SVGigE API

GigE Vision Camera

Software Development Kit (SDK)

On Linux platforms

Version: 2.4.7
Date: 2021.03

Content

Overview 4
General System Requirements: 4
Prerequisites 4
SVGigE SDK components 4
Installation 5
System tuning 5
Sample program 7
Operating a camera (fixed frequency) 8
Operating a camera (software trigger) 8
Implementing functionality into own applications 9
Application callback 9
Find network adapters 10
Device discovery 10
Force IP 10
Open/Close camera 10
Add/Close stream 11
Enable stream 12
Set/Get acquisition control 12
Set/Get acquisition mode 12
Start acquisition cycle 12
Set/Get framerate, exposure, gain, offset 12
Strobe light control 13
Get imager size 13
Get image size 13
Reducing image size by binning 14
Reducing image size by an AOI (area of interest) 14
Pixel information 14
Decoding 12-bit Images 14
Camera information 15
Stream statistics 16
Auto gain/exposure 16

Decoding error messages 17
Show version information about the SDK 17
IO Configuration functions 17
Sequencer 18
Distributing image streams by multicast 19

Overview

The GigE Vision standard is an award-winning specification for connecting cameras to computers by a Gigabit Ethernet cable. That type of connection is well suited for transferring the huge data streams that are usually related to image capturing devices. The current SVS GigE API software development kit (Linux SDK) provides for integrating GigE cameras into own applications on Linux platforms

General System Requirements:

The libraries have been built with the following system environment.

32Bit:

Ubuntu	8.04
gcc	4.2.4
glibc	2.7
libstdc++	4.2.0
ld	2.7

64Bit:

Ubuntu	8.04
gcc	4.2.4
glibc	2.7
libstdc++	4.2.0
ld	2.7

Prerequisites

In order to give an example for data transfer requirements, a camera with a resolution of 1600 by 1200 pixels and which is running at 35 frames per second will generate a data stream of 1600 x 1200 x 35 = 67,2 Mbytes per second. The bandwidth of the network connection must be capable of transferring the image data stream from the camera to the client computer with the image viewing or processing application.

SVGigE SDK components

The SVGigE SDK consists of the following components which become available after unpacking delivered package:

tar xzf SVCam_GigE_Linux_SDK_2.x.x.tar.gz

- bin/libsvgige.so.2.x.x
- include/svgige.h
- Makefile

- console/<sample sources>
- doc/SVGigE Linux SDK user guide.pdf (this document)
- sys_adjust
- example/<mini example>
- example_Multicast_Controller/<mini example>
- example_Multicast_Listener/<mini example>

Installation

Installation of the software requires running the Makefile as root. Subsequently the Makefile in 'console' can be used for compiling a sample program called "gec".

System tuning

Operating a GigE camera will usually challenge a system's performance. Some system tuning steps might be necessary for getting to a smooth operation at highest frame rate that a camera can deliver. In the following some recommendations will be given what system parameters can be used for improving overall performance for operating GigE cameras.

The following system adjustments, if applicable, should be executed before a GigE camera is operated:

- **ifconfig eth0 mtu 9000**
- **echo 2000000 > /proc/sys/net/core/rmem_max**
- **echo 2000000 > /proc/sys/net/core/rmem_default**
- **echo 10000 > /proc/sys/net/core/netdev_max_backlog**

The MTU size in the first command should be set to the maximal packet size that is supported by network hardware. Please refer to the network card's documentation.

Remaining commands adjust a system's read and backlog buffers in order to deal with a huge image data stream in an optimal way. They can also be written into the /etc/sysctl.conf file as follows:

- net.core.rmem max=2000000
- net.core.rmem default=2000000
- net.core.netdev.max_backlog=10000

The packet size that has been adjusted above has to be used by the application when opening a streaming channel in order to take advantage of those jumbo frames.

Besides tuning the MTU and buffer size it is also recommended to accelerate the system's scheduler in order to process all network packets in time. Since changing the scheduler cycle might have impact also on other applications, one has to check that it is not in conflict with

other application's requirements. Usually the only visible effect should be a slightly higher system load, if at all.

The new scheduler settings can be written into /etc/sysctl.conf file as follows:

- kernel.sched_min_granularity_ns=1000000
- kernel.sched_latency_ns=1000000
- kernel.sched_wakeup_granularity_ns=1000000
- kernel.sched_batch_wakeup_granularity_ns=1000000

For system tuning one should start operating a camera in asynchronous mode which will decouple image transfer and image processing. The image transfer can subsequently be adjusted by described tuning parameters. In the result there should be no frame loss in the streaming channel.

Otherwise the

- **buffer count**

should be increased for the streaming channel. Once there is no frame loss in asynchronous mode the application can be switched to synchronous mode.

Application example

When a system has been tuned according to previous hints then the following figures should be received when operating a SVS340CUGE camera (640 x 480, dual tap):

- 480 lines: 264 fps(max lines)
- 233 lines: 500 fps
- 140 lines: 757 fps
- 93 lines: 1000 fps
- 25 lines: 2000 fps
- 4 lines: 2941 fps(max frames)

Please make sure that parameters in gec.h have been adjusted to the following values and the gec program recompiled in order to achieve above figures:

#define BUFFER_COUNT 100

#define PACKET_SIZE 9000

Sample program

Command	Argument	Description
ag	<arg>	auto gain
cc		close connection
co	<ID>	connect (by ID)
dd		device discovery
demo		programmer interface demo
dl	<arg>	inter-packet delay
ea		end acquisition
ex	<arg>	exposure [ms]
fi	<mac> <ip> <subnet mask>	change camera ip
fr	<arg>	frame rate (for fixed frequency) [fps]
gn	<arg>	gain [dB] (auto gain = off)
ls		list
ox	<x>	AOI offset X
oy	<y>	AOI offset Y
q		quit
ni		network interfaces
sa		start acquisition
si	<ip><sm>	set IP (IP address or ‘off’, subnet mask)
so		stream open
st		start acquisition (for software trigger)
stat		toggle statistics on/off
sx	<w>	AOI width
sy	<h>	AOI height
tr	<0>	fixed frequency (free running)
tr	<1>	internal trigger (software)
tr	<2>	external trigger, internal exposure
tr	<3>	external trigger, external exposure

Operating a camera (fixed frequency)

A basic run for free-running mode will involve the following commands:

dd	Device discovery, all cameras in a network will be listed
co 1	Connect to first camera
tr 0	Switch camera to fixed-frequency mode
so	Open a stream for receiving images, a window will open
sa	Start acquisition, a live image will be displayed

When acquisition has started the camera can be adjusted as follows

fr 25.0	Adjust framerate to 25 frames per second
ex 40.0	Adjust exposure time to 40 ms
gn 3.0	Adjust gain to 3.0 dB

The camera will be closed by the following commands:

ea	Acquisition will be stopped
sc	Streaming channel will be closed
cc	Connection to the camera will be closed

Operating a camera (software trigger)

A basic run for software trigger mode will involve the following commands:

dd	Device discovery, all cameras in a network will be listed
co 1	Connect to first camera
tr 1	Switch camera to software trigger mode
so	Open a stream for receiving images, a window will open
sa	Start acquisition, a live image will be displayed

When acquisition has been enabled, an image will arrive after each trigger:

st	Release a first software trigger
st	… repeat software triggers
ex 40.0	Adjust exposure time to 40 ms
gn 3.0	Adjust gain to 3.0 dB
st	Capture an image with new settings

The camera will be closed by the following commands

ea	Acquisition will be stopped
sc	Streaming channel will be closed
cc	Connection to the camera will be closed

Implementing functionality into own applications

The source code of the sample command line program may serve as a template for own applications. In the following the major SDK functions will be discussed how they can be used for controlling a camera and for receiving images.

Application callback

All output from the GigE SDK module is delivered to an application by an ApplicationCallback:

```
SVGigE_RETURN ApplicationCallback(SVGigE_SIGNAL *Signal, void* Context);
```

The ApplicationCallback is a function in the scope of an application which pointer is registered against the SDK for receiving callbacks when new data is available. The payload data is delivered by a pointer which is part of a signal structure. Additionally, a context is delivered along with a signal in order to allow an application for referencing to the object that is supposed to process that data.

The following table shows how to use the “SVGigE_SIGNAL”:

Supported signal type for SignalType field	Structure used for Data field	Explanation:
SVGigE_SIGNAL_CAMERA_FOUND	SVGigE_CAMERA	By using the “SVGigE_CAMERA”-Structure to get information about the camera. Usually it should be used in the callback for discoverCamerasXXX().
SVGigE_SIGNAL_FRAME_COMPLETED	SVGigE_IMAGE	By using the “SVGigE_IMAGE”-Structure to get image and some image-specific info. Usually it should be used in the callback for addStreamXXX().
SVGigE_SIGNAL_FRAME_ABANDONED	NULL	This means the “data field” is empty. We should not dereference the void pointer. It is just an indicator that an image is lost.
SVGigE_SIGNAL_START_OF_TRANSFER	NULL	This means the “data field” is empty. We should not dereference the void pointer. It is just an indicator that an image is started to transfer.

Find network adapters

Determining installed network adapters will usually be the first step in a program.

```
SVGigE_RETURN findNetworkAdapters(unsigned int *Adapters, unsigned char Size);
```

The function returns a list of all available IPs in an array of specified size.

Device discovery

For each installed network adapter, a discover will return all cameras that are connected to a particular network interface.

```
SVGigE_RETURN discoverCameras(unsigned int SourceIP,
                              unsigned long DiscoveryTimeout,
                              ApplicationCallback Callback,
                              void *Context);
```

The function will receive a pointer to an ApplicationCallback function where found cameras will be reported to. A Context pointer may be provided that is passed through from application to the SDK and back to the application's callback function.

Force IP

Successfully operating a camera requires that its IP address/subnet mask matches the interface's network settings where the camera is connected to.

```
SVGigE_RETURN forceIP_MAC(char * MAC,
                          unsigned int IP,
                          unsigned int Subnet,
                          unsigned int SourceIP);
```

A camera with a specified 'MAC' Address will be requested to adjust its network settings to 'IP' and 'Subnet'. The request will be sent out over the network interface 'SourceIP'. The MAC parameter can be either the camera's mac address (last four hex digits without delimiter) or the entire mac Address with colons ":" or hyphens "-" delimiter.

Open/Close camera

When opening a connection to a camera, then a channel will be established to the camera's control port and the camera will be occupied by the application. Therefore this function has to be successfully passed before being able to communicate with a camera.

```
SVGigE_RETURN openCamera(Camera_handle *Camera,
                         unsigned int CameraControlIP,
                         unsigned int SourceIP,
                         unsigned int Heartbeat)
                         MULTICAST_MODE MulticastMode);
```

When closing the connection, the camera's exclusive occupation by the application will be released. The camera will also be released when a heartbeat is missing for more than a specified timeout (usually 3 seconds).

NOTE: A common problem for debug sessions is losing connection to the camera by a heartbeat timeout. A recommended procedure is restarting the application and going through the 'openCamera' function again after a breakpoint was reached.

Add/Close stream

A stream will be opened for receiving image data from a camera. Besides parameters like BufferSize, BufferCount, PacketSize, PacketResendTimeout which characterize a stream itself, an ApplicationCallback function pointer will be provided for receiving image callbacks in the application. Further the local socket's settings for IP and Port will be returned as well as a stream handle.

```
SVGigE_RETURN addStream(Camera_handle Camera,
                        Stream_handle *Stream,
                        unsigned int *LocalIP,
                        unsigned short *LocalPort,
                        int BufferSize,
                        int BufferCount,
                        int PacketSize,
                        int PacketResendTimeout,
                        ApplicationCallback Callback,
                        void *Context);
```

As soon as a stream has been successfully opened, the application is ready for receiving image data from the camera.

The above shown function will ask the operating system for a free network port where image data can be streamed to. That port is returned with 'LocalPort' and it might be a number between 1024 and 65535, dependent on the operating system.

Whenever the port assignment should be limited to a smaller range, e.g. for setting a firewall appropriately, then the extended function for adding a stream can be used. That function allows for specifying an upper limit for port assignment:

```
SVGigE_RETURN addStreamExt(Camera_handle Camera,
                           Stream_handle *Stream,
                           unsigned int *LocalIP,
                           unsigned short *LocalPort,
                           unsigned short MaxPort,
                           int BufferSize,
                           int BufferCount,
                           int PacketSize,
                           int PacketResendTimeout,
                           ApplicationCallback Callback, void *Context);
```

If for example MaxPort =1050 is provided as a parameter then the range for assigning a streaming port will be limited to the range 1024..1050. If the specified port 1050 is still free then that port will be used. Otherwise some lower port numbers from 1049..1024 will be evaluated and the first found free port will be used.

Enable stream

An established stream will receive image data, whenever the camera has captured an image and becomes ready for transferring image data to an application. Before the cameras settings for e.g. the trigger mode are changed, the stream channel should be disabled in order to avoid the delivery of unexpected pending images.

```
SVGigE_RETURN enableStream(Stream_handle Stream, bool Enabled);
```

After switching to a new trigger mode the stream can be enabled again.

Set/Get acquisition control

A camera's actual image acquisition can be started/stopped by the following function:

```
SVGigE_RETURN Camera_setAcquisitionControl(Camera_handle Camera,
                                           ACQUISITION_CONTROL AcquisitionControl);
```

After stopping acquisition, a possibly pending image can still be delivered to the application in the streaming channel.

Set/Get acquisition mode

Before switching between acquisition modes, a camera's acquisition has to be stopped. Otherwise this will be done implicitly. The following function can be used for switching to a new acquisition mode. Additionally it can be instructed to start camera's acquisition immediately by setting "AcquisitionStart" to true.

```
SVGigE_RETURN Camera_setAcquisitionMode(Camera_handle Camera,
                                        ACQUISITION_MODE AcquisitionMode,
                                        bool AcquisitionStart);
```

When adjusting to any of the trigger modes, an external (hardware) or internal (software) trigger has to be released before the camera will do image exposure and image delivery to the application.

Start acquisition cycle

When the camera has been switched to internal (software) trigger mode, then the following function will trigger a single image:

```
SVGigE_RETURN Camera_startAcquisitionCycle(Camera_handle Camera);
```

The image will be delivered as usual in the image callback.

Set/Get framerate, exposure, gain, offset

A camera's acquisition parameters can be adjusted by following functions:

```
SVGigE_RETURN Camera_setFrameRate(Camera_handle Camera, float Framerate);

SVGigE_RETURN Camera_setExposureTime(Camera_handle Camera, float ExposureTime);

SVGigE_RETURN Camera_setGain(Camera_handle Camera, float Gain);

SVGigE_RETURN Camera_setOffset(Camera_handle Camera, float Offset);
```

All parameter's current settings can be queried by appropriate 'get' functions.

Strobe light control

A strobe light controller can be connected to one of the camera's output pins, e.g. to OUT1 by default. The following functions will control generated strobe light pulse:

```
SVGigE_RETURN Camera_setStrobePolarity(Camera_handle Camera,
                                       STROBE_POLARITY StrobePolarity);

SVGigE_RETURN Camera_getStrobePolarity(Camera_handle Camera,
                                       STROBE_POLARITY *StrobePolarity);

SVGigE_RETURN Camera_setStrobePosition(Camera_handle Camera,
                                       float StrobePosition);

SVGigE_RETURN Camera_getStrobePosition(Camera_handle Camera,
                                       float *StrobePosition);

SVGigE_RETURN Camera_getStrobePositionMax(Camera_handle Camera,
                                       float *StrobePositionMax);

SVGigE_RETURN Camera_getStrobePositionIncrement(Camera_handle Camera,
                                       float * StrobePositionIncrement);

SVGigE_RETURN Camera_setStrobeDuration(Camera_handle Camera,
                                       float StrobeDuration);

SVGigE_RETURN Camera_getStrobeDuration(Camera_handle Camera,
                                       float *StrobeDuration);

SVGigE_RETURN Camera_getStrobeDurationMax(Camera_handle Camera,
                                       float *StrobeDurationMax);

SVGigE_RETURN Camera_getStrobeDurationIncrement(Camera_handle Camera,
                                       float *StrobeDurationIncrement);
```

Get imager size

The following functions report a camera's capabilities regarding image resolution:

```
SVGigE_RETURN Camera_getImagerWidth(Camera_handle Camera,
                                    unsigned int *ImagerWidth);

SVGigE_RETURN Camera_getImagerHeight(Camera_handle Camera,
                                    unsigned int *ImagerHeight);
```

Actual image size can be equal or less the reported image resolution.

Get image size

The following functions query for a camera's actual image size:

```
SVGigE_RETURN Camera_getSizeX(Camera_handle Camera, unsigned int *SizeX);

SVGigE_RETURN Camera_getSizeY(Camera_handle Camera, unsigned int *SizeY);
```

Actual image size will be according to an eventual previous AOI setting.

Reducing image size by binning

Binning combines the output signal from multiple pixels in horizontal or vertical or in both directions before ADC stage.

```
SVGigE_RETURN Camera_setBinningMode(Camera_handle Camera,
                                    BINNING_MODE BinningMode);
```

When switching binning on, then the maximal image resolution will shrink appropriately.

Reducing image size by an AOI (area of interest)

An AOI can be set arbitrarily inside the boundaries of maximal image resolution.

```
SVGigE_RETURN Camera_setAreaOfInterest(Camera_handle Camera,
                                       unsigned int SizeX,
                                       unsigned int SizeY,
                                       unsigned int OffsetX,
                                       unsigned int OffsetY);
```

Acquisition has to be turned off and a streaming channel be closed before adjusting a new AOI. After calling above function the camera will report a new image size when using the 'getSizeX/Y' functions.

Pixel information

The pixel type can be retrieved and the pixel depth can be set/get as follows:

```
SVGigE_RETURN Camera_getPixelType(Camera_handle Camera,
                                  GVSP_PIXEL_TYPE *PixelType);

SVGigE_RETURN Camera_setPixelDepth(Camera_handle Camera,
                                   SVGIGE_PIXEL_DEPTH PixelDepth);

SVGigE_RETURN Camera_getPixelDepth(Camera_handle Camera,
                                   SVGIGE_PIXEL_DEPTH *PixelDepth);
```

Decoding 12-bit Images

When a camera is switched to 12-bit mode the received image usually needs decoding in order to further process that image e.g. for displaying purposes (8-bit needed) respectively storage purposes (most often 16-bit needed). A conversion will be based on the layout of 12-bit data where 2 pixels are folded into 3 bytes in the following way:

Pixel A												Pixel B											
11	10	9	8	7	6	5	4	3	2	1	0	3	2	1	0	11	10	9	8	7	6	5	4
Byte 0								Byte 1								Byte 2							

The SDK provides for the following functions that perform conversion from 12-bit to 8¬ bit and 16-bit as well as from 16-bit to 8-bit:

```
Image_getImage12bitAs8bit(unsigned char *ImageData,
                          int ImageWidth,
```

```
                                int ImageHeight,
                                GVSP_PIXEL_TYPE PixelType,
                                unsigned char *Buffer8bit,
                                int BufferLength);

Image_getImage12bitAs16bit(unsigned char *ImageData,
                                int ImageWidth,
                                int ImageHeight,
                                GVSP_PIXEL_TYPE PixelType,
                                unsigned char *Buffer8bit,
                                int BufferLength);

Image_getImage16bitAs8bit(unsigned char *ImageData,
                                int ImageWidth
                                int ImageHeight,
                                GVSP_PIXEL_TYPE PixelType,
                                unsigned char *Buffer8bit,
                                int BufferLength);
```

Camera information

The following functions are available for querying general information from a camera:

```
char * Camera_getManufacturerName(Camera_handle Camera);

char * Camera_getModelName(Camera_handle Camera);

char * Camera_getDeviceVersion(Camera_handle Camera);

char * Camera_getManufacturerSpecificInformation(Camera_handle Camera);

char * Camera_getSerialNumber(Camera_handle Camera);

char * Camera_getUserDefinedName(Camera_handle Camera);

char * Camera_getIPAddress(Camera_handle Camera);

char * Camera_getSubnetMask(Camera_handle Camera);

char * Camera_getMacAddress(Camera_handle Camera);
```

Further, a user-defined name can be uploaded to the camera:

```
SVGigE_RETURN Camera_setUserDefinedName(Camera_handle Camera,
                                        char *UserDefinedName);
```

An uploaded user-defined name will be returned on every discovery run.

Stream statistics

Stream statistics are available for frame loss, frame rate and data rate. They can be obtained by the following functions:

```
SVGigE_RETURN StreamingChannel_getFrameLoss(Stream_handle Stream,
                                            int *FrameLoss);

SVGigE_RETURN StreamingChannel_getActualFrameRate(Stream_handle Stream,
                                            float *ActualFrameRate);

SVGigE_RETURN StreamingChannel_getActualDataRate(  Stream_handle Stream,
                                            float *ActualDataRate);

SVGigE_RETURN StreamingChannel_getTotalPacketResend(Stream_handle Stream,
                                             int *TotalPacketResend);
SVGigE_RETURN StreamingChannel_getTotalPacketCount(Stream_handle Stream,
                                             int *TotalPacketCount);
```

Those values can be accessed only when a stream is actually running. Otherwise all return values will be zero.

Auto gain/exposure

A camera can be forced into auto gain/exposure mode by enabling that operation mode by the following function:

```
SVGigE_RETURN Camera_setAutoGainEnabled(Camera_handle Camera,
                                        bool isAutoGainEnabled);
```

Currently adjusted mode can be obtained by the following function:

```
SVGigE_RETURN Camera_getAutoGainEnabled(Camera_handle Camera,
                                        bool *isAutoGainEnabled);
```

When auto gain/exposure is enabled, a camera tries to reach a given target for image brightness automatically. First, exposure will be adjusted inside limits because this will preserve noise figures. When exposure is on maximum the gain will be used to get an image of requested brightness. Dependent on required gain the noise figures may suffer in dark environments. An application can control the limits of gain/exposure in order to meet requirements by the following functions:

```
SVGigE_RETURN Camera_setAutoGainBrightness(Camera_handle Camera,
                                           float Brightness);

SVGigE_RETURN Camera_getAutoGainBrightness(Camera_handle Camera,
                                           float *Brightness);

SVGigE_RETURN Camera_setAutoGainLimits(Camera_handle Camera,
                                       float MinGain,float MaxGain);
```

```
SVGigE_RETURN Camera_getAutoGainLimits(Camera_handle Camera,
                                       float *MinGain,
                                       float *MaxGain);

SVGigE_RETURN Camera_setAutoExposureLimits(Camera_handle Camera,
                                       float MinExposure,
                                       float MaxExposure);

SVGigE_RETURN Camera_getAutoExposureLimits(Camera_handle Camera,
                                       float *MinExposure,
                                       float *MaxExposure);
```

When auto gain/exposure is being disabled, current values for gain/exposure will continue to be active.

Decoding error messages

All SDK functions will usually return 'SVGigE_SUCCESS' in case of a successful run of a function, error codes otherwise. Those error codes can be translated into readable messages by the following function:

```
const char * getErrorMessage(SVGigE_RETURN ReturnCode)
```

Show version information about the SDK

Display the SDK's version information, which is defined at compile time.

```
SVGigE_RETURN SVGigE_getVersionInfo(SVGigE_VERSION *version);
```

IO Configuration functions

With using IO configuration functions the user can assign a signal source to signal sink.

Additionally, the user can also set the IO Mode for the selected signal sink.

```
SVGigE_RETURN Camera_getMaxIOMuxIN(Camera_handle Camera,
                                   int *MaxIOMuxINSignals);


SVGigE_RETURN Camera_getMaxIOMuxOUT(Camera_handle Camera,
                                   int *MaxIOMuxOUTSignals)


SVGigE_RETURN Camera_getIOInputStatus(Camera_handle Camera,
                                   SVGigE_IOMux_IN Source,
                                   bool * Status);


SVGigE_RETURN Camera_getIOOutputStatus(Camera_handle Camera,
                                   SVGigE_IOMux_OUT Sink,
```

```
                                        bool * Status);

SVGigE_RETURN Camera_setIOConfig(Camera_handle Camera,
                                        SVGigE_IOMux_OUT Sink,
                                        SVGigE_IO_MODE IOMode);

SVGigE_RETURN Camera_getIOConfig(Camera_handle Camera,
                                        SVGigE_IOMux_OUT Sink,
                                        SVGigE_IO_MODE *IOMode);

SVGigE_RETURN Camera_setIOAssignmentExt(Camera_handle Camera,
                                        SVGigE_IOMux_OUT Sink,
                                        SVGigE_IOMux_IN Source);
```

Example1: To assign the “IN1” (signal source) to “TRIGGER” (signal sink)

```
Camera_setIOAssignmentExt(camera, SVGigE_IOMux_OUT_TRIGGER,
SVGigE_IOMUX_IN_IN1 );
```

Example2: For additional setting the IO Mode to the selected signal sink

```
Camera_setIOConfig(camera, SVGigE_IOMux_OUT_TRIGGER, SVGigE_IO_MODE_INV);
Camera_setIOAssignmentExt(camera, SVGigE_IOMux_OUT_TRIGGER,
SVGigE_IOMUX_IN_IN1 );
```

With these two function calls the value of signal sink will be the invert value of signal source.

Sequencer

The sequencer is a special IO module which controls the camera. A new Sequencer can be loaded into the Camera with the function:

```
Camera_loadSequenceParameters(Camera_handle Camera, char * Filename);
```

After loading a Sequencer Parameter from an XML file, The Sequencer can be

started with the function

```
Camera_startSequencer(Camera_handle Camera);
```

To create an XML file use SVCapture.exe. For more information about a Sequencer see (SVCapture User Guide under Windows).

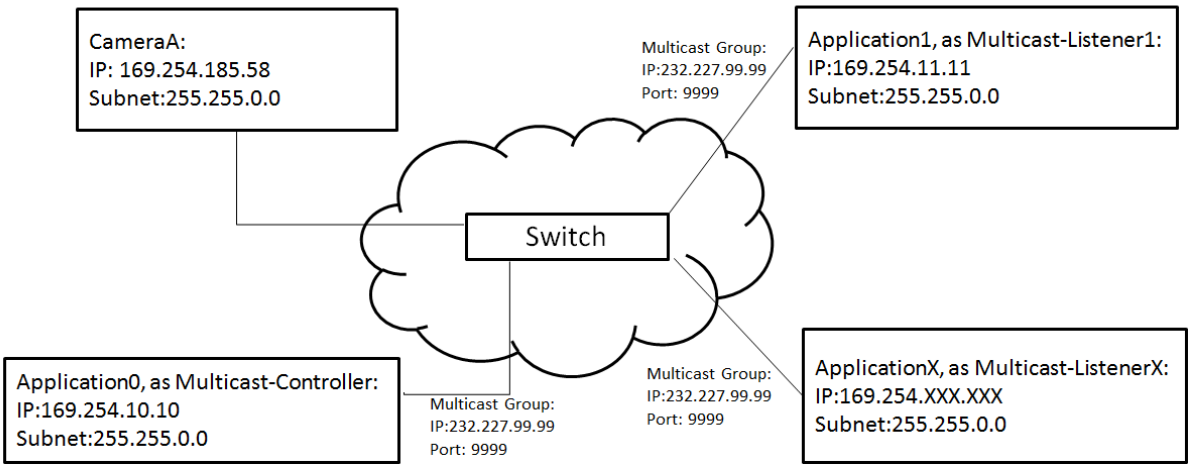
Those functions should be applied after that a stream channel is enabled (enableStream()).

When those functions are successfully applied, then the acquired image(s) will be delivered through image callback.

Distributing image streams by multicast

The image stream of a GigE camera can be distributed to multiple PCs by using the multicast capabilities of current network hardware, in particular multicast-enabled switches

A camera streams image data in a certain IP range (232.x.x.x) which has been determined for dynamic local host multicast allocation by the IANA organisation (see http://www.iana.org/assignments/multicast-addresses for more details).
A multicast-enabled switch is supposed to distribute the image stream to one, two or more applications. The advantage of having a switch distributing image streams is the optimal bandwidth usage between camera and switch as well as between switch and particular applications. There is no bandwidth wasted in the described scenario.
A multicast session consists of one controlling application and one or more listening applications. Only the controlling application has write-access to the camera. All other applications can only listen to the image stream that is initiated and controlled by the controlling application.
The connection between the controlling application and the camera is established as usual based on IP addresses. However, a switch has to be specifically instructed for forwarding an image stream also to one or more listening application. This is done by network packets which are sent out by applications regularly all 60 seconds and which detection in a switch is known as "**IGMP snooping**".
NOTE: A switch has to support "IGMP snooping" in order to establish a successful multicast session. Most often that feature is switched off in normal mode and has to be enabled by the web interface of a switch. Please refer to instructions of a switch that is supposed to work in multicast mode.



Multicast Illustration

For using camera in multicast mode with Linux SDK, first open the camera in desired mode by setting the last parameter of openCamera() / openCameraExt() either to MULTICAST_MODE_CONTROLLER or to MULTICAST_MODE_LISTENER.

After that, use Camera_setMulticastGroup() function to set the multicast group (ip) and port.

For Example: (see Multicast Illustration)

Let us assume:

- CameraA, has ip address 169.254.185.58/255.255.0.0.
- Multicast Controller - Application0 will act as multicast controller. And it uses Ethernet adapter: 169.254.10.10/255.255.0.0 to communicate with CameraA.

Then the corresponding SDK calls should looks like:

```
// set the ip, of which is used to communicate with the camera
int sourcIP = ntohl(inet_addr("169.254.10.10"));
result = openCamera(&camera, ntohl(addr.s_addr), sourcIP, 3000, MULTICAST_MODE_CONTROLLER);
if(result)
{
    printf("console: openCamera failed\n");
    return -1;
}


// set the camera's multicast group(ip) and port for multicast participant
int mcIP = ntohl(inet_addr("232.227.99.99"));
unsigned short mcPort= 9999;
Camera_setMulticastGroup(camera, mcIP, mcPort);
```

- Multicast Listener - Appliciation1 will act as multicast listener. And it uses Ethernet adapter: 169.254.11.11/255.255.0.0 to communicate with CameraA.

Then the corresponding SDK calls should looks like:

```
// set the ip, of which is used to communicate with the camera
int sourcIP = ntohl(inet_addr("169.254.11.11"));
result = openCamera(&camera, ntohl(addr.s_addr), sourcIP, 3000, MULTICAST_MODE_LISTENER);
if(result)
{
    printf("console: openCamera failed\n");
    return -1;
}


// set the camera's multicast group(ip) and port for multicast participant
int mcIP = ntohl(inet_addr("232.227.99.99"));
unsigned short mcPort= 9999;
Camera_setMulticastGroup(camera, mcIP, mcPort);
```