SVCam Kit

Getting started with SDK 2.5.2

Application Note

SVS-Vistek GmbH
Muehlbachstr. 20
82229 Seefeld
Germany

Getting started with SDK

Current SVCam Kit software version used in this document: 2.5.2.
Document revision 7, 05 October 2018.

Purpose of this document

This document is a short introduction into the SVCam-Kit SDK for windows. Focus is how to find, connect and operate a machine vision camera. This introduction does not replace studying the examples.

Targeted audience

People with a solid experience in programming C and C++ code. Users must be familiar with compiling and linking C projects. Experience in windows operating system recommended.

Content

1. Preamble

When installing the software kit from SVS-Vistek „SVCam Kit“, the user has the possibility to install the SDK as well. It contains source code files and examples. This enables the camera user to integrate the industrial camera into own applications.

The SDK provides methods to connect to a camera, modify its configurations and acquire images. Cameras from SVS-Vistek are GenICam compliant. Because of this, the GenICam Standard Features Naming Convention (SFNC, might be downloaded from www.emva.org) is a valuable source of information regarding of how to access a GenICam camera and the GenICam feature naming.

The standard SDK install location (Windows) is C:\Program Files\SVS-VISTEK GmbH\SVCam Kit\SDK. It contains libraries (dlls), includes and examples. The documents contain feature set references for GigE, Camera Link and USB3.

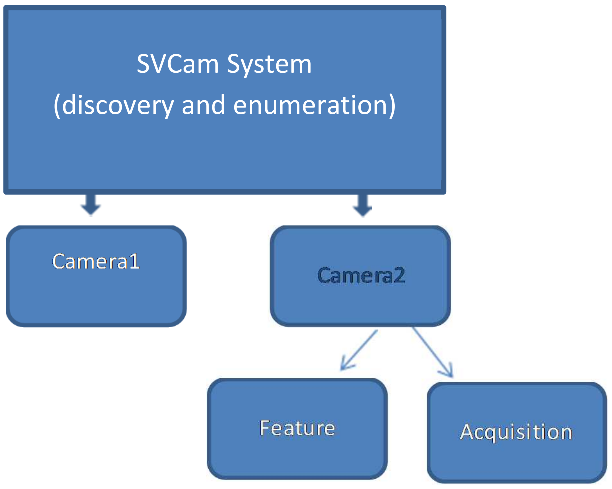
Samples

The sample folder will contain following examples:

○ SVCamMiniSample	basic cmdline sample. Find, connect, initialize and configure of camera, open and close image streaming channel
○ SVCamMiniSample64	
○ SVCamMiniSampleCSharp	
○ SVCamMiniSampleGui	MiniSample with MFC GUI elements
○ SVCamMiniSampleGui64	
○ SVCamMiniSampleIO	shows how to access and connect I/O lines
○ SVCamMiniSampleIO64	
○ SVCamSample	sample with MFC GUI elements. Shows how to parse and manipulate the GenICam tree and features
○ SVCamSample64	
○ SVCamSampleCSharp	

All samples come with ready made solutions and project files for MS Visual studio (min VS 2013). VS Studio has to be started with extended rights. With VS 2017, the project files have to be converted to VS 2017 project.

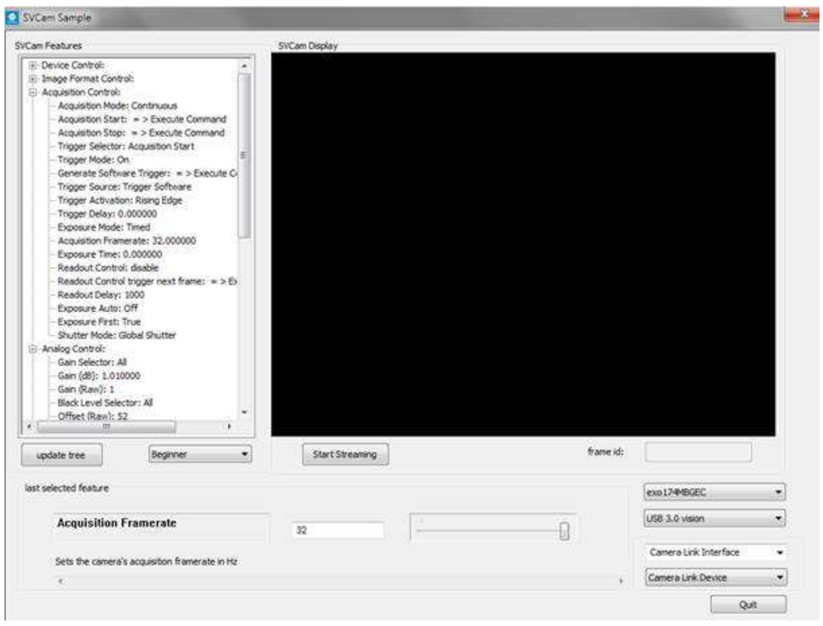
The samples show the SDK usage based on following construction:



SVCamSample

SVCamSample is much more complex compared to the SVCamMinisample.

- Based on Microsoft Foundation Classes
- Parse dynamically generated features and control camera by manipulating the GenICam feature tree
- Switch between available cameras



```
Device Control:
Image Format Control:
Acquisition Control:
    Acquisition Mode: Continuous
    Acquisition Start:  = > Execute Command
    Acquisition Stop:  = > Execute Command
    Trigger Selector: Acquisition Start
    Trigger Mode: On
    Generate Software Trigger:  = > Execute C
    Trigger Source: Trigger Software
    Trigger Activation: Rising Edge
    Trigger Delay: 0.000000
    Exposure Mode: Timed
    Acquisition Framerate: 32.000000
    Exposure Time: 0.000000
    Readout Control: disable
    Readout Control trigger next frame:  = > Ex
    Readout Delay: 1000
    Exposure Auto: Off
    Exposure First: True
    Shutter Mode: Global Shutter
Analog Control:
    Gain Selector: All
    Gain (dB): 1.010000
    Gain (Raw): 1
    Black Level Selector: All
    Offset (Raw): 52
```

Acquisition Framerate 32

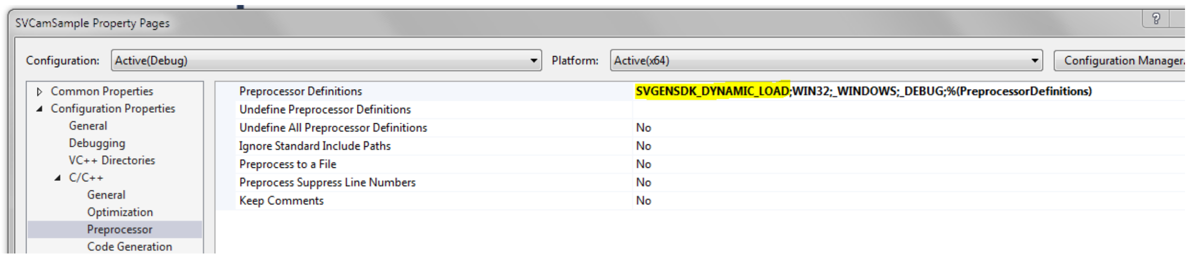
Sets the camera's acquisiton framerate in Hz

exo174MBGEC
USB 3.0 vision
Camera Link Interface
Camera Link Device

Screenshot of SVCamSample

2. Compile / link options

SVS-Vistek is recommending dynamic linking. In the solution properties, make sure dynamic linking is enabled.



SVCamSample Property Pages

Configuration: Active(Debug)   Platform: Active(x64)   Configuration Manager.

Preprocessor Definitions	SVGENSDK_DYNAMIC_LOAD;WIN32;_WINDOWS;_DEBUG;%(PreprocessorDefinitions)
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	No
Keep Comments	No

In sv_gen_sdk.h the preprocessor SVGENSDK_DYNAMIC_LOAD is enabling dynamic linking (line 45 ff). Recommended option.

```
#   ifdef SVGENSDK_DYNAMIC_LOAD
#      define SV_GEN_SDK_API
#   else
#      define SV_GEN_SDK_API __declspec(dllimport)
#   endif
```

3. Initialize SDK

The function SVLibInit() has to be called once in the application for global initialization of SVGenSDK. It will check for correct paths (GenTL components, dlls) and initialization.

```
bool InitSDK()
{
    string ctiPath;
    string genicamPath;
    string genicamCachePath;
    string clProtocolPath;

    char buffer[1024] = { 0 };
    int res = GetEnvironmentVariableA("GENICAM_GENTL32_PATH", buffer, sizeof(buffer));
    if (0 == res)
        return false;

    ctiPath = string(buffer);

    memset(buffer, 0, sizeof(buffer));
    res = GetEnvironmentVariableA("SVS_GENICAM_ROOT", buffer, sizeof(buffer));
    if (0 == res)
        return false;

    genicamPath = string(buffer);

    memset(buffer, 0, sizeof(buffer));
    res = GetEnvironmentVariableA("SVS_GENICAM_CACHE", buffer, sizeof(buffer));
    if (0 == res)
        return false;

    genicamCachePath = string(buffer);

    memset(buffer, 0, sizeof(buffer));
    res = GetEnvironmentVariableA("SVS_GENICAM_CLPROTOCOL", buffer, sizeof(buffer));
    if (0 == res)
        return false;

    clProtocolPath = string(buffer);

    SV_RETURN ret = SVLibInit(ctiPath.c_str(), genicamPath.c_str(), genicamCachePath.c_str(),
        clProtocolPath.c_str());
    if (SV_ERROR_SUCCESS != ret)
    {
        printf("SVLibInit Failed! :%d", ret);
        return false;
    }

    return true;
}
```
(taken from …SDK\sample\SVCamMiniSample\SVCamMiniSample.cpp)

After stopping the acquisition free all resources:

- SVDeviceClose()
- SVInterfaceClose()
- SVSystemClose()
- SVLibClose()

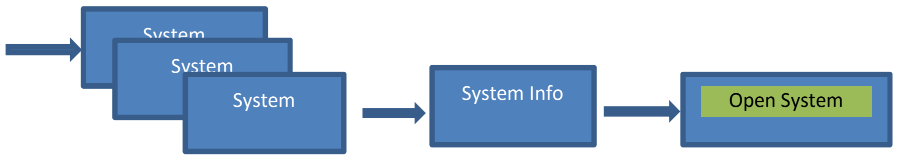
4. Device discovery and enumeration

If you want to access a camera, you need to find the camera you want to access in the first place. For being able to run multiple camera systems (interface types, device ids) on a single host system and interface lists have to be looked at.

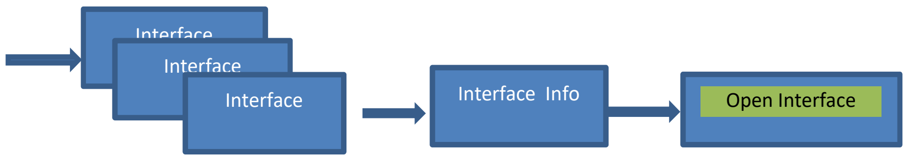
See SVCamMiniSample.cpp line 140 ff.

Every SVLibSystem has to be opened with a specific Transport Layer.

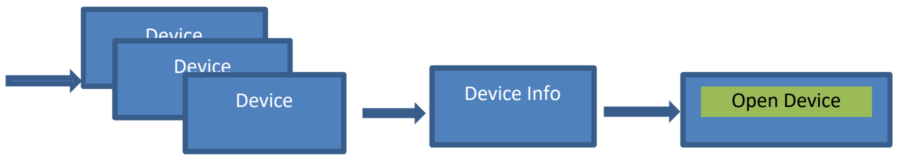
1) SVLibSystemGetCount()
2) SVLibSystemGetInfo()
3) SVLibSystemOpen()



4) SVSystemUpdateInterfaceList()
5) SVSystemGetNumInterfaces()
6) SVSystemGetInterfaceId()
7) SVSystemInterfaceGetInfo()
8) SVSystemInterfaceOpen()



9) SVInterfaceUpdateDeviceList()
10) SVInterfaceGetNumDevices()
11) SVInterfaceGetDeviceId()
12) SVInterfaceDeviceGetInfo()
13) SVInterfaceDeviceOpen()



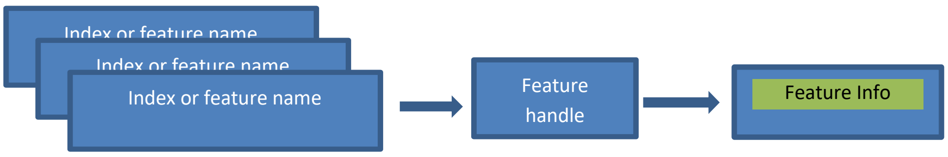
5. Getting camera features

The feature handles can be retrieved by name or by index.

SVFeatureGetByIndex() or SVFeatureGetByName()

To get the list of all currently available features go through the whole indexes until no more data is available. For each feature handle get the feature info (SV_FEATURE_INFO).

1) SVFeatureGetByIndex()
2) SVFeatureGetInfo()



To get the current value of the feature go through all features and for each type get the feature value. For each feature type the corresponding SDK function has to be used.

SV_intfIInteger:	SVFeatureGetValueInt64(), SVFeatureSetValueInt64()
SV_intfIFloat:	SVFeatureGetValueFloat(), SVFeatureSetValueFloat()
SV_intfIBoolean:	SVFeatureGetValueBool(), SVFeatureSetValueBool()
SV_intfICommand:	SVFeatureCommandExecute()
SV_intfIString:	SVFeatureGetValueString()
SV_intfIEnumeration:	SVFeatureGetValueEnum(), SVFeatureSetValueEnum()

A callback can be registered / unregistered.
To be called as soon as the feature is invalidated.

1) SVFeatureGetByName()
2) SVFeatureRegisterInvalidateCB() / SVFeatureUnRegisterInvalidateCB()

The feature info structure below is used to read all feature related information.

```
typedef struct _SV_FEATURE_INFO
{
  uint8_t type;
  char name[SV_STRING_SIZE];
  char node[SV_STRING_SIZE];
  char displayName[SV_STRING_SIZE];
  char toolTip[SV_STRING_SIZE];
  uint8_t level;
  uint8_t visibility;
  uint8_t isImplemented;
  uint8_t isAvailable;
  uint8_t isLocked;

  // feature specific
  int64_t intMin;
  int64_t intMax;
  int64_t intInc;

  double floatMin;
  double floatMax;
  double floatInc;

  uint8_t representation;
  uint8_t displayNotation;
  int64_t displayPrecision;

  int64_t strmaxLength;
  int32_t enumSelectedIndex;
  int64_t enumCount;

  int64_t pollingTime;
  char unit[SV_STRING_SIZE];
} SV_FEATURE_INFO, *PSV_FEATURE_INFO;
```

(taken from …SDK\include\sv_gen_sdk.h)

Example of getting the value of the selected feature

```
void SVCamFeature::getFeatureValue(SV_FEATURE_HANDLE hFeature,
  SVCamFeaturInf *SvCamfeatureInfo)
{
  SVFeatureGetInfo(hRemoteDevice, hFeature, &SvCamfeatureInfo->SVFeaturInf);
  memset(SvCamfeatureInfo->strValue, 0, sizeof(char)*SV_STRING_SIZE);
  memset(SvCamfeatureInfo->subFeatureName, 0, sizeof(char)*SV_STRING_SIZE);

  if(SV_intfIInteger == SvCamfeatureInfo->SVFeaturInf.type) {
    int64_t value = 0;
    SVFeatureGetValueInt64(hRemoteDevice, hFeature, &value);
    SvCamfeatureInfo->intValue = value;
    string st = to_string( value);
    st._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE, SV_STRING_SIZE,0);
  }
  else if(SV_intfIFloat == SvCamfeatureInfo->SVFeaturInf.type) {
    double value = 0.0f;
    SVFeatureGetValueFloat(hRemoteDevice, hFeature, &value);

    SvCamfeatureInfo->doubleValue = value;
    to_string( value)._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE,
      SV_STRING_SIZE,0);
  }
  else if(SV_intfIBoolean == SvCamfeatureInfo->SVFeaturInf.type) {
    bool value = false;
    SVFeatureGetValueBool(hRemoteDevice, hFeature, &value);
    SvCamfeatureInfo->booValue = value;
    if ( value ==0) {
      string s("False");
      s._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE, SV_STRING_SIZE,0);
    }
    else {
      string s("True");
      s._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE, SV_STRING_SIZE,0);
    }
  }
  else if(SV_intfICommand == SvCamfeatureInfo->SVFeaturInf.type)
  {
    string s(" = > Execute Command");
    s._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE, SV_STRING_SIZE,0);
  }
  else if(SV_intfIString == SvCamfeatureInfo->SVFeaturInf.type)
  {  SVFeatureGetValueString(hRemoteDevice, hFeature, SvCamfeatureInfo->strValue,
        SV_STRING_SIZE);
  }
  else if(SV_intfIEnumeration ==SvCamfeatureInfo->SVFeaturInf.type)
    { int64_t pInt64Value =0;
      SV_RETURN  ret = SVFeatureEnumSubFeatures(hRemoteDevice, hFeature,
        SvCamfeatureInfo->SVFeaturInf.enumSelectedIndex,
        SvCamfeatureInfo->subFeatureName,sizeof(SvCamfeatureInfo->subFeatureName));
      //, &pInt64Value);
      string s(SvCamfeatureInfo->subFeatureName);
      if (sizeof(s) > 0)
        s._Copy_s(SvCamfeatureInfo->strValue,sizeof(char)*SV_STRING_SIZE, SV_STRING_SIZE,0);
      SvCamfeatureInfo->intValue = pInt64Value;
    }
}
```

(taken from …SDK\sample\SVCamMiniSample\SVCamFeature.cpp)

6. Image acquisition

6.1. Open streaming channel

1) SVDeviceGetStreamId()
2) SVDeviceStreamOpen()

6.2. Start acquisition

1) Stream related features must be locked

```
// Some of features is locked and are not enabled on the feature tree during Streaming
// (Ex: AOI, binning, Flipping..)
SVFeatureGetByName(hRemoteDev, "TLParamsLocked", &hFeature);
SVFeatureSetValueInt64(hRemoteDev,hFeature, 1);
```

2) Retrieve the payload size to allocate the buffers

```
// retrieve the payload size to allocate the buffers
SVFeatureGetByName(hRemoteDev, "PayloadSize", &hFeature);
SVFeatureGetValueInt64(hRemoteDev, hFeature, &payloadSize);
```

3) Allocate buffers with the retrieved payload size.
4) Allocate memory to the Data Stream associated with the stream handle.
5) Queue a particular buffer for acquisition.

```
// allocate buffers with the retrieved payload size.
for(uint32_t i=0; i<bufcount; i++)
{
   size_t*buffer = new size_t[(size_t)payloadSize];
   memset(buffer, 0, (size_t)payloadSize);

   SV_BUFFER_HANDLE hBuffer = NULL;

   // allocate memory to the Data Stream associated with the hStream
   ret = SVStreamAnnounceBuffer(hDS, buffer, (uint32_t)payloadSize, NULL, &hBuffer);
   if(SV_ERROR_SUCCESS != ret)
   {
      printf(":%s SVStreamAnnounceBuffer[%d] Failed!:%d\n", __FUNCTION__, i, ret);
      delete [] buffer;
      continue;
   }
   SVStreamQueueBuffer(hDS, hBuffer);
}
```

6) Start the acquisition on the host

```
SVStreamFlushQueue(hDS, SV_ACQ_QUEUE_ALL_TO_INPUT);
ret = SVStreamAcquisitionStart(hDS, SV_ACQ_START_FLAGS_DEFAULT, INFINITE);
```

7) Start the acquisition on the remote device

```
SVFeatureGetByName(hRemoteDev, "AcquisitionStart", &hFeature);
ret =  SVFeatureCommandExecute(hRemoteDev, hFeature, ExecuteTimeout);
```

6.3. Acquire images

The (threaded) function below informs the windows application with an event as soon a new image is in the buffer.

1) SVStreamWaitForNewBuffer()
2) SVStreamBufferGetInfo()
3) SVStreamQueueBuffer

```
while(!acqTerminated)
{
   SV_BUFFER_HANDLE hBuffer = NULL;
   SV_RETURN ret = SVStreamWaitForNewBuffer(hDS, NULL, &hBuffer, 1000);
   if(SV_ERROR_SUCCESS == ret)
   {
      SV_BUFFER_INFO *bufferInfo = new SV_BUFFER_INFO();
      ret = SVStreamBufferGetInfo(hDS, hBuffer, bufferInfo);

      if(SV_ERROR_SUCCESS != ret)
      {
         delete bufferInfo;
         continue;
      }
      // Queues a particular buffer for acquisition.
      SVStreamQueueBuffer(hDS, hBuffer);

      if(imageBufferInfo.size() == 0)
      {
         imageBufferInfo.push_back(bufferInfo);
         // inform Windows that a new image has arrived
         SetEvent(m_newImage);
      }
      else
      {
         delete bufferInfo;
      }
   }
   else if(SV_ERROR_TIMEOUT == ret)
   {
      continue;
   }
}
```
(taken from ...SDK\sample\SVCamMiniSample\SVCamAcquisition.cpp)

6.4. Stop acquisition

1) Stop the acquisition on the remote device

```
SVFeatureGetByName(hRemoteDev, "AcquisitionStop", &hFeature);
SVFeatureCommandExecute(hRemoteDev, hFeature, ExecuteTimeout);
```

2) Stop the acquisition on the host

```
SVStreamAcquisitionStop(hDS, SV_ACQ_STOP_FLAGS_DEFAULT);
SVStreamFlushQueue(hDS, SV_ACQ_QUEUE_INPUT_TO_OUTPUT);
SVStreamFlushQueue(hDS, SV_ACQ_QUEUE_OUTPUT_DISCARD);
```

3) Removes the announced buffer from the acquisition engine

```
for(uint32_t i=0; i< dsBufcount; i++)
{
   SV_BUFFER_HANDLE hBuffer = NULL;
   uint8_t *pBuffer = NULL;
   int ret = SVStreamGetBufferId(hDS,0, &hBuffer);

   if(hBuffer)
   {
      SVStreamRevokeBuffer(hDS, hBuffer, (void**)&pBuffer, NULL);
      if(pBuffer)
         delete pBuffer;
   }
 }
```

4) Stream related features must be unlocked again

```
hFeature = NULL;
SVFeatureGetByName(hRemoteDev, "TLParamsLocked", &hFeature);
SVFeatureSetValueInt64(hRemoteDev,hFeature, 0);
```

After stopping the acquisition free all resources:

- SVDeviceClose()
- SVInterfaceClose()
- SVSystemClose()
- SVLibClose()

7. Special controls

(Examples from SVCamMiniSampleIO)

7.1. I/O configuration

In the sample “SVCamMiniSampleIO” you can easily select the entries and set IO connections:

```
SV_RETURN enableFeature(SV_REMOTE_DEVICE_HANDLE hRemoteDev, char *  FeatureName, bool value)
{
   SV_FEATURE_HANDLE hFeature = NULL;
   SVFeatureGetByName(hRemoteDev, FeatureName, &hFeature); // get the feature handle
   SVFeatureSetValueBool(hRemoteDev, hFeature, value);
   return 0;
}

SV_RETURN selectEnum(SV_REMOTE_DEVICE_HANDLE hRemoteDev, char * Enumselector, char *
enumentry)
{
   SV_FEATURE_HANDLE hFeature = NULL;
   SV_RETURN ret = SVFeatureGetByName(hRemoteDev, Enumselector, &hFeature);
   // get the feature handle
   ret = SVFeatureSetValueEnum(hRemoteDev, hFeature, enumentry);
   char buff[512] = {};
   ret = SVFeatureGetValueEnum(hRemoteDev, hFeature, buff, 512);
   return ret;
}


// IOMUX Connection
SV_RETURN setLineConnection (SV_REMOTE_DEVICE_HANDLE hRemoteDev, char* line_selector, char*
line_source, bool isOff)
{
   SV_RETURN ret = selectEnum(hRemoteDev, LineSelector, line_selector);
   ret = selectEnum(hRemoteDev, LineSource, line_source);
   ret = enableFeature(hRemoteDev, LineInverter, isOff);
   return 0;
}
```

7.2. Starting the sequencer

The sample (SVCamMiniSampleIO/SVCamMiniSample.cpp) contains an example application using the sequencer with four sequences using different exposure times.

Here are the exemplary steps to run the sequencer with the corresponding actions in the mini sample:

1) Trigger Mode ➔ ON
2) Trigger Source ➔ Line 1 / Hardware Trigger IO (IO Mux as Trigger Source)
3) Exposure Mode ➔ Trigger Width (Duration of the signal = Exposure)

Example

```
   // selectors
   selectEnum(cam->sv_cam_acq->hRemoteDev, TriggerMode, TriggerOn);
   selectEnum(cam->sv_cam_acq->hRemoteDev, TriggerSource, LineSelector_Output1);
   selectEnum(cam->sv_cam_acq->hRemoteDev, ExposureMode, TriggerWidth);
```

4) PWMEnable ➔ ON
5) SeqEnable ➔ ON

Example

```
   enableFeature(cam->sv_cam_acq->hRemoteDev, PWMEnable, true);
   enableFeature(cam->sv_cam_acq->hRemoteDev, SeqEnable, true);
```

6) Line Selector ➔ Output 1 – Line 1
   Line Source ➔ PWM 1 (Output 1 is connected to PWM1 (PWMChange1))

7) Line Selector ➔ Output 2 – Line 2
   Line Source ➔ PWM 2 (Output 2 is connected to PWM2 (PWMChange2))

8) Line Selector ➔ Trigger – Line 6
   Line Source ➔ SeqPulseA (SeqPulseA is connected to the exposure signal of the camera. This means that the duration of SeqPulseA sets the duration of the exposure

```
// IO Connection
// combine I/O signals
setLineConnection(cam->sv_cam_acq->hRemoteDev, LineSelector_Output1, LineSource_PWM0, false);
setLineConnection(cam->sv_cam_acq->hRemoteDev, lineSelector_Output2, LineSource_PWM1, false);
setLineConnection(cam->sv_cam_acq->hRemoteDev, lineSelector_Trigger,
                  LineSource_SeqPulseA, false);
```

9) Set sequence count to 4 (4 Sequences)
10) Set the interval configuration.

7.3. Sequencer example

Define sequences

SeqSelector	0	settings for interval 0
• SeqInterval	6666666	interval duration of 100ms (6666666 x 15ns = 100ms)
• SeqPulseAStart	0	exposure starts without delay
• SeqPulseAStop	666666	exposure stops 10ms after start of interval
SeqSelector	1	settings for interval 1
• SeqInterval	6666666	interval duration of 100ms (6666666 x 15ns = 100ms)
• SeqPulseAStart	0	exposure starts without delay
• SeqPulseAStop	333333	exposure stops 5ms after start of interval
SeqSelector	2	settings for interval 2
• SeqInterval	6666666	interval duration of 100ms (6666666 x 15ns = 100ms)
• SeqPulseAStart	0	exposure starts without delay
• SeqPulseAStop	1333333	exposure stops 20ms after start of interval
SeqSelector	3	settings for interval 3
• SeqInterval	6666666	interval duration of 100ms (6666666 x 15ns = 100ms)
• SeqPulseAStart	666666	exposure starts with 10ms delay
• SeqPulseAStop	1333333	exposure stops 20ms after start of interval
		exp time is SeqPulseAStop – SeqPulseAStart = 10ms

Additional sequence settings

SeqSelector	0	modify settings for interval 0
• SeqPulseBStart	0	PWM mask start
• SeqPulseBStop	666666	PWM mask stop output
• PWMMax	66666	PWM frequency 1000Hz (15ns x 66666 =1ms or 1000Hz)
• PWMChange1	33333	50% of PWMMax duty cycle
SeqSelector	1	modify settings for interval 1
• SeqPulseBStart	0	PWM mask start
• SeqPulseBStop	333333	PWM mask stop output
• PWMMax	6666	PWM frequency 10000Hz (15ns x 6666=0.1ms or 10kHz)
• PWMChange2	1333	20% of PWMMax duty cycle (20% ON, 80% OFF)
SeqSelector	2	modify settings for interval 2
• SeqPulseBStart	0	PWM mask start
• SeqPulseBStop	1333333	PWM mask stop output
• PWMMax	66666	PWM frequency 1000Hz (15ns x 6666=0.1ms or 10kHz)
• PWMChange1	33333	50% of PWMMax duty cycle (50% ON, 50% OFF)
SeqSelector	3	modify settings for interval 3
• SeqPulseBStart	666666	PWM mask start
• SeqPulseBStop	1333333	PWM mask stop output
• PWMMax	6666	PWM frequency 1000Hz (15ns x 6666=0.1ms or 10kHz)
• PWMChange1	3333	50% of PWMMax duty cycle (50% ON, 50% OFF)
• PWMChange2	6666	100% of PWMMax duty cycle (100% ON)

More info on how to calculate the sequence values can be found in the camera manuals/sequencer.

The example above translates into this code

```
int SeqInterval_value[]      = { 6666666, 6666666, 6666666, 6666666 };
int seqPulseAStart_value[]    = { 0, 0, 0, 666666 };
int seqpulseAStop_value[]    = { 666666, 333333, 1333333, 1333333 };
int SeqPulseBStart_value[]   = { 0, 0, 0, 666666 };
int SeqPulseBStop_Value[]    = { 666666, 333333, 1333333, 1333333 };
int PWMMax_Value[]           = { 6666, 6666, 6666, 6666 };
int PWMChange0_value[]       = { 33333, 0, 0, 0 };
int PWMChange1_value[]       = { 0, 0, 1333, 0 };
int PWMChange2_value[]       = { 0, 3333, 0, 0 };
int PWMChange3_value[]       = { 0, 3333, 0, 0 };



int seq_count = 4;
setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqCount, seq_count);
for (int j = 0; j < seq_count; j++)
{
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqSelector, j);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqInterval, SeqInterval_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqPulseAStart, seqPulseAStart_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqPulseAStop, seqpulseAStop_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqPulseBStart, SeqPulseBStart_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, SeqPulseBStop, SeqPulseBStop_Value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, PWMMax, PWMMax_Value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, PWMChange0, PWMChange0_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, PWMChange1, PWMChange1_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, PWMChange2, PWMChange2_value[j]);
   setFeatureValueInt(cam->sv_cam_acq->hRemoteDev, PWMChange3, PWMChange3_value[j]);
}
```