      public void Reset(Guid logicalType){}`
	`   }`

2	Create a new streamItemProvider and a new streamItemProcessor, connect the things together and start streaming. The PlaybackController is used to control replay sessions, which is not needed for live video.

	`m_streamItemProvider = new Bosch.VideoSDK.StreamItemsLib.`
	`StreamItemProvider();`
	`m_streamItemProvider.SelectStream(m_dataStream,`
	`VsdkGuids.LogicalChannelType_Video, 0);`
	`if (m_playbackController != null)`
	`   m_streamItemProvider.PlaybackController = m_playbackController;`
	`m_streamItemProcessor = new VsdkStreamItemProcessor();`
	`m_streamItemProvider.StreamItemProcessor = m_streamItemProcessor;`
	`m_streamItemProvider.StartStreaming();`

Further information

Video SDK Interface Definition.chm:

- Concepts > **Access to Encoded Media Data**
- Class List > **IStreamItemProcessor**
- Class List > **IStreamItemProvider**
- Class List > **IStreamItem**
- Examples > **VSDKStreamItemProcessor.cs**

Partner Area on ipp.boschsecurity.com:

- Downloads > Code Samples > **GW_01_VSDK**

Related sections in this document:

- **Display Video**, see page 42

RTSP

All RTSP options are described in the document **RTSP usage with Bosch VIP Devices** which can be found in the download area.

Further information

Partner Area on ipp.boschsecurity.com:

- Downloads > Protocols > **RTSP**

RCP+ SDK

Steps and codes

1 Connect to the device with the RtpStreamer class.

```
RtpStreamer streamer("192.168.0.1", 0);
```

2 Setup a video connection with stream properties.

In the example below H.264 video is received on the device default stream (stream 1).

```
flags = FlagH264;
coder = 0;
transmit = 0;
receive = VideoChannel;
streamer.connectVideo(0, flags, coder, transmit, receive, callback, 0);
```

3 Register to the receiver callback to receive RTP data packets.

```
streamer.registerReceiverCallback(false, myRtpCallback, 0);
```

4 Receive the RTP data in the callback function.

```
void myRtpCallback(void *ptr, RtpPayloadType payloadType, const unsigned
char *data, int len)
   {
   /*do something with the RTP data*/
   }
```

5 To terminate the RTP connection, the receiver callback pointer is set to NULL to disable the callback function address.

```
streamer.registerReceiverCallback(raw, 0, 0);
```

Further information

RCP+ SDK Documentation:

– rcpplus::RtpStreamer Class Reference

Sample applications in the RCP+ SDK installation directory:

– Examples > stream Example

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **BSS_RCP_SaR_Sample**

3 PTZ

3.1 Pan Tilt Zoom (PTZ) and Presets

This section describes how to control a PTZ device. The most commonly used functions are pan, tilt, zoom, get presets and set presets. Newer models also support absolute positioning, so the position is given/retrieved via coordinates.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Set the video controller.
	`videoInput.CameraController.SetController(string ModelName, string XmlConfig, Int32 ComPortNumber);`
2	Pan to the left with maximum speed.
	`videoInput.CameraController.PTZ(-100, 0, 0, PTZModeEnum.ptzNormalized);`
3	Tilt up with normal speed.
	`videoInput.CameraController.PTZ(0, 50, 0, PTZModeEnum.ptzNormalized);`
4	Zoom in slow.
	`videoInput.CameraController.PTZ(0, 0, 1, PTZModeEnum.ptzNormalized);`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Camera Control**

– Class List > **ICameraController**

– Examples > **VSDKCameraController.cs**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_08_VSDK**

RCP+ over CGI

There are two ways to control PTZ devices: Over the BICOM (preferred) or over the OSRD protocol. For analog cameras connected to a PTZ unit, the OSRD protocol must be chosen.

With the CONF_RCP_TRANSFER_TRANSPARENT_DATA (0xffdd) RCP+ command, the payload is redirected to the serial interface of the encoder. OSRD offers the basic functions like PTZ and get/set presets.

IP devices are still compatible with the protocol. For pure IP devices, the communication between the encoder and the analog camera part is done via the BICOM protocol.

With the CONF_BICOM_COMMAND (0x09a5) RCP+ command, the payload is redirected to the BICOM interface of the analog camera part. Together with basic PTZ and get/set presets functions BICOM offers some advanced settings which might be Intelligent Tracking, tour replay/record, etc.

Next to PTZ related commands, you can change picture settings, activate night mode or even control the wiper (depending on the device) via this interface.

OSRD

The examples below show the basic functions only using the Opcode 5 for variable-speed PTZ commands. This Opcode is supported by all Bosch devices.

For other options and the command syntax, please check the OSRD documentation.

Steps and codes	
1	Stop all PTZ operations.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000050000000C`

2	Continuous pan to the left with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000500080216`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000500780206`

3	Continuous pan to the right with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000500080115`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000500780105`

4	Continuous tilt up with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000501000815`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000050F000823`

Steps and codes	
5	Continuous tilt down with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x000000058700000501000411`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000050F00041F`

6	Continuous zoom in with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000051000203C`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000057000201C`

7	Continuous zoom out with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000051000102C`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x00000005870000057000100C`

8	Get the prepositions 1 and 99.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x0000000086000007050113`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x0000000086000007056375`

9	Set the prepositions 1 and 99.

	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x0000000086000007040112`
	
	`http://<Device IP>/rcp.xml?command=0xFFDD`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x0000000086000007046374`

BICOM

Please check the BICOM documentation for the command syntax.

The examples below shows the basic functions only.

Continuous Move

Steps and codes	
1	Stop all PTZ operations.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000000`

2	Continuous pan to the left with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085010000`
	
	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x8000060110850F0000`

3	Continuous pan to the right with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085810000`
	
	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x8000060110858F0000`

4	Continuous tilt up with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085008100`
	
	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085008F00`

5	Continuous tilt down with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000100`
	
	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000F00`

6	Continuous zoom in with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000081`
	
	`http://<Device IP>/rcp.xml?command=0x09A5`
	`&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000087`

Steps and codes	
7	Continuous zoom out with minimum and maximum speed.

	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000001`
	
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x800006011085000007`

8	Get the prepositions 1 and 99.

	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x80000201B080070501`
	
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x80000201B080070563`

9	Set the prepositions 1 and 99.

	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x80000201B080070401`
	
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x80000201B080070463`

Absolute Positioning

Steps and codes	
1	Get the pan position. The result is in radians × 10000. To get the degrees, you need to convert it first. The response from the code below is 0xc12d, which is 49453 decimal. Using the formula ((response/10000)/(2 × PI)) × 360, the pan position is 283,34 degrees.
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x810006011201`
2	Get the tilt position, using the same formula as above to get the value in degrees.
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x810006011301`
3	Get the zoom position. The zoom range is 0 to 255, where 0 means zoomed out, and 255 maximum optical zoom.
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x810006011401`
4	Set the pan position to 30 degrees, using the formula (Position/360) × 2 × PI × 10000. The position in radians × 10000 is 5236 (0x1474).
	`http://<Device IP>/rcp.xml?command=0x09A5&type=P_OCTET&direction=WRITE&num=1&payload=0x8100060112031474`

Steps and codes	
5	Set the tilt position to 180 degrees, using the formula above.
	`http://<Device IP>/rcp.xml?command=0x09A5 &type=P_OCTET&direction=W RITE&num=1&payload=0x8100060113037AB7`
6	Set a custom zoom position (0x7F).
	`http://<Device IP>/rcp.xml?command=0x09A5 &type=P_OCTET&direction=WR ITE&num=1&payload=0x810006011403007F`

Further information

Partner Area on ipp.boschsecurity.com:

- Downloads > Code Samples > **GW_02_Exe**
- Downloads > Code Samples > **GW_03_Exe**
- Downloads > Protocols > **BICOM**
- Downloads > Protocols > **OSRD**

RCP+ SDK

OSRD

The examples below show the basic functions only using the Opcode 5 for variable speed PTZ commands. This Opcode is supported by all Bosch devices.

For other options and the command syntax, please see the OSRD documentation.

Steps and codes

1 Stop all PTZ operations.

```
byte[] payloadBytes = new byte[12];

payloadBytes [0] = 0x00;     //options - currently not used
payloadBytes [1] = 0x00;     //reserved
payloadBytes [2] = 0x00;     //lease time high
payloadBytes [3] = 0x00;     //lease time low
payloadBytes [4] = 0x87;     //8x where x is the length
payloadBytes [5] = 0x00;     //camera address hi order
payloadBytes [6] = 0x00;     //camera address low order
payloadBytes [7] = 0x05;     //opcode
                             //variable speed-should be good
                             //for the most cases
payloadBytes [8] = 0x00;     //d1: 0-3 Tilt speed, 4-6 Zoom speed
payloadBytes [9] = 0x00;     //d2: 0:Focus far, 1:Iris Close,
                             //2:Iris Open, 3-6 Pan speed
payloadBytes [10] = 0x00;    //d3: 0:Pan Right, 1:Pan Left,
                             //2: Tilt Down, 3: Tilt Up
                             //4: Zoom Out, 5: Zoom In, 6: Focus Near

payloadBytes [11] = (byte)(( payloadBytes [4] +
   payloadBytes [5] +
   payloadBytes [6] +
   payloadBytes [7] +
   payloadBytes [8] +
   payloadBytes [9] +
   payloadBytes [10]) & 0x7F);  //checksum

payload.write(payloadByte, 11);
client.setNum(1);
RcpInputStream response;
client.sendRequest(0, 0xFFDD0C31, payload, response);
```

2 Continuous pan to the left.

```
payload[9]  = speed<<3;
payload[10] = 2;
```

3 Continuous pan to the right with minimum and maximum speed.

```
payload[9]  = speed<<3;
payload[10] = 1;
```

Steps and codes

4 Continuous tilt up.

```
payload[8] = speed;
payload[10] = 8;
```

5 Continuous tilt down with minimum and maximum speed.

```
payload[8] = speed;
payload[10] = 4;
```

6 Continuous zoom in.

```
payload[8] = speed<<4;
payload[9] = 32;
```

7 Continuous zoom out.

```
payload[8] = speed<<4;
payload[9] = 16;
```

8 Get/set the prepositions.

```
private void controlPrepositions(bool set, int preposNbr)
   {
   byte[] payload = new byte[11];
   payload[0] = 0x00;          //options - currently not used
   payload[1] = 0x00;          //reserved
   payload[2] = 0x00;          //lease time high
   payload[3] = 0x00;          //lease time low
   payload[4] = 0x86;          //8x where x is the length
   payload[5] = 0x00;          //camera address high order
   payload[6] = 0x00;          //camera address low order
   payload[7] = 0x07;          //opcode//Auxiliary ON/OFF and
                               //preposition
                               //SET/SHOT Commands

   if (set)
      payload[8] = 0x04;
   else
      payload[8] = 0x05;

   payload[9] = (byte)preposNbr;

   payload[10] =  (byte)((payload[4]+
   payload[5]+
   payload[6]+
   payload[7]+
   payload[8]+
   payload[9]) & 0x7F); //checksum
   }
```

BICOM

Please check the BICOM documentation for the command syntax.

The examples below shows the basic functions only.

Continuous Move

Steps and codes	
1	Stop all PTZ operations.

	`RcpClient client("<Device IP>", 0, 1);`
	`byte[] payloadBytes = new byte[12];`
	
	`payloadBytes [0] = 0x80;        //Flags available`
	`payloadBytes [1] = 0x00;        //BICOM Server ID: PTZ Server`
	`payloadBytes [2] = 0x06;        //BICOM Server ID: PTZ Server`
	`payloadBytes [3] = 0x01;        //Object ID: Position`
	`payloadBytes [4] = 0x10;        //Object ID: Position`
	`payloadBytes [5] = 0x85;        //Move Cont. with variable Speed`
	`payloadBytes [6] = 0x00;        //Pan with speed`
	`payloadBytes [7] = 0x00;        //Tilt with speed`
	`payloadBytes [8] = 0x00;        //Zoom with speed`
	`payload.write(payloadBytes, 9);`
	`client.sendRequest(0, 0x09A50C31, payload, response);`

2	Continuous pan to the left.
	`payloadBytes [6] = speed;`
3	Continuous pan to the right.
	`payloadBytes [6] = 0x80 & speed;`
4	Continuous tilt up.
	`payloadBytes [7] = 0x80 & speed;`
5	Continuous tilt down.
	`payloadBytes [7] = speed;`
6	Continuous zoom in.
	`payloadBytes [8] = 0x80 & speed;`
7	Continuous zoom out.
	`payloadBytes [8] = 0x80 & speed;`

Steps and codes

8 Get the prepositions 1 and 99.

```
private void controlPrepositions(bool set, int preposNbr)
   {
   byte[] payloadBytes = new byte[12];

   payloadBytes [0] = 0x80;        //Flags available
   payloadBytes [1] = 0x00;        //BICOM Server ID: Device Server
   payloadBytes [2] = 0x02;        //BICOM Server ID: Device Server
   payloadBytes [3] = 0x01;        //Object ID: ACTS
   payloadBytes [4] = 0xB0;        //Object ID: ACTS
   payloadBytes [5] = 0x80;        //Standardly fixed for ACTS
   payloadBytes [6] = 0x07;        //Opcode 7
   payloadBytes [7] = (set ==true)?0x04:0x05; //Get/set Prepos
   payloadBytes [8] = preposNbr;   //Prepos Number
   payload.write(payloadBytes, 9);
   client.sendRequest(0, 0x09A50C31, payload, response);
   }
```

Absolute Positioning

Steps and codes

1 Get the pan position. The result is in radians × 10000. To get the degrees, you need to convert it first using the formula below.

```
payload.writeOctet(0x81);       //flags and return payload expected
payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server
payload.writeUShort(0x0112);    //Object ID: Pan Position
payload.writeOctet(0x01);       //Get request
client.sendRequest(0, 0x09A50C30, payload, response);
response.wait();                //the last two bytes is the hex position
response.skip(6);
value = response.readUShort();
degreePan = ((double)value / 10000) / (2 * Math.PI) * 360;
```

2 Get the tilt position using the same formula as above to get the value in degrees.

```
payload.writeOctet(0x81);       //flags and return payload expected
payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server
payload.writeUShort(0x0113);    //Object ID: Tilt Position
payload.writeOctet(0x01);       //Get request
client.sendRequest(0, 0x09A50C30, payload, response);
response.wait();                //the last two bytes is the hex position
response.skip(6);
value = response.readUShort();
degreeTilt = ((double)value / 10000) / (2 * Math.PI) * 360;
```

Steps and codes	
3	Get the zoom position. The zoom range is 0 to 255, where 0 means zoomed out, and 255 maximum optical zoom.

	`payload.writeOctet(0x81);       //flags and return payload expected`
	`payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server`
	`payload.writeUShort(0x0114);    //Object ID: Zoom Position`
	`payload.writeOctet(0x01);       //Get request`
	`client.sendRequest(0, 0x09A50C30, payload, response);`
	`response.wait();                //the last two bytes is the hex position`
	`response.skip(6);`
	`zoomPos = response.readUShort();`

4	Set the pan position using the formula below.

	`payload.writeOctet(0x81);       //flags and return payload expected`
	`payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server`
	`payload.writeUShort(0x0112);    //Object ID: Pan Position`
	`payload.writeOctet(0x03);       //Set_Get request`
	`value = (ushort)((degreePan/360)*2*Math.PI*10000);`
	`payload.writeUShort(value);`
	`client.sendRequest(0, 0x09A50C30, payload, response);`

5	Set the tilt position.

	`payload.writeOctet(0x81);       //flags and return payload expected`
	`payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server`
	`payload.writeUShort(0x0113);    //Object ID: Tilt Position`
	`payload.writeOctet(0x03);       //Set_Get request`
	`value = (ushort)((degreeTilt/360)*2*Math.PI*10000);`
	`payload.writeUShort(value);`
	`client.sendRequest(0, 0x09A50C30, payload, response);`

6	Set a custom zoom position.

	`payload.writeOctet(0x81);       //flags and return payload expected`
	`payload.writeUShort(0x0006);    //BICOM Server ID: PTZ Server`
	`payload.writeUShort(0x0114);    //Object ID: Zoom Position`
	`payload.writeOctet(0x03);       //Set_Get request`
	`payload.writeUShort(zoomPos);`
	`client.sendRequest(0, 0x09A50C30, payload, response);`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **GW_02_Exe**
– Downloads > Code Samples > **GW_03_Exe**
– Downloads > Protocols > **BICOM**
– Downloads > Protocols > **OSRD**

3.2 ePTZ with ROI

This section describes how to use ROI/ePTZ.

Prerequisites:

- The seconds stream needs to be configured as ROI.
- SessionID is recommended to handle dual ROI configurations correctly.

When the region of interest (ROI) is configured electronic PTZ (ePTZ) can be used to display only part of the video.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

All ePTZ features can be used via the same methods/interfaces as described in the Section **Pan Tilt Zoom (PTZ) and Presets**, see page 47.

4 Audio

4.1 Audio Settings

This section describes how to change the audio settings. You can set the audio on/off, select the input and change the volume level.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Audio input settings are not changed with the VideoSDK, but the rendered audio volume can be adjusted in percent of the device settings. In this example the volume is set to 10 % for the audio line 1.

	`Bosch.VideoSDK.DataStream stream = m_deviceProxy.AudioInputs[1].Stream;`
	`Stream.Volume = 10;`

2	Audio output settings are not changed with the VideoSDK, but the rendered audio volume can be adjusted in percent of the device settings. In this example the volume is set to 10 % for the audio output 1.

	`Bosch.VideoSDK.DataStream stream = m_deviceProxy.AudioOutputs[1].Stream;`
	`Stream.Volume = 10;`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Live Audio**

– Class List > **IAudioInput**

– Class List > **IAudioOutput**

– Examples > **VSDKAudioInput.cs**

RCP+ over CGI

Reading the Audio Parameters

Steps and codes	
1	To get the supported audio options for audio line 1 use the RCP+ command CONF_AUDIO_OPTIONS.
	`http://<Device IP>/rcp.xml?command=0x09bf &type=T_DWORD&direction=READ&num=1`
2	Get the maximum audio input volume with the RCP+ command CONF_AUDIO_INPUT_MAX.
	`http://<Device IP>/rcp.xml?command=0x09ba &type=T_DWORD&direction=READ&num=1`
3	Get the maximum audio output volume with the RCP+ command CONF_AUDIO_OUTPUT_MAX.
	`http://<Device IP>/rcp.xml?command=0x09bb &type=T_DWORD&direction=READ&num=1`
4	Get the maximum audio mic volume with the RCP+ command CONF_AUDIO_MIC_MAX.
	`http://<Device IP>/rcp.xml?command=0x09bd &type=T_DWORD&direction=READ&num=1`

Setting the Audio Parameters

Steps and codes	
1	Set the audio mode ON with the RCP+ command CONF_AUDIO_ON_OFF.
	`http://<Device IP>/rcp.xml?command=0x000c &type=F_FLAG&direction=WRITE&payload=1`
2	Set the audio input type with the RCP+ command CONF_AUDIO_INPUT for the audio line 1 to the Line (0x0).
	`http://<Device IP>/rcp.xml?command=0x09b8 &type=T_DWORD&direction=WRITE&num=1&payload=0`
3	Change the audio input volume with the RCP+ command CONF_AUDIO_INPUT_LEVEL. In this example the volume is set to the value 10 for the audio line 1.
	`http://<Device IP>/rcp.xml?command=0x000a &type=T_DWORD&direction=WRITE&num=1&payload=0x10`
4	Change the audio output volume with the RCP+ command CONF_AUDIO_OUTPUT_LEVEL. In this example the volume is set to the value 10 for the audio line 1.
	`http://<Device IP>/rcp.xml?command=0x09b7 &type=T_DWORD&direction=WRITE&num=1&payload=0x10`

Steps and codes	
5	Change the audio mic volume with the RCP+ command CONF_AUDIO_MIC_LEVEL. In this example the volume is set to the value 10 for the audio line 1.

	`http://<Device IP>/rcp.xml?command=0x09bc`
	`&type=T_DWORD&direction=WRITE&num=1&payload=0x10`

RCP+ SDK

Reading the Audio Parameters

Steps and codes	
1	To get the supported audio options for audio line 1 use the RCP+ command CONF_AUDIO_OPTIONS.

	`ulong audioOptions;`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09bf0830, response);`
	`response.wait();`
	`audioOptions = response.readULong();`

2	Get the maximum audio input volume with the RCP+ command CONF_AUDIO_INPUT_MAX.

	`client.sendRequest(0, 0x09ba0830, response);`
	`response.wait();`
	`inputLevelMax = response.readULong();`

3	Get the maximum audio output volume with the RCP+ command CONF_AUDIO_OUTPUT_MAX.

	`client.sendRequest(0, 0x09bb0830, response);`
	`response.wait();`
	`outputLevelMax = response.readULong();`

4	Get the maximum audio mic volume with the RCP+ command CONF_AUDIO_MIC_MAX.

	`client.sendRequest(0, 0x09bd0830, response);`
	`response.wait();`
	`micLevelMax = response.readULong();`

Setting the Audio Parameters

Steps and codes	
1	Set the audio mode ON with the RCP+ command CONF_AUDIO_ON_OFF.

	`payload.writeOctet(AudioON);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x000c0031, payload, response);`
	`response.wait();`

Steps and codes	
2	Set the audio input type with the RCP+ command CONF_AUDIO_INPUT for the audio line 1. **Note:** This command is only applicable for CPP3 cameras and encoders and CPP4 HD cameras with changeable frame rate 25fps/30fps and 30fps/60fps. For other de-vices, this parameter has no influence.

	`payload.writeULong(audioInputMode);`
	`client.setNum(1);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09b80831, payload, response);`
	`response.wait();`

3	Change the audio input volume with the RCP+ command CONF_AUDIO_INPUT_LEVEL. In this ex-ample the volume is set to the value 10 for the audio line 1.

	`payload.writeULong(inputLevel);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x000a0831, payload, response);`
	`response.wait();`

4	Change the audio output volume with the RCP+ command CONF_AUDIO_OUTPUT_LEVEL. In this example the volume is set to the value 10 for the audio line 1.

	`payload.writeULong(outputLevel);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09b70831, payload, response);`
	`response.wait();`

5	Change the audio mic volume with the RCP+ command CONF_AUDIO_MIC_LEVEL. In this example the volume is set to the value 10 for the audio line 1.

	`payload.writeULong(micLevel);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x09bc0831, payload, response);`
	`response.wait();`

4.2 Audio Streaming

4.2.1 Playback Audio

Displaying video covers:

- Receiving video from the device
- Decoding the video
- Rendering and displaying the video

Integration methods:

☑ VideoSDK

☐ RCP+ over CGI

☐ RTSP

☐ RCP+ SDK

VideoSDK

Steps and codes	
1	Create a device connector.
	`private Bosch.VideoSDK.Device.DeviceConnector m_deviceConnector = new Bosch.VideoSDK.Device.DeviceConnector();`
2	Create an audio receiver to playback the audio source.

	`/*creation of Audio Receiver */`
	`private Bosch.VideoSDK.AudioLib.AudioReceiver m_audioReceiver = new Bosch.VideoSDK.AudioLib.AudioReceiver();`
	`/*set the audio input stream to the audioReceiver */`
	`m_audioReceiver.AddStream(inputStream);`

3	Register to the connection event.
	`m_deviceConnector.ConnectResult += new Bosch.VideoSDK.GCALib._IDeviceConnectorEvents_ConnectResultEventHandler (DeviceConnector_ConnectResult);`
4	Connect to the BVIP device.
	`m_deviceConnector.ConnectAsync(string URL, string ProgID);`

Steps and codes	
5	When the connection is successfully established, the video stream can be assigned to the cameo to display the video.
	`private void DeviceConnector_ConnectResult(Bosch.VideoSDK.Device.ConnectResultEnum connectResult, string url, Bosch.VideoSDK.Device.DeviceProxy deviceProxy) { if (connectResult == Bosch.VideoSDK.Device.ConnectResultEnum.creInitialized) { devProxyDst.AudioOutputs[1].SetStream(inputStream) } }`

```
private void
DeviceConnector_ConnectResult(Bosch.VideoSDK.Device.ConnectResultEnum
connectResult, string url, Bosch.VideoSDK.Device.DeviceProxy
deviceProxy)
{
if (connectResult ==
Bosch.VideoSDK.Device.ConnectResultEnum.creInitialized)
  {
  devProxyDst.AudioOutputs[1].SetStream(inputStream)
  }
}
```

Further information

Video SDK Interface Definition.chm:

– Concepts > **Live Audio**
– Class List > **IAudioReceiver**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_10_VSDK**

Related sections in this document:

– **Get Stream**, see page 65

4.2.2 Get Stream

This section describes how to get the raw audio stream.

Integration methods:

☑ VideoSDK

☐ RCP+ over CGI

☑ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Implement the virtual StreamItemProcessor class, mainly the ProcessFrame method, which is called when audio bytes available.

	`//Very simple implementation of the virtual StreamItemProcessor class.`
	
	`public partial class VsdkStreamItemProcessor`
	`   : Bosch.VideoSDK.StreamItemsLib.IStreamItemProcessor`
	`   {`
	`      public int m_numBytes;`
	`      public VsdkStreamItemProcessor()`
	`      {`
	`         m_numBytes; =0;`
	`      }`
	`      public bool AcceptFrame(Guid logicalType){return true;}`
	
	`      public void ProcessFrame(Guid logicalType,`
	`      Bosch.VideoSDK.StreamItemsLib.IStreamItem streamItem)`
	`      {`
	`         //do something with the raw stream here`
	`         if (streamItem.Payload > 0)`
	`            m_numBytes++;`
	`      }`
	`      public void Reset(Guid logicalType){}`
	`   }`

2	Create a new streamItemProvider and a new streamItemProcessor, connect the things together and start streaming. The PlaybackController is used to control replay sessions, which is not needed for live video.

	`m_streamItemProvider = new Bosch.VideoSDK.StreamItemsLib.`
	`StreamItemProviderClass();`
	`m_streamItemProvider.SelectStream(m_dataStream,`
	`VsdkGuids.LogicalChannelType_Audio, 0);`
	`if (m_playbackController != null)`
	`   m_streamItemProvider.PlaybackController = m_playbackController;`
	`m_streamItemProcessor = new VsdkStreamItemProcessor();`
	`m_streamItemProvider.StreamItemProcessor = m_streamItemProcessor;`
	`m_streamItemProvider.StartStreaming();`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Access to Encoded Media Data**
– Class List > **IStreamItemProcessor**
– Class List > **IStreamItemProvider**
– Class List > **IStreamItem**
– Examples > **VSDKStreamItemProcessor**.cs

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **GW_01_VSDK**

Related sections in this document:

– **Playback Audio**, see page 63

RTSP

All RTSP Options are described in the Document **RTSP_usage_with_Bosch_Devices** which can be found in the download area.

Partner Area on ipp.boschsecurity.com:

– Downloads > Protocols > **RTSP**

RCP+ SDK

Steps and codes	
1	Connect to the device with the RtpStreamer Class.
	`RtpStreamer streamer("192.168.0.1", 0);`
2	Setup an audio connection with stream properties. In the example below AAC audio is received on the device default stream.

	`flags = FlagAAC;`
	`coder = 0;`
	`transmit = 0;`
	`receive = AudioChannel;`
	`streamer.connectVideo(0, flags, coder, transmit, receive, callback, 0);`

3	Register to the receiver callback to receive RTP data packets.
	`streamer.registerReceiverCallback(false, myRtpCallback, 0);`
4	Receive the RTP data in the callback function.

	`void myRtpCallback(void *ptr, RtpPayloadType payloadType, const unsigned char *data, int len)`
	`{`
	`   /*do something with the RTP data */`
	`}`

5	To terminate the RTP connection, the receiver callback with is called again with disabling the callback function address.
	`streamer.registerReceiverCallback(raw, 0, 0);`

Further information

RCP+ SDK Documentation:

– rcpplus::RtpStreamer Class Reference

Sample applications in the RCP+ SDK installation directory:

– Examples > Stream example

5 Recording

5.1 Search and Replay

5.1.1 Search Video

In order to play back recordings, a search needs to be performed first to locate the desired recording.

With VideoSDK, the connection is established with the iSCSI medium itself, which means the BVIP device is not needed.

With RCP+ SDK, the connection is established to the device itself, no matter if the recordings are local, locally managed or centrally managed.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

For locally managed recordings, a connection to the device has to be established. If the recordings are centrally managed, a connection to the VRM has to be established.

First a track search is made to find out how many recording tracks are available. A camera can have either one track if single recording is active or two tracks if dual recording is active.

The track concept is very important for centrally managed recordings. In this case, the VRM has a unique TrackID for each recorded camera. This means that a camera has (up to) two tracks while a VRM can have multiple tracks, depending on how many cameras are recorded.

Once a track is selected, additional searches can be made on that track. The most common search is the **VideoRecorded** search, which shows recordings and gaps within a track.

Steps and codes	
1	Create a track search session.
	`m_trackSearchSession = m_deviceProxy.MediaDatabase.CreateSearchSession(Bosch.VideoSDK .MediaDatabase.SearchTypeEnum.steTrack);`
2	Set event handlers.

	`m_trackSearchSession.TrackAvailable += new Bosch.VideoSDK.GCALib. _ISearchSessionEvents_TrackAvailableEventHandler (m_searchSession_TrackAvailable);`
	`m_trackSearchSession.Progress += new Bosch.VideoSDK.GCALib. _ISearchSessionEvents_ProgressEventHandler (m_trackSearchSession_Progress);`

Steps and codes

3 Start the track search.

```
m_trackSearchSession.Start();
```

4 Collect the results.

```
void m_searchSession_TrackAvailable(
Bosch.VideoSDK.MediaDatabase.SearchTypeEnum Type,
Bosch.VideoSDK.MediaDatabase.Track pResult,
Bosch.VideoSDK.MediaDatabase.SearchSession pSearchSession)
{
   /*do something here */
}
```

5 Wait until the track search has finished.

```
void m_trackSearchSession_Progress(int Progress,
Bosch.VideoSDK.MediaDatabase.SeachSession pSearchSession)
{
   if (Progress == 100)
      {/*do something here*/}
}
```

6 Create an event search session.

```
m_eventSearchSession =
m_deviceProxy.MediaDatabase.CreateSearchSession(Bosch.VideoSDK
.MediaDatabase.SearchTypeEnum.steEvent);
```

7 Define the event type and track ID.

```
m_eventSearchSession.AddEventFilter(EventTypeEnum EventType);
m_eventSearchSession.AddIdentifierFilter(Int32 TrackID);
```

8 Set event handlers.

```
m_eventSearchSession.ResultAvailable += new Bosch.VideoSDK.GCALib.
_ISearchSessionEvents_ResultAvailableEventHandler
(m_eventSearchSession_ResultAvailable);
m_eventSearchSession.Progress += new Bosch.VideoSDK.GCALib.
_ISearchSessionEvents_ProgressEventHandler
(m_eventSearchSession_Progress);
```

9 Start the event search.

```
m_eventSearchSession.Start();
```

10 Collect the results.

```
void m_eventSearchSession_ResultAvailable(
Bosch.VideoSDK.MediaDatabase.SearchTypeEnum Type,
Bosch.VideoSDK.MediaDatabase.SearchResult pResult,
Bosch.VideoSDK.MediaDatabase.SearchSession pSearchSession)
{/*do something here*/}
```

Steps and codes	
11	Wait until the event search has finished.
	`void m_eventSearchSession_Progress (int Progress, Bosch.VideoSDK.MediaDatabase.SearchSession pSearchSession) { if (Progress == 100) {/*do something here*/} }`

```
void m_eventSearchSession_Progress
(int Progress, Bosch.VideoSDK.MediaDatabase.SearchSession
pSearchSession)
{
   if (Progress == 100)
      {/*do something here*/}
}
```

Further information

Video SDK Interface Definition.chm:

– Concepts > **Media Database Search**
– Class List > **IMediaDatabase**
– Class List > **ISearchSession**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_09_VSDK**

Related sections in this document:

– **Replay**, see page 75
– **Export**, see page 80

RCP+ over CGI

Note:

Search of recorded video over RTSP only works for locally managed recordings, not for centrally managed (by VRM) recordings and requires firmware version 5.70 or higher.

To enable searching on a recorded video stream a correct setup of the RTSP replay connection is preconditioned.

The track, the search is performed on, is specified in the setup of the RTSP replay connection.

Steps and codes	
1	Set up a RTSP connection for replay.
	`rtsp://<Device IP>/?rtsp_tunnel?line=1&inst=1&rec=1&rnd=718`
2	Get the RTSP session ID with the help of the random number specified in the URL used to request the recorded video stream via RTSP with the command CONF_GET_RTSP_-SESSION_ID.
	`http://<Device IP>/rcp.xml?command=0x0ae8` `&type=T_DWORD&direction=READ&protocol=TCP&num=718` The session ID is for example: 154861654
3	Start the search with the command CONF_HD_PARTITION_FILE_INFO. Set the start and end time of the search in seconds since 01.01.2000 00:00h and the max entry number. In the code below the start time is set to 29.11.2012 06:00:00 and the end time is set to 29.11.2012 06:00:00.
	`http://<Device IP>rcp.xml?command=0x0901` `&type=P_OCTET&direction=READ&protocol=TCP` `&payload=0x1849B660184A0AC0000000400000000&num=1&sessionid=154861654`
	The reply contains all slices up to the max entry number specified in the search command.

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Protocols > **RTSP**

Related sections in this document:

– **Replay**, see page 75

RCP+ SDK

Note:

For centrally managed (by VRM) recordings firmware version 5.70 and VRM version 3.0 or higher is required.

Steps and codes

1 Connect to the device with the RtpStreamer Class.

```
RtpStreamer streamer("192.168.0.1", 0);
```

2 Setup a video connection with stream properties.

In the example below H.264 video is received on the device default stream.

```
gflags = GFlag_HDD;          // for replay session
vflags = 0x40;               // H264
aflags = 0;
xflags = 0;
coder = 0;
transmit = 0;
receive = VideoChannel;
streamer.connectVideo(0, gflags, vflags, aflags, xflags, coder,
transmit, receive, callback, 0);
```

3 Search to a time where recordings exists with the with the RCP+ command CONF_HD_PARTITION_FILE_INFO.

End and start time of the slice is the time in seconds since 2000 local time. The maximum number of entries returns maximum the number of recordings that exist between the time period. The counter starts with the start time. The stop time can also be the time before the start time (searching backwards).

```
RcpOutputStream payload;

unsigned long startTime = 431308800;    // 2013-Sep-01 – 0:00:00h
unsigned long stopTime = 431568000;     // 2013-Sep-04 – 0:00:00h
payload.writeULong (startTime);         // 4 bytes: stop time of the
                                        // search
payload.writeULong (stopTime);          // 4 bytes: max number of
                                        // entries to be returned
payload.writeULong (maxEntries);        // 4 bytes reserved
payload.writeULong (0);

streamer.sendRequest (3000 /*timeout*/, 0x09010c30/*opCode*/
,sessionId, num, payload, response, this);
```

4 Register to the receiver callback to receive RTP data packets.

```
streamer.registerReceiverCallback(false, myRtpCallback, 0);
```

Steps and codes	
5	Receive the RTP data in the callback function.

	`void myRtpCallback(void *ptr, RtpPayloadType payloadType, const unsigned char *data, int len)`
	`{`
	`   /*do something with the RTP data*/`
	`}`

5	To terminate the RTP connection, the receiver callback pointer is set to NULL to disable the callback function address.
	`streamer.registerReceiverCallback(raw, 0, 0);`

Further information

RCP+ SDK Documentation:

– rcpplus::RtpStreamer Class Reference

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **BSS_RCP_SaR_Sample**

Related sections in this document:

– **Replay**, see page 75

5.1.2 Replay

Replay lets you view recorded video material. The recordings can be centrally managed (by VRM) or locally managed by the device itself.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

For locally managed recordings, a connection to the device has to be established. If the recordings are centrally managed, a connection to the VRM has to be established.

In order to replay video, the desired recording must first be located using a search. This can be done using the **Search video** function of the VideoSDK.

Steps and codes	
1	Create a playbackController to navigate in the recording.
	`m_playbackController = new Bosch.VideoSDK.MediaDatabase.PlaybackController();`
2	Connect the playbackController to a specific track which has been received with the TrackSearch session before.
	`m_mediaSession = m_mediaDatabase.GetMediaSession(m_trackInfo.TrackID, m_playbackController);`
3	Seek to the time you are interested in.
	`m_playbackController.Seek(Bosch.VideoSDK.MediaDatabase.Time64 seektime);`
4	Set the ReplaySpeed. VideoSDK supports forward (value >0), backward replay (value <0) and pause (value = 0) modes.
	`m_playbackController.Play(Int32 Rate);`
5	Assign the replay stream to a cameo.
	`m_cameo.SetVideoStream(m_mediaSession.GetVideoStream());`

Further information

Video SDK Interface Definition.chm:

– Concepts > **Media Database Replay**
– Concepts > **Cameo ActiveX Control**
– Class List > **IMediaDatabase**
– Class List > **IPlaybackController**

Sample applications in the Video SDK installation directory:

– Sample Applications – Simple – **CSharpDesignerCameo**
– Sample Applications – Simple – **CSharpRuntimeCameo**

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **IP_10_VSDK**

Related sections in this document:

– **Search Video**, see page 69
– **Export**, see page 80

RCP+ over CGI

Note:
For centrally managed (by VRM) recordings firmware version 5.70 and VRM version 3.0 or higher is required.

In case of controlling the replay of a recorded video stream a correct setup of the RTSP replay connection is preconditioned.

Steps and codes	
1	Set up a RTSP connection for replay with a random choosen rnd number.
	`rtsp://<Device IP>/?rtsp_tunnel?line=1&inst=1&rec=1&rnd=718`
2	Get the RTSP session ID with the help of the random number specified in the URL used to request the recorded video stream via RTSP with the RCP+ command CONF_GET_RTSP_SESSION_ID. In the example below, the sessionID is 154861654.

	`http://<Device IP>/rcp.xml?command=0x0ae8`
	`&type=T_DWORD&direction=READ&protocol=TCP&num=718`

3	Stop/pause the HD replay at the current position (default value is play with speed 100 %) with the RCP+ command CONF_HD_REPLAY_START.

	`http://<Device IP>/rcp.xml?command=0x0902`
	`&type=T_INT&direction=WRITE&num=1&sessionid=154861654`
	`&payload=0x00000000`

4	Seek to the time in seconds since 01.01.2000 00:00h you are interested in with the RCP+ command CONF_HD_REPLAY_SEEK_TIME. Example below: 29.11.2012 11:38:20 (local time).

	`http://<Device IP>/rcp.xml?command=0x0905`
	`&type=P_OCTET&direction=WRITE&num=1&sessionid=154861654`
	`&payload=0x184A05AC`

5	Start the playback of the recorded video stream with normal speed (100%) with the RCP+ command CONF_HD_REPLAY_START. Forward (values >0), backward (values <0)and stop/pause (value = 0) replay modes are supported.

	`http://<Device IP>/rcp.xml?command=0x0902`
	`&type=T_INT&direction=WRITE&num=1&sessionid=154861654`
	`&payload=0x00000064`

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Protocols > **RTSP**

Related sections in this document:

– **Search Video**, see page 69

RCP+ SDK

Note:

For centrally managed (by VRM) recordings firmware version 5.70 and VRM version 3.0 or higher is required.

In case of controlling the replay of a recorded video stream a correct CONF_CONNECT_PRIMITIVE connection is preconditioned (**VideoSDK**, see page 69).

Steps and codes	
1	Connect to the device with the RtpStreamer Class.
	`RtpStreamer streamer(“192.168.0.1”, 0);`
2	Setup a video connection with stream properties. In the example below H.264 video is received on the device default stream.

	`gflags = GFlag_HDD;        // for replay session`
	`vflags = 0x40;             // for H264`
	`aflags = 0;`
	`xflags = 0;`
	`coder = 0;`
	`transmit = 0;`
	`receive = VideoChannel;`
	`streamer.connectVideo(0, gflags, vflags, aflags, xflags, coder,`
	`transmit, receive, callback, 0);`

3	Seek to a time where a recording exists and the replay should start. The absolute time is the time in seconds since 2000-Jan-01 0:00:00h. If no recording time is known please see page 69 to search for recordings.

	`RcpOutputStream payload;`
	
	`unsigned long secondsSince2000 = 431308800;  // 2013-Sep-01 – 0:00:00h`
	`payload.writeULong (secondsSince2000);       // seek time`
	`payload.writeULong (0);                      // 4 bytes reserved`
	
	`streamer.sendRequest (0 /*timeout*/, 0x09050c30/*opCode*/,sessionId,`
	`num, payload, response, 0);`

4	Start the replay session. If the speed is a negative value the replay is backwards. Otherwise it is forward. To stop the replay session the value needs to be 0. The speed parameter is in percent of a real time replay (default +100%).

	`RcpOutputStream payload;`
	
	`unsigned int speed = 100;       // normal speed`
	`payload.writeUInt(speed);       // replay forward`
	
	`streamer.sendRequest(0 /*timeout*/, 0x09020430/*opCode*/,sessionId,`
	`num, payload, response, 0);`

Steps and codes	
5	Register to the receiver callback to receive RTP data packets.
	`streamer.registerReceiverCallback(false, myRtpCallback, 0);`
6	Receive the RTP data in the callback function.

	`void myRtpCallback(void *ptr, RtpPayloadType payloadType, const unsigned char *data, int len)`
	`{`
	`   /*do something with the RTP data */`
	`}`

7	To terminate the RTP connection, the receiver callback pointer is set to NULL to disable the callback function address.
	`streamer.registerReceiverCallback(raw, 0, 0);`

Further information

RCP+ SDK Documentation:

– rcpplus::RcpReplayClient Class Reference

Partner Area on ipp.boschsecurity.com:

– Downloads > Code Samples > **BSS_RCP_SaR_Sample**

Related sections in this document:

– **Search Video**, see page 69

5.2 Export

Recorded video and audio can be exported to a file on the local HDD or to FTP/cloud storage.

Integration methods:

☑ VideoSDK

☐ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

The export file is saved on local or network attached storage.

It can be in VideoSDK’s native format which means that no post processing is done when the re-cording is retrieved from the storage. It can be played back either with your own solution or via Bosch BVC or BVMS.

The file can also be stored in .asf format, which can be opened with Windows Media Player or other media players. In this case, the video data is transcoded in real time as it is received from storage, which may cause heavy CPU load.

In order to export video, a search needs to be executed (see page 69).

Steps and codes	
1	Create PlaybackController.

	`m_playbackController =`
	`Bosch.VideoSDK.MediaDatabase.PlaybackController();`

2	Init MediaFileWriter and set the export properties.

	`m_mediaFileWriter = new`
	`Bosch.VideoSDK.MediaDatabase.MediaFileWriter();`
	`m_mediaFileWriter.FileFormat = format;// asf or MPEGActiveX`
	`m_mediaFileWriter.FileSizeLimitKB = 0; // no limit`
	`m_mediaFileWriter.MaximumNumberOfFiles = 0; //no limit`
	`m_mediaFileWriter.RecordingStartTime =`
	`Bosch.VideoSDK.MediaDatabase.Time64 RecordingStartTime;`
	`m_mediaFileWriter.RecordingEndTime =`
	`Bosch.VideoSDK.MediaDatabase.Time64 RecordingEndTime;`

3	Set event handlers.

	`m_mediaFileWriter.Progress +=new`
	`Bosch.VideoSDK.GCALib._IMediaFileWriterEvents_`
	`ProgressEventHandler(m_mediaFileWriter_Progress);`
	`m_mediaFileWriter.RecordingStopped +=new`
	`Bosch.VideoSDK.GCALib._IMediaFileWriterEvents_`
	`RecordingStoppedEventHandler(m_mediaFileWriter_RecordingStopped);`
	`m_mediaFileWriter.NewFileCreated +=new`
	`Bosch.VideoSDK.GCALib._IMediaFileWriterEvents_`
	`NewFileCreatedEventHandler(m_mediaFileWriter_NewFileCreated);`

Steps and codes	
4	Set stream options.
	`m_mediaFileWriter.AddStream(Bosch.VideoSDK.DataStream Stream, MediaTypeEnum MediaType, Int32 TrackID, string TrackName, string SourceURL, Int32 SourceID);`
5	Start export.
	`m_mediaFileWriter.StartRecording(string Filename, string fileComment);`

Further information

Video SDK Interface Definition.chm:

- Concepts > **Media Database Export**
- Class List > **IMediaDatabase**
- Class List > **IPlaybackController**
- Class List > **IMediaFileWriter**

Partner Area on ipp.boschsecurity.com:

- Downloads > Code Samples > **IP_11_VSDK**
- Downloads > Code Samples > **IP_20_VSDK**

Related sections in this document:

- **Search Video**, see page 69
- **Replay**, see page 75

RCP+ SDK

Steps and codes

1 Connect to the device with the RtpStreamer Class.

```
RtpStreamer streamer(“192.168.0.1”, 0);
```

2 Setup a video connection with stream properties.

In the example below H.264 video is received on the device default stream.

```
gflags = GFlag_HDD;        // for replay session
vflags = 0x40;             // for H264
aflags = 0;
xflags = 0;
coder = 0;
transmit = 0;
receive = VideoChannel;
streamer.connectVideo(0, gflags, vflags, aflags, xflags, coder,
transmit, receive, callback, 0);
```

3 Seek to a time where a recording exists and the replay should start. The absolute time is the time in seconds since 2000-Jan-01 0:00:00h.

If no recording time is known please see page 69 to search for recordings.

```
RcpOutputStream payload;

unsigned long secondsSince2000 = 431308800;  // 2013-Sep-01 – 0:00:00h
payload.writeULong (secondsSince2000);       // seek time
payload.writeULong (0);                      // 4 bytes reserved

streamer.sendRequest (0 /*timeout*/, 0x09050c30/*opCode*/,sessionId,
num, payload, response, 0);
```

4 Start the replay session. If the speed is a negative value the replay is backwards. Otherwise it is forward. To stop the replay session the value needs to be 0. The speed parameter is in percent of a real time replay (default +100%).

Note: There is a maximum speed of four times faster than normal speed permitted to receive all I-, P- and B-frames. For all speeds faster than 400% there just the I-frames guaranteed.

```
RcpOutputStream payload;

unsigned int speed = 400;    // 4x speed -> faster export!
payload.writeUInt(speed);    // replay forward

streamer.sendRequest(0 /*timeout*/, 0x09020430/*opCode*/,sessionId,
num, payload, response, 0);
```

5 Register to the receiver callback to receive RTP data packets.

```
streamer.registerReceiverCallback(false, myRtpCallback, 0);
```

Steps and codes

6 Receive the RTP data in the callback function. Here it is the export to be define e.g. where the data should be saved etc.

```
void myRtpCallback(void *ptr, RtpPayloadType payloadType, const
unsigned char *data, int len)
{
   /*save the RTP data to a file */
}
```

7 To terminate the RTP connection, the receiver callback pointer is set to NULL to disable the callback function address.

```
streamer.registerReceiverCallback(raw, 0, 0);
```

Further information

RCP+ SDK Documentation:

– rcpplus::RtpStreamer Class Reference

– rcpplus::RcpReplayClient Class Reference

Sample applications in the RCP+ SDK installation directory:

– Examples > Stream example

6 Inputs and Outputs

6.1 I/O Settings

This section describes how to change the settings of alarm inputs/outputs.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Get the alarm input state for video line 1.
	`bool alarmInputState = m_deviceProxy.AlarmInputs[1].GetState();`
2	Get the current logical level of the alarm output (relay) line 1.
	`bool alarmOutputState = m_deviceProxy.Relays[1].GetState();`
3	Set the logical level of an alarm output state (relay) to the mode ON for the alarm output line 1.

	`bool setStateTrue = true;`
	`m_deviceProxy.Relays[1].SetState(setStateTrue);`

Further information

Video SDK Interface Definition.chm:

- Concepts > **Relay Control**
- Class List > **IRelay**
- Class List > **IAlarmInput**
- Examples > **VSDKRelay.cs**
- Examples > **VSDKAlarmInput.cs**

Related sections in this document:

- **General Event Handling**, see page 89

RCP+ over CGI

Reading the I/O Parameters

Steps and codes	
1	Get the state for the first alarm input state with the RCP+ command CONF_INPUT_PIN_STATE.
	`http://<Device IP>/rcp.xml?command=0x01c0 &type=F_FLAG&direction=READ&num=1`
2	Get the current logical level of the first alarm output (relay) the RCP+ command CONF_RELAY_OUTPUT_STATE.
	`http://<Device IP>/rcp.xml?command=0x01c1 &type=F_FLAG&direction=READ&num=1`
3	Get the state for the first virtual alarm with the RCP+ command CONF_VIRTUAL_ALARM_STATE.
	`http://<Device IP>/rcp.xml?command=0x0a8b &type=F_FLAG&direction=READ&num=1`

Setting the I/O Parameters

Steps and codes	
1	Set the logical level of the first alarm output (relay) with the RCP+ command CONF_RELAY_OUTPUT_STATE.
	`http://<Device IP>/rcp.xml?command=0x01c1 &type=F_FLAG&direction=WRITE&num=1&payload=1`
2	Set the first virtual alarm with the RCP+ command CONF_VIRTUAL_ALARM_STATE.
	`http://<Device IP>/rcp.xml?command=0x0a8b &type=F_FLAG&direction=WRITE&num=1&payload=1`

Further information

Related sections in this document:

– **General Event Handling**, see page 89

RCP+ SDK

Reading the I/O Parameters

Steps and codes	
1	Get the state for the first alarm input state with the RCP+ command CONF_INPUT_PIN_STATE.

	`m_RcpClient.setNum(InputNumber);`
	`m_RcpClient.sendRequest(2000, (OpCode)0x01c00030, m_RcpInputStream);`
	`m_RcpInputStream.wait();`
	`bool state = m_RcpInputStream.readOctet();`

2	Get the current logical level of the first alarm output (relay) the RCP+ command CONF_RELAY_OUTPUT_STATE.

	`m_RcpClient.setNum(OutputNumber);`
	`m_RcpClient.sendRequest(2000, (OpCode)0x01c10030, m_RcpInputStream);`
	`m_RcpInputStream.wait();`
	`bool state = m_RcpInputStream.readOctet();`

3	Get the state for the first virtual alarm with the RCP+ command CONF_VIRTUAL_ALARM_STATE.

	`client.setNum(virtualAlarmNbr);`
	`client.sendRequest(0, 0x0a8b0030, response);`
	`response.wait();`
	`bool state = m_RcpInputStream.readOctet();`

Setting the I/O Parameters

Steps and codes	
1	Set the logical level of the first alarm output (relay) with the RCP+ command CONF_RELAY_OUTPUT_STATE.

	`client.writeOctet(On);`
	`client.setNum(OutputNumber);`
	`client.sendRequest(2000, (OpCode)0x01c10031, payload, response);`

2	Set the first virtual alarm with the RCP+ command CONF_VIRTUAL_ALARM_STATE.

	`payload.writeOctet(VirualAlarmState);`
	`client.setNum(virtualAlarmNbr);`
	`RcpInputStream response;`
	`client.sendRequest(0, 0x0a8b0031, payload, response);`
	`response.wait();`

Further information

Related sections in this document:

– **General Event Handling**, see page 89

7 Events

7.1 General Event Handling

Bosch video over IP devices send events which the user can choose to receive or not. They can be used to check device status like video loss, detect a broken/disconnected video cable or implement your own alarm scenarios when motion/inputs are triggered.

Integration methods:

☑ VideoSDK

☑ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

VideoSDK

Steps and codes	
1	Register to the alarm input event handler to handle input events.
	`m_alarmInput.StateChanged += new Bosch.VideoSDK.GCALib._IAlarmInputEvents_StateChangedEventHandler(AlarmInput_StateChanged);`
2	Get the input events.
	`private void AlarmInput_StateChanged(Bosch.VideoSDK.Live.AlarmInput eventSource, bool state) {/*do something here*/}`
3	Register to the output (relay) events.
	`m_relay.StateChanged += new Bosch.VideoSDK.GCALib._IRelayEvents_StateChangedEventHandler (Relay_StateChanged);`
4	Register to video loss and motion events.
	`m_videoInput.SignalStateChanged += new Bosch.VideoSDK.GCALib._IVideoInputEvents_SignalStateChangedEventHandler(VideoInput_SignalStateChanged);`
5	Get the video loss and motion events.
	`private void VideoInput_SignalStateChanged(Bosch.VideoSDK.Live.VideoInput event-Source, bool signalState, bool motion, bool alarm) {/*do something here*/}`
6	Register to recording status events.
	`m_videoInputRecordingEvents.RecordingStateChanged += new Bosch.VideoSDK.GCALib._IVideoInputRecordingEvents_RecordingStateChangedEventHandler (VideoInputRecordingEvents_RecordingStateChanged);`

Steps and codes	
7	Get the recording status events.
	`private void VideoInputRecordingEvents_RecordingStateChanged (Bosch.VideoSDK.Live.VideoInput eventSource, Bosch.VideoSDK.GCALib.RecordingStateEnum recordingState) {/*do something here*/}`

Further information

Video SDK Interface Definition.chm:

- Concepts > Fundamentals > **Event Handling in Managed Applications**
- Class List > **IAlarmInputEvents**
- Class List > **IRelayEvents**
- Class List > **IVideoInputStateEvents**
- Class List > **IVideoInputRecordingEvents**

Partner Area on ipp.boschsecurity.com:

- Downloads > Code Samples > **IP_04_VSDK**

RCP+ over CGI

RCP+ messages will be processed and received by the CGI client using a poll mechanism. A CGI command set with the requested messages command tag numbers need to be issued. After issuing a request, the reply returns immediately after a message has been sent or the default timeout of 1000 ms has been expired. When the timeout expires, a message count of 0 is returned. Otherwise the number of received messages is signaled. The default timeout can be altered by setting the 'collect-ms' CGI value to the appropriate number of milliseconds.

In order to ensure correct assignment of messages in case several clients are polling for messages or if a polling client uses different socket connections, a unique message domain ID (CGI parameter 'msgdomainID') should be provided. Messages are then collected separately for each ID and can be uniquely assigned to the polling client even if requesting via different socket connections.

Steps and codes	
1	Subscribe to messages. In the example below, subscription is done for motion alarm and input pin state messages.
	`http://<Device IP>/rcp.xml?message=0x01c0$0x01c3&collectms=5000`

Further information

Partner Area on ipp.boschsecurity.com:

- Downloads > Protocols > **RCP+ over CGI**

RCP+ SDK

	Steps and codes
1	Subscribe to messages.

```
OpCode[] p = new OpCode[6];
p[0] = (OpCode)0x01c00c30;        /*Input change*/
p[1] = (OpCode)0x01c10030;        /*Relay change*/
p[2] = (OpCode)0x01c20030;        /*Video Loss*/
p[3] = (OpCode)0x01c30030;        /*Motion alarm*/
p[4] = (OpCode)0x0aae0c30;        /*Recording Status*/
p[5] = (OpCode)0x08070c30;        /*IVA Alarms*/
m_rcpClient.subscribeMessageCallback(MessageCallback, 0, p, 6);
```

2	Receive messages in the callback function.

```
private void MessageCallback(Object obj, RcpHeader hdr, RcpInputStream
ris)
{
   /*do something here*/
}
```

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Protocols > **RCP+ over CGI**

7.2 Receive Video Analysis Events

Bosch video over IP devices with build in intelligent video analysis (IVA) are able to receive IVA alarms by recognizing it’s type (object in field, crossing line, etc.) and the task name.

Integration methods:

☐ VideoSDK

☐ RCP+ over CGI

☐ RTSP

☑ RCP+ SDK

RCP+ SDK

Steps and codes	
1	Retrieve the alarm task types and names with the RCP+ command CONF_VIPROC_RE_TASK_NAMES.

```
bool CConfViprocReTaskNames::parse(rcpplus::RcpHeader &hdr,
rcpplus::RcpInputStream &payload)
{
   // read the payload command header (2byes)
   m_vcaType = payload.readUShort();

   // read number of tasks (2bytes)
   m_numOfTasks = payload.readUShort();
   SViprocTask taskDescr;
   // read each task
   for (unsigned short iter = 0; iter < m_numOfTasks; iter++)
   {
      // read the ID of the task
      taskDescr.taskId   = payload.readOctet();
      // read the type of the task
      taskDescr.taskType = payload.readOctet();

      // read the description of the task
      for (int descrIter = 0; descrIter < CVRTNDescriptionLength;
      descrIter++)
         taskDescr.taskDescription [descrIter] = payload.readOctet();

      // push back the complete task description struct into the vector
      // for later refernce
      m_taskList.push_back(taskDescr);
   }

   // we are through
   return true;
}
```

Steps and codes

2 Subscribe to the IVA alarm message.

```
OpCode[] p = new OpCode[1];
p[0] = (OpCode)0x08070c30;      /*IVA Alarms*/
m_rcpClient.subscribeMessageCallback(MessageCallback, 0, p, 1);
```

3 Receive messages in the callback function.

```
private void MessageCallback(Object obj, RcpHeader hdr, RcpInputStream
ris)
{
   if (hdr.OpCode == 0x08070c30)
   {
      parseIVAMessages()
   }
}
```

4 Read out the ID for identification of the IVA profile which caused the alarm, the alarm flags to identify if this is a motion or tamper alarm, and the IVA Task ID, to know which IVA task caused the event.

```
void parseIVAMessages()
{
   m_alarmFlags = payload.readUShort();
   m_IVA_ID = payload.readUShort();
   m_configId = payload.readOctet();
```

Steps and codes	
5	Parse the alarm flags.

```
//continuation of the parseIVAMessages() function
if (m_alarmFlags & 0x8000)
{
   os <<"\t - Motion Alarm" << std::endl;
}
if (m_alarmFlags & 0x4000)
{
   os <<"\t - Global Change flag" << std::endl;
}
if (m_alarmFlags & 0x2000)
{
   os <<"\t - Signal too bright" << std::endl;
}
if (m_alarmFlags & 0x1000)
{
   os <<"\t - Signal too dark" << std::endl;
}
if (m_alarmFlags & 0x800)
{
   os <<"\t - Signal too noisy" << std::endl;
}
if (m_alarmFlags & 0x400)
{
   os <<"\t - Signal too blury" << std::endl;
}
if (m_alarmFlags & 0x200)
{
   os <<"\t - Signal loss" << std::endl;
}
if (m_alarmFlags & 0x100)
{
   os <<"\t - Reference image check failed flag" << std::endl;
}
if (m_alarmFlags & 0x80)
{
   os <<"\t - Invalid config" << std::endl;
}
if (m_alarmFlags == 0x0)
{
   os <<"\t - No alarms pending" << std::endl;
}
```

Steps and codes

6 In case of motion alarm, get the IVA task ID.

```
unsigned short mask = 0x1;
// 16 ID’s at max
for (unsigned short iter = 1; iter <= 16; iter++)
{
   if ( (m_IVA_ID  & mask) > 0 )
   {
      mapIDtoName(mask)
   }
   mask = mask << 1;
}
```

7 With the ID, the type and name can be retrieved.

```
void mapIDtoName(char TaskID)
{
for (std::vector <Bosch::RCPPParser::SViprocTask>::iterator
   idIter = p_vtd->begin();
   idIter != p_vtd->end();
   ++idIter)

if (idIter->taskId == TaskID)
   std::cout << "\t\tactive Alarm Id: " << (int)idIter->taskId
   << " " << idIter->taskDescription << std::endl;
}
```

Further information

Partner Area on ipp.boschsecurity.com:

– Downloads > Protocols > **RCP+ over CGI**

– Downloads > Code Samples > **IP_02_RCP+**

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