Machine Vision Camera SDK (C)

Developer Guide

Legal Information

© 2022 Hangzhou Hikrobot Technology Co., Ltd. All rights reserved.

This Document (hereinafter referred to be "the Document") is the property of Hangzhou Hikrobot Digital Technology Co., Ltd. or its affiliates (hereinafter referred to as "Hikrobot"), and it cannot be reproduced, changed, translated, or distributed, partially or wholly, by any means, without the prior written permission of Hikrobot. Unless otherwise expressly stated herein, Hikrobot does not make any warranties, guarantees or representations, express or implied, regarding to the Document, any information contained herein.

LEGAL DISCLAIMER

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE DOCUMENT IS PROVIDED "AS IS" AND "WITH ALL FAULTS AND ERRORS". HIKROBOT MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. IN NO EVENT WILL HIKROBOT BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES, INCLUDING, AMONG OTHERS, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION OR LOSS OF DATA, CORRUPTION OF SYSTEMS, OR LOSS OF DOCUMENTATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, IN CONNECTION WITH THE USE OF THE DOCUMENT, EVEN IF HIKROBOT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR LOSS.
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE DOCUMENT IS PROVIDED "AS IS" AND "WITH ALL FAULTS AND ERRORS". OUR COMPANY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. IN NO EVENT WILL OUR COMPANY BE LIABLE FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES, INCLUDING, AMONG OTHERS, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION OR LOSS OF DATA, CORRUPTION OF SYSTEMS, OR LOSS OF DOCUMENTATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, IN CONNECTION WITH THE USE OF THE DOCUMENT, EVEN IF OUR COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR LOSS.

Contents

Chapter 1 Overview

Machine vision camera SDK (MvCameraSDK) contains API definitions, example, and camera driver. It is compatible with standard protocols, and currently, CoaXPress, GigE Vision, USB3 Vision, and Camera Link protocols are supported.

1.1 Introduction

This manual mainly introduces the MvCameraSDK based on C language, which provides several APIs to implement the functions of image acquisition, parameter configuration, image post-process, device upgrade, and so on.

Parameter configuration and image acquisition are two basic functions, see details below:

- Parameter configuration: Get and set all parameters of cameras, such as image width, height, exposure time, which are realized by the general configuration API.
- Image acquisition: When the camera sends image data to PC, the image data will be saved to the SDK. SDK provides two methods for getting the image, including search method and callback method. These two methods cannot be adopted at same time, the user should choose one method according to actual application.

Remarks

- The driver program can be selected to be installed during installing Machine Vision Software (MVS), or be installed directly via the executable program "Driver_Installation_Tool.exe". You can get it in the toolbar of start menu.
- The API for Windows operating system are provided in dynamic linking library (DLL), and the default directory is: \Program Files (x86)\Common Files\MVS\Runtime. And the directory will be added by default to PATH environment variable after installing the Machine Vision System.

1.2 Development Environment

The development environment of MvCameraSDK is shown in the table below.

Operating System

Item	Required
Operating System	Microsoft® Windows XP (32-bit)/Windows 7 (32/64-bit)/Windows 10 (32/64-bit)/Windows Server Supports driver

Development Folder Contents

By default, the Machine Vision Software (MVS) is installed by default in the path of C:\Program Files (x86)\MVS. After installation, folder MVS contains the folder Development, of which the contents are as below:

Content Name	Description
Documentations	Programming documents
Includes	Header files
Libraries	lib files
Samples	Sample codes

Note

In the path of C:\Program Files (or Program Files (x86))\Common Files\MVS, there are three folders: Drivers, Runtime (32-bit/64-bit dynamic linking library), and Service (camera log service).

Prerequisites

Install the Machine Vision Software (MVS) to get the development kit (including programming manuals, head files, library files, and demos) and prepare environment for development. After installing the MVS client, and before starting the secondary development based on SDK, add the camera and check the connection and live view of the camera.

Note

- The default installation path for the MVS client is C:\Program Files (x86)\MVS and the development kit is in this installation directory.
- Multiple demos developed based on different programming languages or functions are provided for reference, including BasicDemo, VC60 demo, VS demo, VB demo, C# demo, LabView demo, Halcon demo, and DirectShow demo. See details in the user manual of corresponding demos.
- The checklist for GigE camera contains frame rate (whether same to actual frame rate), number of errors (non-0: frame is lost, exception), number of lost packets (non-0: exception), while the checklist for USB3Vision camera only contains frame rate (whether same to actual frame rate).

1.3 Update History

The update history shows the summary of changes in MvCameraSDK with different versions.

Summary of Changes in Version 4.0_April/2022

Version	Content
Version 4.0_April/2022	1. Added the Runtime and programming package for frame grabber SDK.
	2. Supported Windows Server operating system.
	3. Added API for enumerating devices according to the specified sorting type: ***MV_CC_EnumDevicesEx2*** .
	4. Added API for getting the number of valid images in current image cache: ***MV_CC_GetValidImageNum*** .
	5. Added API for getting the enumerator name according to the node name and assigned value: ***MV_CC_GetEnumEntrySymbolic*** .
	6. Added API for opening the Graphical User Interface (GUI) of camera parameters configurations: ***MV_CC_OpenParamsGUI*** .
	7. Added API for unloading the CTI library: ***MV_CC_UnloadGenTLLibrary*** .
	8. Added API for enabling or disabling the smoothing function of interpolation algorithm: ***MV_CC_SetBayerFilterEnable*** .
	9. Added API for adjusting the image contrast: ***MV_CC_ImageContrast*** .
	10. Added APIs for drawing auxiliary rectangle frames/circle frames/ lines on the image: ***MV_CC_DrawRect*** / ***MV_CC_DrawCircle*** / ***MV_CC_DrawLines*** .
	11. Updated the algorithm of image contrast adjustment.
	12. Added API for reconstructing the image: ***MV_CC_ReconstructImage*** .
	13. Added API for registering the callback function for receiving the stream exceptions: ***MV_USB_RegisterStreamExceptionCallBack*** .
	14. Added API for setting the number of event cache nodes for USB3.0 cameras: ***MV_USB_SetEventNodeNum*** .
	15. Added the sample code OpenParamsGUI.cpp, see ***Open GUI of Camera Property Settings*** for details.

Summary of Changes in Version 3.5.0_Jan./2021

Version	Content
Version 3.5.0_Jan./2021	1. Implemented the image processing via ISP Tool. ISP Tool calls the SDK to generate different calibration files and configuration parameters of algorithm, the SDK can process the images behind image acquisition APIs via generated configuration files. Refer to ***Image Acquisition and Display*** for details. For methods of generating configuration files, see the ISP Tool user manual.
	2. Obsoleted following image processing APIs: MV_CC_ColorCorrect, MV_CC_SetBayerCLUTParam, MV_CC_NoiseEstimate, MV_CC_SpatialDenoise, MV_CC_ImageContrast, and MV_CC_ImageSharpen.
	3. Optimized the sample code ***Correct Lens Shading*** . Deleted the following sample codes: ImageEnhance, SpatialDenoise, and ColorCorrect.
	4. Supports getting the lossless stream of USB3 vision camera.
	5. Supports multithreading to optimize the ISP algorithm.

Summary of Changes in Version 3.4.0_Aug./2020

Version	Content
Version 3.4.0_Aug./2020	1. Added API for color correction: MV_CC_ColorCorrect.
	2. Added API for setting gamma parameters of Bayer pattern: ***MV_CC_SetBayerGammaParam*** .
	3. Added API for enabling/disabling CCM and setting CCM parameters of Bayer pattern: ***MV_CC_SetBayerCCMParamEx*** .
	4. Added API for enabling/disabling CLUT and setting CLUT parameters of Bayer pattern: MV_CC_SetBayerCLUTParam.
	5. Added API for LSC calibration: ***MV_CC_LSCCalib*** .
	6. Added API for LSC correction: ***MV_CC_LSCCorrect*** .
	7. Added API for adjusting image contrast: MV_CC_ImageContrast.
	8. Added API for adjusting image sharpness: MV_CC_ImageSharpen.
	9. Added API for estimating noise: MV_CC_NoiseEstimate.
	10. Added API for spatial denoising: MV_CC_SpatialDenoise.

Version	Content
	11. Added the sample code for correcting the color of the image of a camera with gamma, CCM, and CLUT: Correct Color.
	12. Added the sample code for enhancing the image of a camera by configuring contrast and sharpness: Enhance Image.
	13. Added the sample code for correcting lens shading: ***Correct Lens Shading*** .
	14. Added the sample code for spatial denoising: Spatial Denoising.

Summary of Changes in Version 3.3.0_Mar./2020

Version	Content
Version 3.3.0_Mar./2020	1. Added API for setting device ACK packet type: ***MV_GIGE_SetDiscoveryMode*** .
	2. Added APIs for setting or getting GVSP streaming timeout: ***MV_GIGE_SetGvspTimeout*** , ***MV_GIGE_GetGvspTimeout*** .
	3. Added APIs for setting or getting the maximum times one packet can be resent: ***MV_GIGE_SetResendMaxRetryTimes*** , ***MV_GIGE_GetResendMaxRetryTimes*** .
	4. Added APIs for setting or getting the packet resending interval: ***MV_GIGE_SetResendTimeInterval*** , ***MV_GIGE_GetResendTimeInterval*** .
	5. Added API for rotating pictures: ***MV_CC_RotateImage*** .
	6. Added API for flipping pictures: ***MV_CC_FlipImage*** .
	7. Added API for setting the gamma value after Bayer interpolation: ***MV_CC_SetBayerGammaValue*** .
	8. Added API for color correction after Bayer interpolation: ***MV_CC_SetBayerCCMParam*** .
	9. Added API for lossless decoding: ***MV_CC_HB_Decode*** .
	10. Added API for estimating noise based on pictures in Bayer format: MV_CC_BayerNoiseEstimate.
	11. Added API for spatial noise reduction based on pictures in Bayer format: MV_CC_BayerSpatialDenoise.

Version	Content
	12. Extended the pixel format enumeration ***MvGvspPixelType*** : added lossless decoding pixel formats.
	13. Delete the node sheet MvCameraNode.

Summary of Changes in Version 3.2.0_June/2019

Version	Content
Version 3.2.0_June/2019	1. Added API for getting multicast status: ***MV_GIGE_GetMulticastStatus*** .
	2. Added API for saving the 3D point cloud data: ***MV_CC_SavePointCloudData*** .
	3. Added API for saving image to file: ***MV_CC_SaveImageToFile*** .
	4. Added API for enumerating interfaces via GenTL: ***MV_CC_EnumInterfacesByGenTL*** .
	5. Added API for enumerating devices via GenTL: ***MV_CC_EnumDevicesByGenTL*** .
	6. Added API for creating a device handle via GenTL device information: ***MV_CC_CreateHandleByGenTL*** .
	7. Deleted the obsolete APIs.

Summary of Changes in Version 3.1.0_May/2019

Version	Content
Version 3.1.0_May/2019	1. Added API for setting streaming strategy: ***MV_CC_SetGrabStrategy*** .
	2. Added API for setting the output queue size: ***MV_CC_SetOutputQueueSize*** .
	3. Added API for getting the current node type: ***MV_XML_GetNodeInterfaceType*** .
	4. Added API for getting current node access mode: ***MV_XML_GetNodeAccessMode*** .
	5. Added API for setting the GVCP command retransmission times: ***MV_GIGE_SetRetryGvcpTimes*** .

Version	Content
	6. Added API for getting the number of GVCP retransmission commands: ***MV_GIGE_GetRetryGvcpTimes*** .
	7. Added API for getting the GVCP command timeout: ***MV_GIGE_GetGvcpTimeout*** .
	8. Added API for clearing streaming data buffer: ***MV_CC_ClearImageBuffer*** .
	9. Added API for setting the packet size of USB3 vision device: ***MV_USB_SetTransferSize*** .
	10. Added API for getting the packet size of USB3 vision device: ***MV_USB_GetTransferSize*** .
	11. Added API for setting the number of transmission channels for USB3 vision device: ***MV_USB_SetTransferWays*** .
	12. Added API for getting the number of transmission channels for USB3 vision device: ***MV_USB_GetTransferWays*** .

Summary of Changes in Version 3.0.0_April/2019

New document.

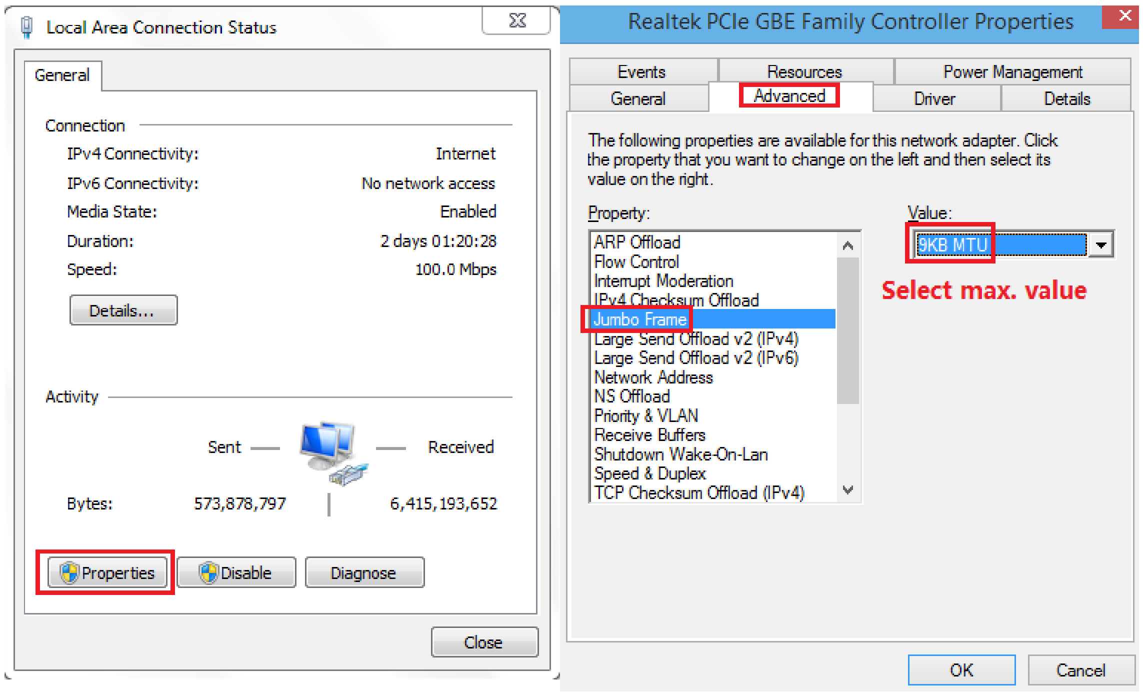
1.4 Notice

Install Camera Driver

Before using SDK for connection and development of machine vision camera, make sure appropriate camera driver is installed. If not, disable Windows Firewall when the PC is streaming.

NIC

It is recommended to use Intel series 1000M NIC, and go to Local Area Connection Status to enable Jumbo Frame function, as shown below:


Local Area Connection Status

General

Connection

IPv4 Connectivity:	Internet
IPv6 Connectivity:	No network access
Media State:	Enabled
Duration:	2 days 01:20:28
Speed:	100.0 Mbps

Details...

Activity

	Sent	Received
Bytes:	573,878,797	6,415,193,652

Properties Disable Diagnose

Close

Realtek PCIe GBE Family Controller Properties

Events | Resources | Power Management | General | Advanced | Driver | Details

The following properties are available for this network adapter. Click the property that you want to change on the left and then select its value on the right.

Property:
- ARP Offload
- Flow Control
- Interrupt Moderation
- IPv4 Checksum Offload
- Jumbo Frame
- Large Send Offload v2 (IPv4)
- Large Send Offload v2 (IPv6)
- Network Address
- NS Offload
- Priority & VLAN
- Receive Buffers
- Shutdown Wake-On-Lan
- Speed & Duplex
- TCP Checksum Offload (IPv4)

Value: 9KB MTU

Select max. value

OK Cancel

Chapter 2 Connect Device

Before operating the device to implement the functions of image acquisition, parameter configuration, and so on, you should connect the device (open device).

Steps

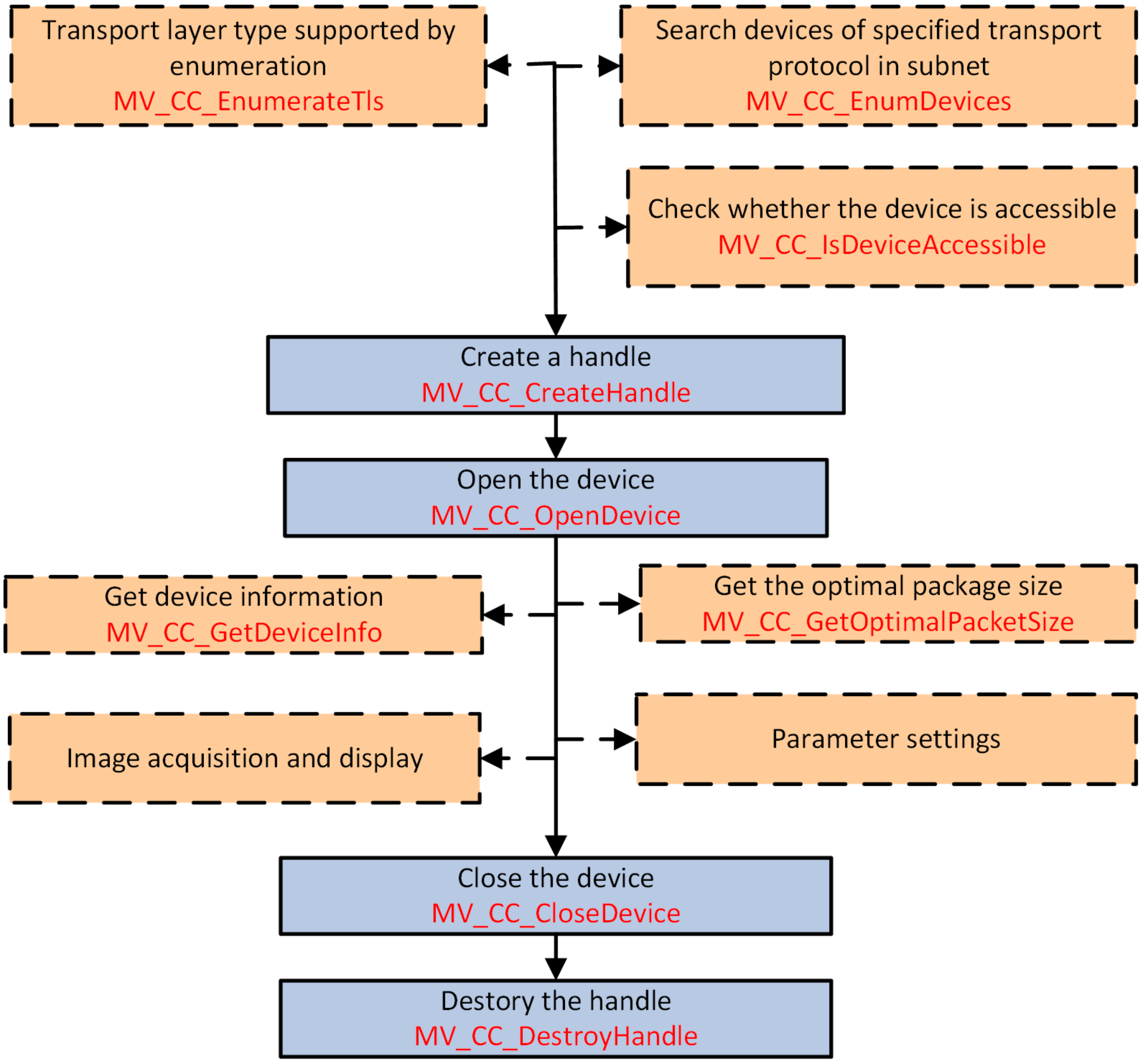


Figure 2-1 Programming Flow of Connecting Device

1. **Optional:** Call ***MV_CC_EnumDevices*** to enumerate all devices corresponding to specified transport protocol on the subnet.

   The information of found devices is returned in the structure ***MV_CC_DEVICE_INFO_LIST*** by **pstDevList**.

2. **Optional:** Call ***MV_CC_IsDeviceAccessible*** to check if the specified device is accessible before opening it.
3. Call ***MV_CC_CreateHandle*** to create a device handle.
4. Call ***MV_CC_OpenDevice*** to open the device.
5. **Optional:** Perform one or more of the following operations.

Get Device Information	Call ***MV_CC_GetDeviceInfo***
Get Optimal Package Size	Call ***MV_CC_GetOptimalPacketSize***

6. **Optional:** Other operations, such as image acquisition and display, parameters configuration, and so on. Refer to ***Image Acquisition and Display*** for details.
7. Call ***MV_CC_CloseDevice*** to close the device.
8. Call ***MV_CC_DestroyHandle*** to destroy the handle and release resources.

Chapter 3 Image Acquisition and Display

Two methods of image acquisition are provided in the MvCameraSDK. You can get the image directly after starting stream or get the image in registered callback function.

- For detailed programming flow of getting image directly, refer to ***Get Image Directly*** .
- For detailed programming flow of getting image in callback function, refer to ***Get Image in Callback Function*** .

Note

Now supports processing the images behind the image acquisition APIs via generated ISP configuration files. You should create a folder named "ISPTool" in Users folder of C disk (e.g., C:\Users\Kevin\ISPTool), and copy the calibration file (.bin) and configuration file (.xml) to the "ISPTool" folder.

3.1 Get Image Directly

You can directly get the image after starting getting stream, or adopts asynchronous mode (thread or timer) to get the image.

Steps

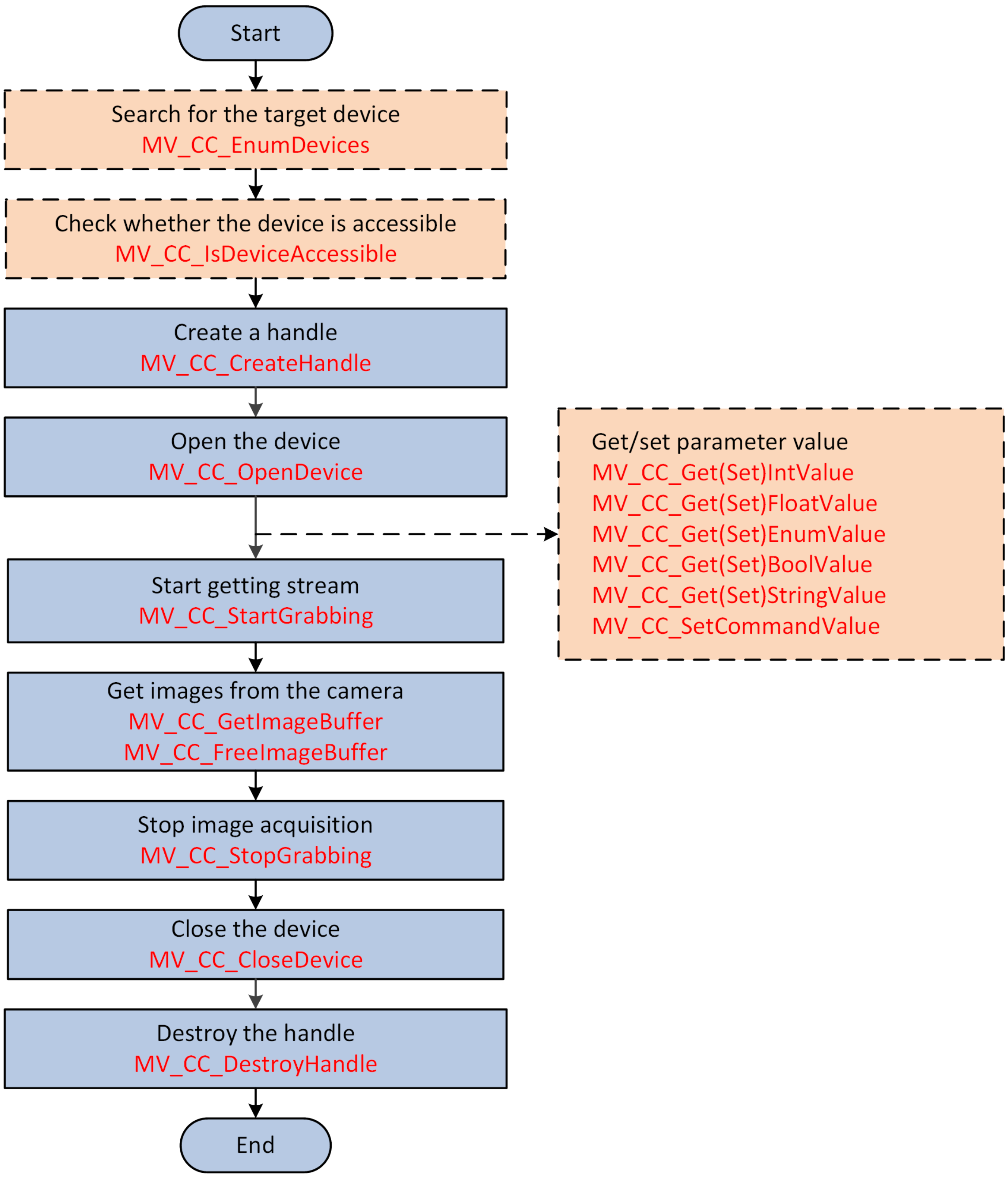


Figure 3-1 Programming Flow of Getting Image Directly

1. Call ***MV_CC_EnumDevices*** to enumerate all devices corresponding to specified transport protocol within subnet.

   The information of found devices is returned in the structure ***MV_CC_DEVICE_INFO_LIST*** by **pstDevList**.

2. **Optional:** Call ***MV_CC_IsDeviceAccessible*** to check if the specified device is accessible before opening it.
3. Call ***MV_CC_CreateHandle*** to create a device handle.
4. **Optional:** Perform one or more of the following operations to get/set different types parameters.

Get/Set Camera Bool Node Value	Call ***MV_CC_GetBoolValue*** / ***MV_CC_SetBoolValue***
Get/Set Camera Enum Node Value	Call ***MV_CC_GetEnumValue*** / ***MV_CC_SetEnumValue***
Get/Set Camera Float Node Value	Call ***MV_CC_GetFloatValue*** / ***MV_CC_SetFloatValue***
Get/Set Camera Int Node Value	Call ***MV_CC_GetIntValueEx*** / ***MV_CC_SetIntValueEx***
Get/Set Camera String Node Value	Call ***MV_CC_GetStringValue*** / ***MV_CC_SetStringValue***
Set Camera Command Node	Call ***MV_CC_SetCommandValue***

Note

- You can get and set the acquisition mode including single frame acquisition, multi-frame acquisition, and continuous acquisition via the API ***MV_CC_GetEnumValue*** (handle, "AcquisitionMode", &stEnumValue) and ***MV_CC_SetEnumValue*** (handle, "AcquisitionMode", value).
- You can set triggering parameters.
  a. Call ***MV_CC_SetEnumValue*** (handle, "TriggerMode", value) to set the triggering mode.
  b. If the triggering mode is enabled, call ***MV_CC_SetEnumValue*** (handle, "TriggerSource", value) to set the triggering resource. The triggering source includes triggered by hardware and software.
  c. Call ***MV_CC_GetFloatValue*** (handle, " TriggerDelay", &stFloatValue) and ***MV_CC_SetFloatValue*** (handle, " TriggerDelay", value) to get and set the triggering delay time.
  d. When triggered by software, call ***MV_CC_SetCommandValue*** (handle, "TriggerSoftware ") to capture; when triggered by hardware, capture by device local input.
- You can set the image parameters, including image width/height, pixel format, frame rate, AIO offset, gain, exposure mode, exposure value, brightness, sharpness, saturation, grayscale, white balance, Gamma value, and so on, by calling the following APIs: ***MV_CC_SetIntValueEx*** , ***MV_CC_SetEnumValue*** , ***MV_CC_SetFloatValue*** , ***MV_CC_SetBoolValue*** , ***MV_CC_SetStringValue*** , ***MV_CC_SetCommandValue*** .

5. Call ***MV_CC_StartGrabbing*** to start getting streams.

Note

- Before starting the acquisition, you can call ***MV_CC_SetImageNodeNum*** to set the number of image buffer nodes. When the number of obtained images is larger than this number, the earliest image data will be discarded automatically.
- For original image data, you can call ***MV_CC_ConvertPixelType*** to convert the image pixel format, or you can call ***MV_CC_SaveImageEx2*** to convert the image to JPEG or BMP format and save as a file.

6. Perform one of the following operations to acquire images.
   - Call ***MV_CC_GetOneFrameTimeout*** repeatedly in the application layer to get the frame data with specified pixel format.
   - Call ***MV_CC_GetImageBuffer*** in the application layer to get the frame data with specified pixel format and call ***MV_CC_FreeImageBuffer*** to release the buffer.

Note

- When getting the frame data, the application program should control the frequency of calling this API according to the frame rate.
- The differences of above two image acquisition methods are:
  MV_CC_GetImageBuffer should be used with ***MV_CC_FreeImageBuffer*** in pairs, the data pointer of **pstFrame** should be released by ***MV_CC_FreeImageBuffer*** .
  Compared with ***MV_CC_GetOneFrameTimeout*** , ***MV_CC_GetImageBuffer*** is more efficient, and its stream buffer is allocated by SDK, while the stream buffer of ***MV_CC_GetOneFrameTimeout*** should allocated by the developer.
- The above two methods and the method of acquiring image in callback function cannot be used at the same time.
- The **pData** returns an address pointer, it is recommended to copy the data of **pData** to create another thread.

7. **Optional:** Call ***MV_CC_DisplayOneFrame*** to input the window handle and start displaying.
8. Call ***MV_CC_StopGrabbing*** to stop the acquisition or stop displaying.
9. Call ***MV_CC_CloseDevice*** to close the device.
10. Call ***MV_CC_DestroyHandle*** to destroy the handle and release resources.

3.2 Get Image in Callback Function

The API MV_CC_RegisterImageCallBackEx is provided for registering callback function. You can customize the callback function and the obtained image will automatically called back. This method can simplify the application logic.

Steps

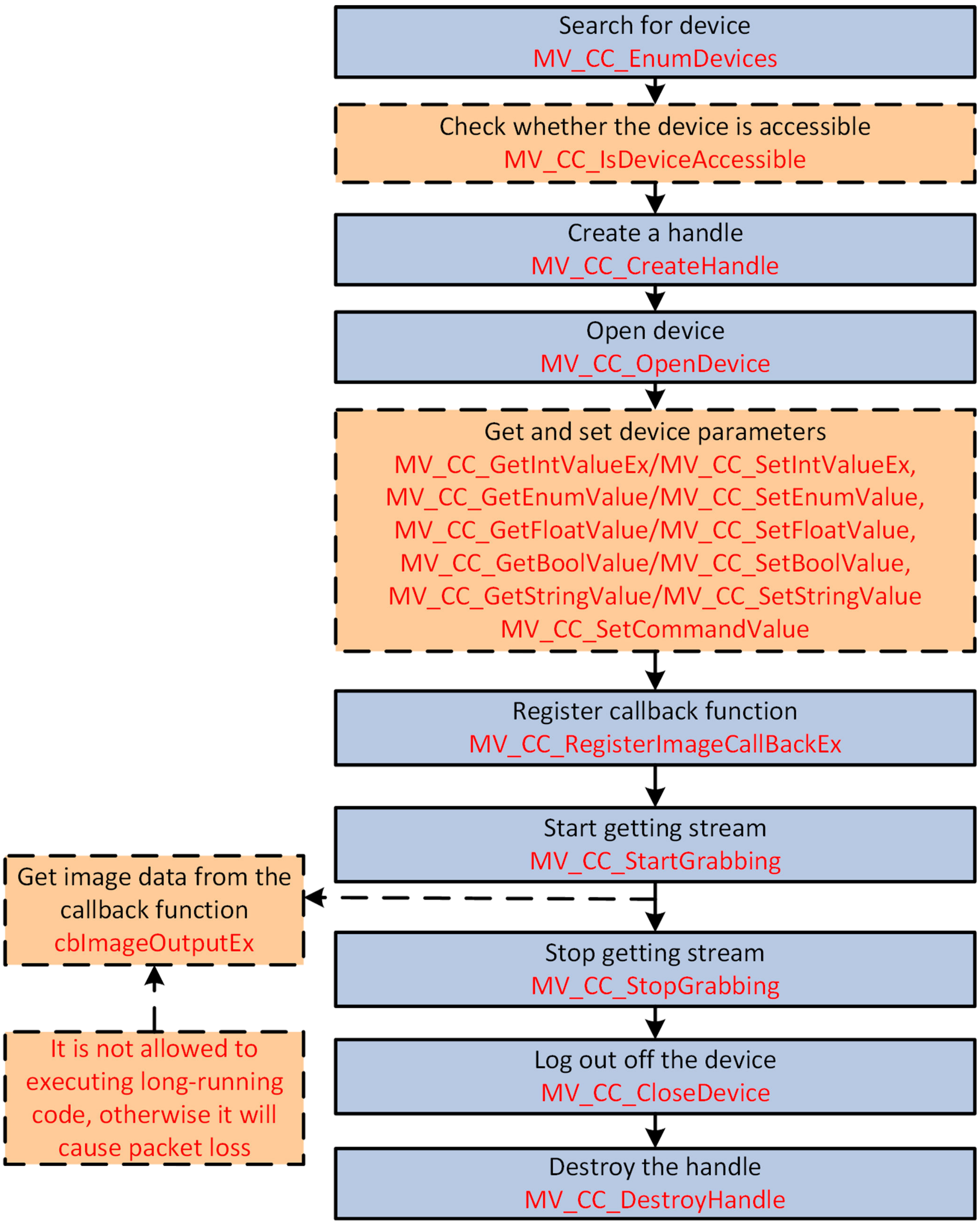

Search for device
MV_CC_EnumDevices
Check whether the device is accessible
MV_CC_IsDeviceAccessible
Create a handle
MV_CC_CreateHandle
Open device
MV_CC_OpenDevice
Get and set device parameters
MV_CC_GetIntValueEx/MV_CC_SetIntValueEx,
MV_CC_GetEnumValue/MV_CC_SetEnumValue,
MV_CC_GetFloatValue/MV_CC_SetFloatValue,
MV_CC_GetBoolValue/MV_CC_SetBoolValue,
MV_CC_GetStringValue/MV_CC_SetStringValue
MV_CC_SetCommandValue
Register callback function
MV_CC_RegisterImageCallBackEx
Start getting stream
MV_CC_StartGrabbing
Get image data from the callback function
cbImageOutputEx
It is not allowed to executing long-running code, otherwise it will cause packet loss
Stop getting stream
MV_CC_StopGrabbing
Log out off the device
MV_CC_CloseDevice
Destroy the handle
MV_CC_DestroyHandle

Figure 3-2 Programming Flow of Getting Image in Callback Function

1. Call ***MV_CC_EnumDevices*** to enumerate all devices corresponding to specified transport protocol within subnet.

   The information of found devices is returned in the structure ***MV_CC_DEVICE_INFO_LIST*** by **pstDevList**.
2. **Optional:** Call ***MV_CC_IsDeviceAccessible*** to check if the specified device is accessible before opening it.
3. Call ***MV_CC_CreateHandle*** to create a device handle.
4. **Optional:** Perform one or more of the following operations to get/set different types parameters.

Get/Set Camera Bool Node Value	Call ***MV_CC_GetBoolValue*** / ***MV_CC_SetBoolValue***
Get/Set Camera Enum Node Value	Call ***MV_CC_GetEnumValue*** / ***MV_CC_SetEnumValue***
Get/Set Camera Float Node Value	Call ***MV_CC_GetFloatValue*** / ***MV_CC_SetFloatValue***
Get/Set Camera Int Node Value	Call ***MV_CC_GetIntValueEx*** / ***MV_CC_SetIntValueEx***
Get/Set Camera String Node Value	Call ***MV_CC_GetStringValue*** / ***MV_CC_SetStringValue***
Set Camera Command Node	Call ***MV_CC_SetCommandValue***

Note

- You can get and set the acquisition mode including single frame acquisition, multi-frame acquisition, and continuous acquisition via the API ***MV_CC_GetEnumValue*** (handle, "AcquisitionMode", &stEnumValue) and ***MV_CC_SetEnumValue*** (handle, "AcquisitionMode", value).
- You can set triggering parameters.
  a. Call ***MV_CC_SetEnumValue*** (handle, "TriggerMode", value) to set the triggering mode.
  b. If the triggering mode is enabled, call ***MV_CC_SetEnumValue*** (handle, "TriggerSource", value) to set the triggering resource. The triggering source includes triggered by hardware and software.
  c. Call ***MV_CC_GetFloatValue*** (handle, " TriggerDelay", &stFloatValue) and ***MV_CC_SetFloatValue*** (handle, " TriggerDelay", value) to get and set the triggering delay time.
  d. When triggered by software, call ***MV_CC_SetCommandValue*** (handle, "TriggerSoftware ") to capture; when triggered by hardware, capture by device local input.
- You can set the image parameters, including image width/height, pixel format, frame rate, AIO offset, gain, exposure mode, exposure value, brightness, sharpness, saturation, grayscale, white balance, Gamma value, and so on, by calling the following APIs: ***MV_CC_SetIntValueEx*** , ***MV_CC_SetEnumValue*** , ***MV_CC_SetFloatValue*** , ***MV_CC_SetBoolValue*** , ***MV_CC_SetStringValue*** , ***MV_CC_SetCommandValue*** .

5. Acquire images.
   1) Call ***MV_CC_RegisterImageCallBackEx*** to set data callback function.
   2) Call ***MV_CC_StartGrabbing*** to start the acquisition.

Note

- Before starting the acquisition, you can call ***MV_CC_SetImageNodeNum*** to set the number of image buffer nodes. When the number of obtained images is larger than this number, the earliest image data will be discarded automatically.
- For original image data, you can call ***MV_CC_ConvertPixelType*** to convert the image pixel format, or you can call ***MV_CC_SaveImageEx2*** to convert the image to JPEG or BMP format and save as a file.

6. **Optional:** Call ***MV_CC_DisplayOneFrame*** to input the window handle and start displaying.
7. Call ***MV_CC_StopGrabbing*** to stop the acquisition or stop displaying.
8. Call ***MV_CC_CloseDevice*** to close the device.
9. Call ***MV_CC_DestroyHandle*** to destroy the handle and release resources.

Chapter 4 API Reference

4.1 General

4.1.1 MV_CC_GetSDKVersion

Get the SDK version No.

API Definition

```
unsigned int MV_CC_GetSDKVersion(
);
```

Return Value

Return SDK version No., the format is as follows:
|Main |Sub |Revision |Test
|8bits |8bits |8bits |8bits

Remarks

For example, if the return value is 0x01000001, the SDK version is V1.0.0.1.

4.1.2 MV_CC_EnumDevices

Enumerate all devices corresponding to specified transport protocol on the subnet.

API Definition

```
int MV_CC_EnumDevices(
  unsigned int              nTLayerType,
  MV_CC_DEVICE_INFO_LIST    *pstDevList
);
```

Parameters

nTLayerType

[IN] Transport layer protocol type, indicated by bit, supporting multiple selections, available protocol types are shown in the table below:

Macro Definition	Value	Description
MV_UNKNOW_DEVICE	0x00000000	Unknown device type
MV_GIGE_DEVICE	0x00000001	GigE device
MV_1394_DEVICE	0x00000002	1394-a/b device
MV_USB_DEVICE	0x00000004	USB3.0 device
MV_CAMERALINK_DEVICE	0x00000008	CameraLink device
MV_VIR_GIGE_DEVICE	0x00000010	Virtual GigE device
MV_VIR_USB_DEVICE	0x00000020	Virtual USB3.0 device
MV_GENTL_GIGE_DEVICE	0x00000040	Virtual CameraLink device

For example, if nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE, which means searching GigE and USB 3.0 device.

pstDevList

[OUT] Information list of found devices, see the structure ***MV_CC_DEVICE_INFO_LIST*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

The memory of storing device list is allocated by the SDK and will be released and applied for during multiple-threaded API calling. It is recommended to avoid the multiple-threaded enumeration.

4.1.3 MV_CC_EnumDevicesEx

Enumerate all the devices of specified transport protocol and manufacturer on the subnet.

API Definition

```
int MV_CC_EnumDevicesEx(
  unsigned int             nTLayerType,
  MV_CC_DEVICE_INFO_LIST   *pstDevList,
  const char               *pManufacturerName
);
```

Parameters

nTLayerType

[IN] Transport layer protocol type, indicated by bit, supporting multiple selections, available protocol types are shown in the table below:

Macro Definition	Value	Description
MV_UNKNOW_DEVICE	0x00000000	Unknown device type
MV_GIGE_DEVICE	0x00000001	GigE device
MV_1394_DEVICE	0x00000002	1394-a/b device
MV_USB_DEVICE	0x00000004	USB3.0 device
MV_CAMERALINK_DEVICE	0x00000008	CameraLink device

For example, if **nTLayerType** = MV_GIGE_DEVICE | MV_USB_DEVICE, which means searching GigE and USB 3.0 device.

pstDevList

[OUT] Device information list, see the structure ***MV_CC_DEVICE_INFO_LIST*** for details.

pManufacturerName

[IN] Manufacturer name, for example, "abc"-enumerate abc cameras.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

The memory of storing device list is allocated by the SDK and will be released and applied for during multiple-threaded API calling. It is recommended to avoid the multiple-threaded enumeration.

4.1.4 MV_CC_EnumDevicesEx2

Enumerate devices according to the specified sorting type. You can set the sorting type for enumerated devices and filter them by the manufacturer name.

API Definition

```
int MV_CC_EnumDevicesEx2(
  unsigned int                nTLayerType,
  MV_CC_DEVICE_INFO_LIST      *pstDevList,
  const char                  *strManufacturerName,
  MV_SORT_METHOD              enSortMethod
);
```

Parameters

nTLayerType

[IN] Transport layer protocol type, indicated by bit, supporting multiple selections. Different types of devices are distinguished. The available protocol types are shown in the table below:

Macro Definition	Value	Description
MV_UNKNOW_DEVICE	0x00000000	Unknown device type
MV_GIGE_DEVICE	0x00000001	GigE device
MV_1394_DEVICE	0x00000002	1394-a/b device
MV_USB_DEVICE	0x00000004	USB3.0 device
MV_CAMERALINK_DEVICE	0x00000008	CameraLink device
MV_VIR_GIGE_DEVICE	0x00000010	Virtual GigE device
MV_VIR_USB_DEVICE	0x00000020	Virtual USB3.0 device
MV_GENTL_GIGE_DEVICE	0x00000040	Virtual CameraLink device

For example, if nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE, which means searching GigE and USB 3.0 device.

pstDevList

[IN][OUT] Device information list, see the structure ***MV_CC_DEVICE_INFO_LIST*** for details.

strManufacturerName

[IN] Manufacturer name, for example, "abc" (enumerate the cameras of abc). It can be set to NULL, which indicates to enumerate all devices according to the specified sorting type.

enSortMethod

[IN] The sorting type, see the enumeration ***MV_SORT_METHOD*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

The memory of storing device list is allocated by the SDK and will be released and applied for during multiple-threaded API calling. It is recommended to avoid the multiple-threaded enumeration.

4.1.5 MV_CC_IsDeviceAccessible

Check if the specified device can be accessed.

API Definition

```
bool MV_CC_IsDeviceAccessible(
  MV_CC_DEVICE_INFO    *pstDevInfo,
  unsigned int         nAccessMode
);
```

Parameters

pstDevInfo

[IN] Device information, see the structure ***MV_CC_DEVICE_INFO*** for details.

nAccessMode

[IN] Access type, see the table below for details.

Macro Definition	Value	Description
MV_ACCESS_Exclusive	1	Exclusive permission, for other apps, the CCP register is only allowed to be read
MV_ACCESS_ExclusiveWithSwitch	2	Preempt permission in mode 5, and then open with exclusive permission
MV_ACCESS_Control	3	Control permission, for other apps, all registers are allowed to be read
MV_ACCESS_ControlWithSwitch	4	Preempt permission in mode 5, and then open with control permission
MV_ACCESS_ControlSwitchEnable	5	Open with control permission that can be preempted
MV_ACCESS_ControlSwitchEnableWithKey	6	Preempt permission in mode 5, and then open with control permission that can be preempted
MV_ACCESS_Monitor	7	Open device with reading mode, suitable under control permission

Return Value

Return *true* to indicate the device is accessible, and return *false* to indicate no permission or the device is offline.

Remarks

- You can read the device CCP register value to check the current access permission.
- Return false if the device does not support the modes MV_ACCESS_ExclusiveWithSwitch, MV_ACCESS_ControlWithSwitch, MV_ACCESS_ControlSwitchEnableWithKey. Currently the

device does not support the 3 preemption modes, neither do the devices from other mainstream manufacturers.
- This API is not supported by CameraLink device.

See Also

MV_CC_CreateHandle

4.1.6 MV_CC_SetSDKLogPath

Set the SDK log saving path.

API Definition

```
int MV_CC_SetSDKLogPath(
 const char      *pSDKLogPath
);
```

Parameters

pSDKLogPath

[IN] SDK log saving path.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

For version 2.4.1 and above, the log service has added, and no need to set the log saving path, therefore this API is invalid when the log service is enabled.

4.1.7 MV_CC_CreateHandle

Create a handle.

API Definition

```
int MV_CC_CreateHandle(
  void                        *handle,
  const MV_CC_DEVICE_INFO     *pstDevInfo
);
```

Parameters

handle

[OUT] Device handle

pstDevInfo

[IN] Device information, including device version, MAC address, transport layer type and other device information, see the structure ***MV_CC_DEVICE_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Create required resources within library and initialize internal module according to input device information. Create handle and call SDK interface through this interface, and SDK log file will be created by default and will be saved in MvSdkLog folder under current executable program path. Creating handle through ***MV_CC_CreateHandleWithoutLog*** will not generate log files.

See Also

MV_CC_CreateHandleWithoutLog
MV_CC_EnumDevices
MV_CC_DestroyHandle

4.1.8 MV_CC_CreateHandleWithoutLog

Create a handle without log.

API Definition

```
int MV_CC_CreateHandleWithoutLog(
  void                          *handle,
  const MV_CC_DEVICE_INFO       *pstDevInfo
);
```

Parameters

handle

[OUT] Device handle

pstDevInfo

[IN] Device information, including device version, MAC address, transport layer type and other device information, see the structure ***MV_CC_DEVICE_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Create required resources within library and initialize internal module according to input device information. Create handle and call SDK interface through this interface, and SDK log file will not be created. To create logs, create handle through ***MV_CC_CreateHandle*** , and log files will be automatically generated and saved to MvSdkLog folder under current executable program path.

See Also

MV_CC_EnumDevices
MV_CC_DestroyHandle

4.1.9 MV_CC_DestroyHandle

Destroy device example and related resources.

API Definition

```
int MV_CC_DestroyHandle
  void    *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

See Also

MV_CC_CreateHandle

4.1.10 MV_CC_OpenDevice

Open the device (connect to the device).

API Definition

```
int MV_CC_OpenDevice(
  void                *handle,
  unsigned int        nAccessMode = MV_ACCESS_Exclusive,
  unsigned short      nSwitchoverKey = 0
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nAccessMode

[IN] Device access mode, it is exclusive mode by default, see the table below for details.

Macro Definition	Value	Description
MV_ACCESS_Exclusive	1	Exclusive permission, for other apps, the CCP register is only allowed to be read
MV_ACCESS_ExclusiveWithSwitch	2	Preempt permission in mode 5, and then open with exclusive permission
MV_ACCESS_Control	3	Control permission, for other apps, all registers are allowed to be read
MV_ACCESS_ControlWithSwitch	4	Preempt permission in mode 5, and then open with control permission
MV_ACCESS_ControlSwitchEnable	5	Open with control permission that can be preempted
MV_ACCESS_ ControlSwitchEnableWithKey	6	Preempt permission in mode 5, and then open with control permission that can be preempted
MV_ACCESS_Monitor	7	Open device with reading mode, suitable under control permission

nSwitchoverKey

[IN] Key for switching permissions, it is null by default, and it is valid when access mode supports permission switching (2/4/6 mode).

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- You can find the specific device and connect according to inputted device parameters.
- When calling this API, the parameters **nAccessMode** and **nSwitchoverKey** are optional, and the device access mode is exclusive by default. Currently the device does not support the following preemption modes: MV_ACCESS_ExclusiveWithSwitch, MV_ACCESS_ControlWithSwitch, and MV_ACCESS_ControlSwitchEnableWithKey.
- For USB3Vision device, the parameters **nAccessMode** and **nSwitchoverKey** are invalid.

See Also

MV_CC_CloseDevice

4.1.11 MV_CC_CloseDevice

Shut down the device.

API Definition

```
int MV_CC_CloseDevice(
  void     *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After connecting to device via calling API ***MV_CC_OpenDevice*** , you can call this API to disconnect and release resources.

See Also

MV_CC_OpenDevice

4.1.12 MV_CC_GetDeviceInfo

Get the device information.

API Definition

```
int MV_CC_GetDeviceInfo(
  void                  *handle,
  MV_CC_DEVICE_INFO     *pstDevInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstDevInfo

[OUT] Device information, see the structure ***MV_CC_DEVICE_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- The API is not supported by USB3 vision cameras.
- The API is not supported by CameraLink devices.

See Also

MV_CC_OpenDevice

4.1.13 MV_CC_GetValidImageNum

Get the number of valid images in current image cache

API Definition

```
int MV_CC_GetValidImageNum(
  void           *handle,
  unsigned int   *pnValidImageNum
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pnValidImageNum

[OUT] The pointer to the number of valid images in the current image cache.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

4.2 Parameter Settings

4.2.1 MV_CC_GetBoolValue

Get the camera value of type bool.

API Definition

```
int MV_CC_GetBoolValue(
  void          *handle,
  const char    *strKey,
  bool          *pBoolValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pBoolValue

[OUT] Obtained node value

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

After the device is connected, call this API to get specified bool nodes. The node values of IBoolean can be obtained through this API, **strKey** value corresponds to the Name column.

See Also

MV_CC_SetBoolValue

4.2.2 MV_CC_SetBoolValue

Set the value of camera bool type node.

API Definition

```
int MV_CC_SetBoolValue(
  void          *handle,
  const char    *strKey,
  bool          pBoolValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pBoolValue

[IN] Node value

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to set the value of specified bool node after connecting the device. All the node values of "IBoolean" can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetBoolValue

4.2.3 MV_CC_GetEnumValue

Get the value of camera Enum type node.

API Definition

```
int MV_CC_GetEnumValue(
  void               *handle,
  const char         *strKey,
  MVCC_ENUMVALUE     *pEnumValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pEnumValue

[OUT] Obtained node value, see the structure ***MVCC_ENUMVALUE*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After the device is connected, call this API to get specified Enum nodes. The node values of IEnumeration can be obtained through this API, **strKey** value corresponds to the Name column.

See Also

MV_CC_SetEnumValue

4.2.4 MV_CC_SetEnumValue

Set the value of camera Enum type node.

API Definition

```
int MV_CC_SetEnumValue(
  void            *handle,
  const char      *strKey,
  unsigned int    nValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

nValue

[IN] Node value

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

You can call this API to set specified Enum node after connecting the device. All the node values of "IEnumeration" in the list can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetEnumValue

4.2.5 MV_CC_SetEnumValueByString

Set the value of camera Enum type node.

API Definition

```
int MV_CC_SetEnumValueByString(
  void          *handle,
  const char   *strKey,
  const char   *sValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

sValue

[IN] Camera property string to be set

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to set specified Enum node after connecting the device. All the node values of "IEnumeration" in the list can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetEnumValue
MV_CC_SetEnumValue

4.2.6 MV_CC_GetEnumEntrySymbolic

Get the enumerator name according to the node name and assigned value.

API Definition

```
int MV_CC_GetEnumEntrySymbolic(
  void             *handle,
  const char       *strKey,
  MVCC_ENUMENTRY   *pstEnumEntry
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name (property key). For example, the node name of pixel format is "PixelFormat". Refer to the Name (key) column in MvCameraNode for details.

pstEnumEntry

[IN][OUT] The enumerator name, see ***MVCC_ENUMENTRY*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After connecting to the device, you can get the enumerator name according to the node name and assigned value.
- For the IEnumeration type nodes in MvCameraNode, their enumerator names can be obtained via this API.

4.2.7 MV_CC_GetFloatValue

Get the value of camera float type node.

API Definition

```
int MV_CC_GetFloatValue(
  void               *handle,
  const char         *strKey,
  MVCC_FLOATVALUE    *pFloatValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pFloatValue

[OUT] Obtained node value, see the structure ***MVCC_FLOATVALUE*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get the value of specified float nodes after connecting the device. All the node values of "IFloat" in the list can be obtained via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_SetFloatValue

4.2.8 MV_CC_SetFloatValue

Set the value of camera float type node.

API Definition

```
int MV_CC_SetFloatValue(
  void          *handle,
  const char    *strKey,
  float         fValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

fValue

[IN] Node value

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

You can call this API to set specified float node after connecting the device. All the node values of "IFloat" in the list can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetFloatValue

4.2.9 MV_CC_GetIntValueEx

Get the value of camera integer type node (supports 64-bit).

API Definition

```
int MV_CC_GetIntValueEx(
  void                 *handle,
  const char           *strKey,
  MVCC_INTVALUE_EX     *pIntValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pIntValue

[OUT] Obtained node value, see the structure ***MVCC_INTVALUE_EX*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get the value of camera node with integer type after connecting the device. All the node values of "IInteger" in the list can be obtained via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_SetIntValueEx

4.2.10 MV_CC_SetIntValueEx

Set the value of camera integer type node (supports 64-bit).

API Definition

```
int MV_CC_SetIntValueEx(
  void            *handle,
  const char      *strKey,
  int64_t         nValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

nValue

[IN] Node value

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

You can call this API to set the value of camera node with integer type after connecting the device. All the node values of "IInteger" in the list can be set via this API. **strKey** corresponds to the Name column.

4.2.11 MV_CC_GetStringValue

Get the value of camera string type node.

API Definition

```
int MV_CC_GetStringValue(
  void                 *handle,
  const char           *strKey,
  MVCC_STRINGVALUE     *pStringValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

pStringValue

[OUT] Obtained node value, see the structure ***MVCC_STRINGVALUE*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get specified string node after connecting the device. All the node values of "IString" in the list can be obtained via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_SetStringValue

4.2.12 MV_CC_SetStringValue

Set the camera value of type string.

API Definition

```
int MV_CC_SetStringValue(
  void          *handle,
  const char    *strKey,
  const char    *sValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

sValue

[IN] Node value

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

You can call this API to set the specified string type node after connecting the device. All the node values of "IString" in the list can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetStringValue

4.2.13 MV_CC_SetCommandValue

Set the camera Command node.

API Definition

```
int MV_CC_SetCommandValue(
  void             *handle,
  const char       *strKey
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to set specified Command node after connecting the device. All the node values of "ICommand" in the list can be set via this API. **strKey** corresponds to the Name column.

4.2.14 MV_CC_ReadMemory

Read data from device register.

API Definition

```
int MV_CC_ReadMemory(
  void       *handle,
  void       *pBuffer,
  __int64    nAddress,
  __int64    nLength
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pBuffer

[OUT] Data buffer, saving memory value that is read (memory value is stored based on big endian mode)

nAddress

[IN] Memory address to be read, the address can be obtained from Camera.xml, in a form similar to xml node value of xxx_RegAddr (Camera.xml will automatically generate in current program directory after the device is opened).

nLength

[IN] Length of memory to be read

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Access device, read the data from certain register.

See Also

MV_CC_WriteMemory

4.2.15 MV_CC_WriteMemory

Write data into device register.

API Definition

```
int MV_CC_WriteMemory(
  void           *handle,
  const void     *pBuffer,
  __int64        nAddress,
  __int64        nLength
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pBuffer

[OUT] Memory value to be written (the value is to be stored according to big endian mode)

nAddress

[IN] Memory address to be written, the address can be obtained from Camera.xml, in a form similar to xml node value of xxx_RegAddr (Camera.xml will automatically generate in current program directory after the device is opened).

nLength

[IN] Length of memory to be written

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Access device, write a piece of data into a certain segment of register.

See Also

MV_CC_ReadMemory

4.3 Functional

4.3.1 General APIs

MV_CC_FeatureLoad

Import camera property files in XML format.

API Definition

```
int MV_CC_FeatureLoad(
  void           *handle,
  const char     *pFileName
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pFileName

[IN] Camera property file name.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

See Also

MV_CC_FeatureSave

MV_CC_FeatureSave

Save the camera property file in XML format.

API Definition

```
int MV_CC_FeatureSave(
  void         *handle,
  const char   *pFileName
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pFileName

[IN] Camera property file name.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After connecting to the device, you can call this API to save the camera property file to the local PC.

See Also

MV_CC_FeatureLoad

MV_CC_FileAccessRead

Read files from camera.

API Definition

```
int MV_CC_FileAccessRead(
  void                *handle,
  MV_CC_FILE_ACCESS   pstFileAccess
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstFileAccess

[IN] Structure for getting or saving files, see the structure ***MV_CC_FILE_ACCESS*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After connecting to the device, you can call this API to read files from the camera and save them to local PC.

See Also

MV_CC_FileAccessWrite

MV_CC_FileAccessWrite

Write local files to the camera.

API Definition

```
int MV_CC_FileAccessWrite(
  void                *handle,
  MV_CC_FILE_ACCESS   pstFileAccess
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstFileAccess

[IN] Structure for saving files, see the structure ***MV_CC_FILE_ACCESS*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API should be called after connecting to device.

See Also

MV_CC_FileAccessRead

MV_CC_GetAllMatchInfo

Get the information of all types.

API Definition

```
int MV_CC_GetAllMatchInfo(
  void                 *handle,
  MV_ALL_MATCH_INFO    *pstInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstInfo

[IN] [OUT] Information structure, see ***MV_ALL_MATCH_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Input required information type (specify nType in structure MV_ALL_MATCH_INFO) in the API and get corresponding information (return in pInfo of structure MV_ALL_MATCH_INFO).
- The calling precondition of this API is determined by obtained information type. Call after enabling capture to get MV_MATCH_TYPE_NET_DETECT information of GigE device, and call after starting device to get MV_MATCH_TYPE_USB_DETECT information of USB3Vision device.
- This API is not supported by CameraLink device.

See Also

MV_CC_StartGrabbing

MV_CC_GetFileAccessProgress

Get the progress of importing and exporting camera parameters.

API Definition

```
int MV_CC_GetFileAccessProgress(
  void                          *handle,
  MV_CC_FILE_ACCESS_PROGRESS    *pstFileAccessProgress
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstFileAccessProgress

[IN] Progress, see details in ***MV_CC_FILE_ACCESS_PROGRESS*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_CC_IsDeviceConnected

Check if device is connected.

API Definition

```
bool MV_CC_IsDeviceConnected(
  void        *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_CC_RegisterAllEventCallBack

Register the callback function for multiple events.

API Definition

```
int MV_CC_RegisterAllEventCallBack(
  void              *handle,
  void              *cbEvent,
  void              *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbEvent

[IN] Callback function for receiving events, see the details below.

```
void(__stdcall* cbEvent)(
  MV_EVENT_OUT_INFO    *pEventInfo,
  void                 *pUser
);
```

pEventInfo

Output event information, see the enumeration ***MV_EVENT_OUT_INFO*** for details.

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API to set the event callback function to get the event information, such as acquisition, exposure, and so on.
- This API is not supported by CameraLink device.

See Also

MV_CC_OpenDevice

MV_CC_RegisterEventCallBackEx

Register single event callback function.

API Definition

```
int MV_CC_RegisterEventCallBackEx(
  void         *handle,
  const char   *pEventName,
  void         *cbEvent,
  void         *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pEventName

[IN] Event name

cbEvent

[IN] Callback function for receiving event information, see details below:

```
void(__stdcall* cbEvent)(
  MV_EVENT_OUT_INFO      *pEventInfo,
  void                   *pUser
);
```

pEventInfo

Output event information, see enumeration ***MV_EVENT_OUT_INFO*** for details.

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API to set the event callback function to get the event information, such as acquisition, exposure, and so on.
- This API is supported by CameraLink device only for device offline event.

See Also

MV_CC_RegisterAllEventCallBack

MV_CC_RegisterExceptionCallBack

Register exception message callback.

API Definition

```
int MV_CC_RegisterExceptionCallBack(
  void           *handle,
  void           *cbException,
  void            *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbException

[IN] Callback function to receive exception messages, see the details below:

```
void(__stdcall* cbException)(
  unsigned int    nMsgType,
  void            *pUser
);
```

nMsgType

Exception message type

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API after the device is opened by ***MV_CC_OpenDevice*** . When device is exceptionally disconnected, the exception message can be obtained from callback function. For disconnected GigE device, first call ***MV_CC_CloseDevice*** to shut down device, and then call ***MV_CC_OpenDevice*** to reopen the device.
- For exception message type macro definition see below:

Macro Definition	Value	Description
MV_GIGE_EXCEPTION_DEV_ DISCONNECT	0x00008001	Device disconnected.

- This API is not supported by CameraLink device.

See Also

MV_CC_OpenDevice

MV_CC_SetGrabStrategy

Set the streaming strategy.

API Definition

```
int MV_CC_SetGrabStrategy(
  void                  *handle
  MV_GRAB_STRATEGY      enGrabStrategy
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

enGrabStrategy

[IN] Streaming strategy, see the enumeration ***MV_GRAB_STRATEGY*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

There are four defined streaming strategies, from which you can choose the suitable one according to the actual requirement. See the detailed streaming strategies below.

Macro Definition	Description
OneByOne	Get image frames one by one in the chronological order, it is the default strategy.
LatestImagesOnly	Only get the latest one frame from the output buffer list, and clear the rest images in the list.
LatestImages	Get the latest image from the output buffer list, and the quantity of frames depends on the parameter **OutputQueueSize**, value range: [1,ImageNodeNum]. If the **OutputQueueSize** values "1", the strategy is same to "LatestImagesOnly", and if the **OutputQueueSize** values "ImageNodeNum", the strategy is same to "OneByOne".

Macro Definition	Description
	Note • You can set the **OutputQueueSize** via API . • You can set the **ImageNodeNum** via API ***MV_CC_SetImageNodeNum***
UpcomingImage	Ignore all the images in the output buffer list and wait for the next upcoming frame. **Note** This strategy is supported only by GigE camera.

Example

The following sample code is for reference only.

```
#include "stdio.h"
#include "Windows.h"
#include "process.h"
#include "conio.h"
#include "MvCameraControl.h"

//Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }

    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall UpcomingThread(void* pUser)
{
    Sleep(3000);

    MV_CC_SetCommandValue(pUser, "TriggerSoftware");

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char * pData = NULL;

    do
    {
        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }

        //Select device and create handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Open device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Set trigger mode and trigger source
        nRet = MV_CC_SetEnumValueByString(handle, "TriggerMode", "On");
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }
        nRet = MV_CC_SetEnumValueByString(handle, "TriggerSource", "Software");
        if (MV_OK != nRet)
        {
            printf("Set Trigger Source fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nImageNodeNum = 5;
        //Set number of image nodes
        nRet = MV_CC_SetImageNodeNum(handle, nImageNodeNum);
        if (MV_OK != nRet)
        {
            printf("Set number of image node fail! nRet [0x%x]\n", nRet);
            break;
        }


printf("\n*******************************************************************
****\n");
        printf("* 0.MV_GrabStrategy_OneByOne;
1.MV_GrabStrategy_LatestImagesOnly;  *\n");
        printf("* 2.MV_GrabStrategy_LatestImages;
3.MV_GrabStrategy_UpcomingImage;     *\n");

printf("*******************************************************************
**\n");

        printf("Please Intput Grab Strategy:");
        unsigned int nGrabStrategy = 0;
        scanf("%d", &nGrabStrategy);

        //U3V device does not support strategy of UpcomingImage
        if (nGrabStrategy == MV_GrabStrategy_UpcomingImage && MV_USB_DEVICE ==
stDeviceList.pDeviceInfo[nIndex]->nTLayerType)
        {
            printf("U3V device not support UpcomingImage\n");
            break;
        }

        switch(nGrabStrategy)
        {
        case MV_GrabStrategy_OneByOne:
            {
                printf("Grab using the MV_GrabStrategy_OneByOne default strategy
\n");
                nRet = MV_CC_SetGrabStrategy(handle, MV_GrabStrategy_OneByOne);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_LatestImagesOnly:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImagesOnly
\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImagesOnly);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_LatestImages:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImages\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImages);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                //Set output queue size
                nRet = MV_CC_SetOutputQueueSize(handle, 2);
                if (MV_OK != nRet)
                {
                    printf("Set Output Queue Size fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_UpcomingImage:
            {
                printf("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_UpcomingImage);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                unsigned int nThreadID = 0;
                void* hThreadHandle = (void*) _beginthreadex( NULL , 0 ,
UpcomingThread , handle, 0 , &nThreadID );
                if (NULL == hThreadHandle)
                {
                    break;
                }
            }
            break;
        default:
            printf("Input error!Use default strategy:MV_GrabStrategy_OneByOne
\n");
            break;
        }

        //Start grabbing image
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Send trigger software command
        for (unsigned int i = 0;i < nImageNodeNum;i++)
        {
            nRet = MV_CC_SetCommandValue(handle, "TriggerSoftware");
            if (MV_OK != nRet)
            {
                printf("Send Trigger Software command fail! nRet [0x%x]\n",
nRet);
                break;
            }
            Sleep(500);//Make sure that the trigger software command takes
effect and the last frame data has been stored in buffer list
        }

        MV_FRAME_OUT stOutFrame = {0};
        if (nGrabStrategy != MV_GrabStrategy_UpcomingImage)
        {
            while(true)
            {
                nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 0);
                if (nRet == MV_OK)
                {
                    printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d]
\n",
                        stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);
                }
                else
                {
                    break;
                }

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
        }
        else//Only for upcoming
        {
            nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 5000);
            if (nRet == MV_OK)
            {
                printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d]\n",
                    stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
            else
            {
                printf("No data[0x%x]\n", nRet);
            }
        }

        //Stop grabbing image
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Close device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Destroy handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

MV_CC_SetImageNodeNum

Set the number of SDK internal image buffer nodes.

API Definition

```
int MV_CC_SetImageNodeNum(
  void              *handle,
  unsigned int      nNum
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nNum

[IN] The number of SDK internal image buffer nodes; its value should be larger than or equal to 1, and the default value is "1".

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API to set the number of SDK internal image buffer nodes. The API should be called before calling ***MV_CC_StartGrabbing*** for capturing.
- This API is not supported by CameraLink device.

See Also

MV_CC_OpenDevice

MV_CC_SetOutputQueueSize

Set the output queue size.

API Definition

```
int MV_CC_SetOutputQueueSize(
  void                *handle
  unsigned int        nOutputQueueSize
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nOutputQueueSize

[IN] Output queue size, range: [1,10].

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is valid only when the streaming strategy is "LatestImages". You can set the maximum number of frames that can be stored in the buffer.

Example

The following sample code is for reference only.

```
#include "stdio.h"
#include "Windows.h"
#include "process.h"
#include "conio.h"
#include "MvCameraControl.h"

//Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall UpcomingThread(void* pUser)
{
    Sleep(3000);

    MV_CC_SetCommandValue(pUser, "TriggerSoftware");

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char * pData = NULL;

    do
    {
        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }

        //Select device and create handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Open device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nImageNodeNum = 5;
        //Set the number of image nodes
        nRet = MV_CC_SetImageNodeNum(handle, nImageNodeNum);
        if (MV_OK != nRet)
        {
            printf("Set number of image node fail! nRet [0x%x]\n", nRet);
            break;
        }


printf("\n*****************************************************************
****\n");
        printf("* 0.MV_GrabStrategy_OneByOne;
1.MV_GrabStrategy_LatestImagesOnly;  *\n");
        printf("* 2.MV_GrabStrategy_LatestImages;
3.MV_GrabStrategy_UpcomingImage;     *\n");

printf("*****************************************************************
**\n");

        printf("Please Intput Grab Strategy:");
        unsigned int nGrabStrategy = 0;
        scanf("%d", &nGrabStrategy);

        //U3V device does not support UpcomingImage
        if (nGrabStrategy == MV_GrabStrategy_UpcomingImage && MV_USB_DEVICE ==
stDeviceList.pDeviceInfo[nIndex]->nTLayerType)
        {
            printf("U3V device not support UpcomingImage\n");
            break;
        }

        switch(nGrabStrategy)
        {
            case MV_GrabStrategy_OneByOne:
            {
                printf("Grab using the MV_GrabStrategy_OneByOne default strategy
\n");
                nRet = MV_CC_SetGrabStrategy(handle, MV_GrabStrategy_OneByOne);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
            case MV_GrabStrategy_LatestImagesOnly:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImagesOnly
\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImagesOnly);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
            case MV_GrabStrategy_LatestImages:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImages\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImages);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                //Set output queue size
                nRet = MV_CC_SetOutputQueueSize(handle, 2);
                if (MV_OK != nRet)
                {
                    printf("Set Output Queue Size fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
            case MV_GrabStrategy_UpcomingImage:
            {
                printf("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_UpcomingImage);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                unsigned int nThreadID = 0;
                void* hThreadHandle = (void*) _beginthreadex( NULL , 0 ,
UpcomingThread , handle, 0 , &nThreadID );
                if (NULL == hThreadHandle)
                {
                    break;
                }
            }
            break;
            default:
            printf("Input error!Use default strategy:MV_GrabStrategy_OneByOne
\n");
            break;
        }
        //Start grabbing image
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Send trigger software command
        for (unsigned int i = 0;i < nImageNodeNum;i++)
        {
            nRet = MV_CC_SetCommandValue(handle, "TriggerSoftware");
            if (MV_OK != nRet)
            {
                printf("Send Trigger Software command fail! nRet [0x%x]\n", 
nRet);
                break;
            }
            Sleep(500);//Make sure that the trigger software command takes 
effect and the last frame data has been stored in buffer list
        }

        MV_FRAME_OUT stOutFrame = {0};
        if (nGrabStrategy != MV_GrabStrategy_UpcomingImage)
        {
            while(true)
            {
                nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 0);
                if (nRet == MV_OK)
                {
                    printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d] 
\n",
                        stOutFrame.stFrameInfo.nWidth, 
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);
                }
                else
                {
                    break;
                }

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
        }
        else//Only for upcoming
        {
            nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 5000);
            if (nRet == MV_OK)
            {
                printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d]\n",
                    stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
            else
            {
                printf("No data[0x%x]\n", nRet);
            }
        }

        //Stop grabbing image
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Close device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Destroy handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

MV_CC_OpenParamsGUI

Open the Graphical User Interface (GUI) of camera parameters configurations.

API Definition

```
int MV_CC_OpenParamsGUI(
  void      *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get or set camera parameters after connecting to the camera via API ***MV_CC_OpenDevice*** .

4.3.2 GigE APIs

MV_CC_GetOptimalPacketSize

Get the optimal packet size.

API Definition

```
int MV_CC_GetOptimalPacketSize(
  void     *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

If succeed, the return value is larger than 0, which refers to the packet size; if failed, the return value is smaller than 0, which refers to the corresponding ***Error Code*** .

Remarks

- The optimized packet size is the size of a packet transported via the network. For GigEVision device it is SCPS, and for USB3Vision device it is the size of packet read from driver each time. The API should be called after ***MV_CC_OpenDevice*** and before ***MV_CC_StartGrabbing*** .
- This API is supported only by GigE camera, it is not supported by USB3 or CameraLink device.

See Also

MV_CC_OpenDevice
MV_CC_StartGrabbing

MV_GIGE_ForceIpEx

Force camera network parameter, including IP address, subnet mask, default gateway.

API Definition

```
int MV_GIGE_ForceIpEx(
  void            *handle,
  unsigned int    nIP,
  unsigned int    nSubNetMask,
  unsigned int    nDefaultGateWay
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nIP

[IN] IP address

nSubNetMask

[IN] Subnet mask

nDefaultGateWay

[IN] Default gateway

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- This function is supported only by GigEVision cameras.
- After forcing camera network parameters (including IP address, subnet mask, default gateway), you should create the device handle again.
- If device is in DHCP status, after calling this API to force camera network parameter, the device will restart.

MV_GIGE_GetGvspTimeout

Get GVSP streaming timeout.

API Definition

```
int MV_GIGE_getGvspTimeout(
  void                      *handle,
  unsigned int              *pnMillilsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pnMillilsec

[IN] [OUT] Timeout period, unit: millisecond

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

MV_GIGE_SetGvspTimeout

Set GVSP streaming timeout.

API Definition

```
int MV_GIGE_SetGvspTimeout(
  void                      *handle,
  unsigned int              nMillilsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nMillilsec

[IN] Timeout period, which is 300 by default, and the minimum value is 10, unit: millisecond

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

MV_GIGE_GetResendMaxRetryTimes

Get the maximum times one packet can be resent.

API Definition

```
int MV_GIGE_GetResendMaxRetryTimes(
  void                 *handle,
  unsigned int         *pnRetryTimes
);
```

Parameters

handle

[IN] Device handle

pnRetryTimes

[OUT] The maximum times one packet can be resent.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

You should call this API after enabling the function of resending packets by calling ***MV_GIGE_SetResend*** .

MV_GIGE_SetResendMaxRetryTimes

Set the maximum times one packet can be resent.

API Definition

```
int MV_GIGE_SetResendMaxRetryTimes(
  void                  *handle,
  unsigned int          nRetryTimes
);
```

Parameters

handle

[IN] Device handle

nRetryTimes

[IN] The maximum times one packet can be resent, which is 20 by default, and the minimum value is 0.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

You should call this API after enabling the function of resending packets by calling ***MV_GIGE_SetResend*** .

MV_GIGE_GetResendTimeInterval

Get the packet resending interval.

API Definition

```
int MV_GIGE_GetResendTimeInterval(
  void                    *handle,
  unsigned int            *pnMillilsec
);
```

Parameters

handle

[IN] Device handle

pnMillilsec

[IN][OUT] Packet resending interval, unit: millisecond

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

You should call this API after enabling the function of resending packets by calling ***MV_GIGE_SetResend*** .

MV_GIGE_SetResendTimeInterval

Set the packet resending interval.

API Definition

```
int MV_GIGE_SetResendTimeInterval(
  void                 *handle,
  unsigned int         nMillilsec
);
```

Parameters

handle

[IN] Device handle

nMillilsec

[IN] Packet resending interval, which is 10 by default, unit: millisecond

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

You should call this API after enabling the function of resending packets by calling ***MV_GIGE_SetResend*** .

MV_GIGE_GetMulticastStatus

Get the device multicast status.

API Definition

```
int MV_GIGE_GetMulticastStatus(
  unsigned int MV_CC_DEVICE_INFO      pstDevInfo
  unsigned bool                       pStatus
);
```

Parameters

pstDevInfo

[IN] Device information structure, see ***MV_CC_DEVICE_INFO*** for details.

pStatus

[OUT] Status: "true"-in multicast, "false"-not inmulticast

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

When enumerating the device, you can call this API to check if the device is in multiple status without opening the device.

Example

The following sample code is for reference only.

```
MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE, &stDeviceList);
if (MV_OK != nRet)
{
    printf("Enum Devices fail! nRet [0x%x]\n", nRet);
    break;
}

if (stDeviceList.nDeviceNum > 0)
{
    bool bState = false;
    MV_GIGE_GetMulticastStatus(stDeviceList.pDeviceInfo[i], &nState);
    printf("nState[%d]\n", nState);
}
```

MV_GIGE_GetNetTransInfo

Get network transmission information, including received data size, number of lost frames.

API Definition

```
int MV_GIGE_GetNetTransInfo(
  void                *handle,
  MV_NETTRANS_INFO    *pstInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstInfo

[OUT] Network transmission information, including received data size, number of lost frames, and so on. See ***MV_NETTRANS_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Call this API after starting image acquiring through ***MV_CC_StartGrabbing*** .
This API is supported only by GigEVision Camera.

MV_GIGE_IssueActionCommand

Send PTP (Precision Time Protocol) command of taking photo.

API Definition

```
int MV_GIGE_IssueActionCommand(
  MV_ACTION_CMD_INFO           *pstActionCmdInfo,
  MV_ACTION_CMD_RESULT_LIST    *pstActionCmdResults
);
```

Parameters

pstActionCmdInfo

[IN] Command information, see the structure ***MV_ACTION_CMD_INFO*** for details.

pstActionCmdResults

[OUT] Returned information list, see the structure ***MV_ACTION_CMD_RESULT_LIST*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is supported only by GigEVision camera.

MV_GIGE_SetIpConfig

Configure IP mode.

API Definition

```
int MV_GIGE_SetIpConfig(
  void            *handle,
  unsigned int    nType
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nType

[IN] IP configuration mode, see the details below:

Macro Definition	Value	Description
MV_IP_CFG_STATIC	0x05000000	Static mode
MV_IP_CFG_DHCP	0x06000000	DHCP mode
MV_IP_CFG_LLA	0x04000000	LLA (Link-local address)

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- This API is valid only when the IP address is reachable, and after calling this API, the camera will reboot.
- Send command to set the MVC IP configuration mode, such as DHCP, LLA, and so on. This API is only supported by GigEVision camera.

MV_GIGE_SetNetTransMode

Set SDK internal priority network mode.

API Definition

```
int MV_GIGE_SetNetTransMode(
  void            *handle,
  unsigned int    nType
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nType

[IN] Network mode, see the details below:

Macro Definition	Value	Description
MV_NET_TRANS_DRIVER	0x00000001	Driver mode
MV_NET_TRANS_SOCKET	0x00000002	Socket mode

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

The internal priority network mode is driver mode by default, and supported only by GigEVision camera.

MV_GIGE_SetResend

Set parameters of resending packets.

API Definition

```
int MV_GIGE_SetResend(
  void            *handle,
  unsigned int    bEnable,
  unsigned int    nMaxResendPercent,
  unsigned int    nResendTimeout
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

bEnable

[IN] Enable resending packet or not: 0-Disable, 1-Enable

nMaxResendPercent

[IN] Maximum packet resending percentage, range: [0,100]

nResendTimeout

[IN] Packet resending timeout, unit: ms

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After the device is connected, call this API to set resend packet properties, supported only by GigEVision camera.

MV_GIGE_SetTransmissionType

Set transmission mode.

API Definition

```
int MV_GIGE_SetTransmissionType(
  void                         *handle,
  MV_TRANSMISSION_TYPE         *pstTransmissionType
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstTransmissionType

[IN] Transmission mode, see the structure ***MV_TRANSMISSION_TYPE*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Call this API to set the transmission mode as single cast mode and multicast mode. And this API is supported only by GigEVision camera.

4.3.3 CameraLink Camera

MV_CAML_GetDeviceBauderate

Get device baud rate.

API Definition

```
int MV_CAML_GetDeviceBauderate(
  void           *handle,
  unsigned int   *pnCurrentBaudrate
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pnCurrentBaudrate

[IN] Baud rate of current camera, supported baud rate is as follows:

Macro Definition	Value	Description
MV_CAML_BAUDRATE_9600	0x00000001	9600 baud rate
MV_CAML_BAUDRATE_19200	0x00000002	19200 baud rate
MV_CAML_BAUDRATE_38400	0x00000004	38400 baud rate
MV_CAML_BAUDRATE_57600	0x00000008	57600 baud rate
MV_CAML_BAUDRATE_115200	0x00000010	115200 baud rate
MV_CAML_BAUDRATE_230400	0x00000020	230400 baud rate
MV_CAML_BAUDRATE_460800	0x00000040	460800 baud rate
MV_CAML_BAUDRATE_921600	0x00000080	921600 baud rate
MV_CAML_BAUDRATE_AUTOMAX	0x40000000	The maximum self-adaptive baud rate

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is supported only by CameraLink device.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"
#include <stdio.h>
#include <Windows.h>
#include <conio.h>

//Wait for key input
void WaitForKeyPress(void)
```

```
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    bool bDevConnected = false;       //Whether device is connected
     do
    {

        //Enumerate device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_CAMERALINK_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

        // Select device and create handle
         nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[0]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Get device baud rate
         unsigned int nCurrentBaudrate = 0;
        nRet = MV_CAML_GetDeviceBauderate(handle, &nCurrentBaudrate);
        if (MV_OK != nRet)
        {
            printf("Get device bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Current device bauderate [0x%x]\n", nCurrentBaudrate);

        //Open specified device
         nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = true;
        printf("Current device bauderate [0x%x]\n", nCurrentBaudrate);

        //Shut device
         nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = false;

        //Destroy handle and release resource
         nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Device successfully closed.\n");
    } while (0);

    //Make sure the device is shutted down
    if (nRet != MV_OK)
    {
        if ( bDevConnected )
        {
            MV_CC_CloseDevice(handle);
            bDevConnected = false;
        }
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

MV_CAML_SetDeviceBauderate

Set device baud rate.

API Definition

```
int MV_CAML_SetDeviceBauderate(
  void                *handle,
  unsigned int        nBaudrate
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nBaudrate

[IN] Baud rate, supported baud rate is as follows:

Macro Definition	Value	Description
MV_CAML_BAUDRATE_9600	0x00000001	9600 baud rate
MV_CAML_BAUDRATE_19200	0x00000002	19200 baud rate
MV_CAML_BAUDRATE_38400	0x00000004	38400 baud rate
MV_CAML_BAUDRATE_57600	0x00000008	57600 baud rate
MV_CAML_BAUDRATE_115200	0x00000010	115200 baud rate
MV_CAML_BAUDRATE_230400	0x00000020	230400 baud rate
MV_CAML_BAUDRATE_460800	0x00000040	460800 baud rate
MV_CAML_BAUDRATE_921600	0x00000080	921600 baud rate
MV_CAML_BAUDRATE_AUTOMAX	0x40000000	The maximum self-adaptive baud rate

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is supported only by CameraLink device.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"
#include <stdio.h>
#include <Windows.h>
#include <conio.h>

//Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    bool bDevConnected = false;  //Whether device is connected
     do
    {

        //Enumerate device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_CAMERALINK_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

        //Select device and create handle
         nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[0]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Set device baud rate
         nRet = MV_CAML_SetDeviceBauderate(handle, MV_CAML_BAUDRATE_115200);
        if (MV_OK != nRet)
        {
            printf("Set Device Bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }
        //Open specified device
         nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = true;
        printf("Current device bauderate [0x%x]\n", nCurrentBaudrate);

        //Shut device
         nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = false;

        //Destroy handle and release resource
         nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Device successfully closed.\n");
    } while (0);

    //Make sure the device is shutted down
    if (nRet != MV_OK)
    {
        if ( bDevConnected )
        {
            MV_CC_CloseDevice(handle);
            bDevConnected = false;
        }
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

MV_CAML_GetSupportBauderates

Get supported baud rate for connecting device and host.

API Definition

```
int MV_CAML_GetSupportBauderates(
  void             *handle,
  unsigned int     *pnBaudrateAblity
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pnBaudrateAblity

[OUT] Supported baud rate or result in current environment, supported baud rate is as follows:

Macro Definition	Value	Description
MV_CAML_BAUDRATE_9600	0x00000001	9600 baud rate
MV_CAML_BAUDRATE_19200	0x00000002	19200 baud rate
MV_CAML_BAUDRATE_38400	0x00000004	38400 baud rate
MV_CAML_BAUDRATE_57600	0x00000008	57600 baud rate
MV_CAML_BAUDRATE_115200	0x00000010	115200 baud rate
MV_CAML_BAUDRATE_230400	0x00000020	230400 baud rate
MV_CAML_BAUDRATE_460800	0x00000040	460800 baud rate
MV_CAML_BAUDRATE_921600	0x00000080	921600 baud rate
MV_CAML_BAUDRATE_AUTOMAX	0x40000000	The maximum self-adaptive baud rate

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is supported only by CameraLink device.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"
#include <stdio.h>
#include <Windows.h>
#include <conio.h>

//Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    bool bDevConnected = false;  //Whether device is connected
     do
    {
        // Enumerate device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_CAMERALINK_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

        // Select device and create handle
         nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[0]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        /******************The following content is valid only for CameraLink
device*********************/
        // Get supported baud rate of connecting device and host
         unsigned int nBaudrateAblity = 0;
        nRet = MV_CAML_GetSupportBauderates(handle, &nBaudrateAblity);
        if (MV_OK != nRet)
        {
            printf("Get supported bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Current device supported bauderate [0x%x]\n", nBaudrateAblity);

        // Open specified device
         nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = true;

        // Shut device
         nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = false;

        //Destroy handle and release resource
         nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Device successfully closed.\n");
    } while (0);
    // Make sure the device is shutted down
     if (nRet != MV_OK)
    {
        if ( bDevConnected )
        {
            MV_CC_CloseDevice(handle);
            bDevConnected = false;
        }
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

MV_CAML_SetGenCPTimeOut

Set the waiting time of serial port operation.

API Definition

```
int MV_CAML_SetGenCPTimeOut(
  void              *handle,
  unsigned int      nMillisec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nMillisec

[IN] Waiting time of serial port operation, unit: ms

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"
#include <stdio.h>
#include <Windows.h>
#include <conio.h>

// Wait for entering
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    int nMillisec = 0;
    bool bDevConnected = false;  //Whether the device is connected to
     do
    {

        // Enumerate devices
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_CAMERALINK_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return;
        }

        // Select a device and create a device handle
         nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[0]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Set the waiting time of serial port operation
         nRet = MV_CAML_SetGenCPTimeOut(handle, nMillisec);
        if (MV_OK != nRet)
        {
            printf("Set Device Bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open specified device
         nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = true;
        printf("Current device bauderate [0x%x]\n", nCurrentBaudrate);

        // Shut down device
         nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = false;

        // Destroy handle and release resources
         nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Device successfully closed.\n");
    } while (0);

    // Make sure the device is shutted down
    if (nRet != MV_OK)
    {
        if ( bDevConnected )
        {
            MV_CC_CloseDevice(handle);
            bDevConnected = false;
        }
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

4.3.4 GenTL APIs

MV_CC_CreateHandleByGenTL

Create a device handle via GenTL device information.

API Definition

```
int MV_CC_CreateHandleByGenTL(
  void                        **handle,
  const MV_GENTL_DEV_INFO     *pDevInfo
);
```

Parameters

handle

[OUT] Device handle

pDevInfo

[IN] Device information structure pointer, see ***MV_GENTL_DEV_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Before calling this API, you should call API ***MV_CC_EnumDevicesByGenTL*** to get the device information first.

Example

The following sample code is for reference only.

```
int nRet  = MV_CC_CreateHandleByGenTL(&m_hDevHandle,
m_stDevList.pDeviceInfo[0]);
if (MV_OK != nRet)
{
    return nRet;
}
```

MV_CC_EnumDevicesByGenTL

Enumerate devices via GenTL interface.

API Definition

```
int MV_CC_EnumDevicesByGenTL(
  MV_GENTL_IF_INFO         *pstIFInfo,
  MV_GENTL_DEV_INFO_LIST   *pstDevList
);
```

Parameters

pstIFInfo

[IN] Interface information, see ***MV_GENTL_IF_INFO*** for details.

pstDevList

[IN] [OUT] Device list, see the structure ***MV_GENTL_DEV_INFO_LIST*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Before calling this API, you should call API ***MV_CC_EnumInterfacesByGenTL*** to enumerate the interface first.

Example
The following sample code is for reference only.

```
MV_GENTL_DEV_INFO_LIST m_stDevList;
memset(&m_stDevList, 0, sizeof(MV_GENTL_DEV_INFO_LIST));

//Enumerate all devices within subnet
int nRet = MV_CC_EnumDevicesByGenTL(m_stIFList.pIFInfo[0],&m_stDevList);
if (MV_OK != nRet)
{
    return;
}
```

MV_CC_EnumInterfacesByGenTL

Enumerate interfaces via GenTL.

API Definition

```
int MV_CC_EnumInterfacesByGenTL(
  MV_GENTL_IF_INFO_LIST      *pstIFList,
  const char                 *pGenTLPath
);
```

Parameters

pstIFList

[IN] [OUT] Interface list, see the structure ***MV_GENTL_IF_INFO_LIST*** for details.

pGenTLPath

[IN] GenTL CTI file path

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

When importing the CTI file, you should check if the file has been saved. If the CTI file is saved, you can directly use the saved dynamic link library; if the CTI file is not saved, you should save it first and enumerate the interfaces.

Example
The following sample code is for reference only.

```
MV_GENTL_IF_INFO_LIST  m_stIFList;
memset(&m_stIFList, 0, sizeof(MV_GENTL_IF_INFO_LIST));
//Enumerate interfaces based on GenTL
int nRet = MV_CC_EnumInterfacsByGenTL(&m_stIFList, chCtiPath);
if (MV_OK != nRet)
{
    return;
}
```

MV_CC_UnloadGenTLLibrary

Unload the CTI library.

API Definition

```
int MV_CC_UnloadGenTLLibrary(
  const char      *pGenTLPath
);
```

Parameters

pGenTLPath

[IN] The storage path of CTI files.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Make sure that all devices enumerated by the CTI file are closed before calling this API, otherwise, an error will occur.

4.4 Image Acquisition

4.4.1 MV_CC_ClearImageBuffer

Clear the streaming data buffer.

API Definition

```
int MV_CC_ClearImageBuffer(
  void       *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- You can call this API to clear the needless images in the buffer even when the streaming is in progress.
- You can call this API to clear history data when the continuous mode is switched to the trigger mode.

4.4.2 MV_CC_FreeImageBuffer

Release image buffer (this API is used to release the image buffer, which is no longer used, and it should be used with API: MV_CC_GetImageBuffer).

API Definition

```
int MV_CC_FreeImageBuffer(
  void                     *handle,
  MV_FRAME_OUT             *pFrame,
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pFrame

[IN] Image data and information

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- This API and ***MV_CC_GetImageBuffer*** should be called in pairs, before calling ***MV_CC_GetImageBuffer*** to get image data pFrame, you should call MV_CC_FreeImageBuffer to release the permission.
- Compared with API ***MV_CC_GetOneFrameTimeout*** , this API has higher efficiency of image acquisition. The max. number of nodes can be outputted is same as the "**nNum**" of API ***MV_CC_SetImageNodeNum*** , default value is 1.
- This API is not supported by CameraLink device.
- This API is supported by both USB3 vision camera and GigE camera.

See Also

MV_CC_GetImageBuffer

4.4.3 MV_CC_GetImageBuffer

Get one frame of picture, support getting chunk information and setting timeout.

API Definition

```
int MV_CC_GetImageBuffer(
  void                  *handle,
  MV_FRAME_OUT          *pFrame,
  int                   nMsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pFrame

[OUT] Image data and information, see the structure ***MV_FRAME_OUT*** for details.

nMsec

[IN] Timeout duration, unit: millisecond

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Before calling this API to get image data frame, you should call ***MV_CC_StartGrabbing*** to start image acquisition. This API can get frame data actively, the upper layer program should control the frequency of calling this API according to the frame rate. This API supports setting timeout, and SDK will wait to return until data appears. This function will increase the streaming stability, which can be used in the situation with high stability requirement.
- This API and ***MV_CC_FreeImageBuffer*** should be called in pairs, after processing the acquired data, you should call MV_CC_FreeImageBuffer to release the data pointer permission of **pFrame**.
- This API whose streaming buffer is allocated by the SDK automatically, has higher image acquisition efficiency than ***MV_CC_GetOneFrameTimeout*** (). Interface A is more efficient than interface B, because the buffer of interface A is automatically allocated by the SDK, and interface B is manually allocated by the user
- This API cannot be called to stream after calling ***MV_CC_DisplayOneFrame*** .

- This API is not supported by CameraLink device.
- This API is supported by both USB3 vision camera and GigE camera.

4.4.4 MV_CC_GetImageForBGR

Get a frame of BGR24 data, search the frame data in the memory and transform it to BGR24 format for return. Setting timeout is supported.

API Definition

```
int MV_CC_GetImageForBGR(
  void                     *handle,
  unsigned char            *pData,
  unsigned int             nDataSize,
  MV_FRAME_OUT_INFO_EX     *pFrameInfo,
  int                      nMsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN] Buffer address used to save image data

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, BGR24 format, see the structure ***MV_FRAME_OUT_INFO_EX*** for details.

nMsec

[IN] Waiting timeout, unit: millisecond

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Each time the API is called, the internal buffer is checked for data. If there is data, it will be transformed as BGR24 format for return, if there is no data, return error code. As time-consuming exists when transform the image to BGR24 format, this API may cause frame loss when the data frame rate is too high.

- Before calling this API to get image data frame, call ***MV_CC_StartGrabbing*** to start image acquisition. This API can get frame data actively, the upper layer program should control the frequency of calling this API according to the frame rate.
- This API is not supported by CameraLink device.

4.4.5 MV_CC_GetImageForRGB

Get a frame of RGB24 data, search the frame data in the memory and transform it to RGB24 format for return. Setting timeout is supported.

API Definition

```
int MV_CC_GetImageForRGB(
  void                     *handle,
  unsigned char            *pData,
  unsigned int             nDataSize,
  MV_FRAME_OUT_INFO_EX     *pFrameInfo,
  int                      nMsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN] Buffer address used to save image data

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, RGB24 format, see the structure ***MV_FRAME_OUT_INFO_EX*** for details.

nMsec

[IN] Waiting timeout, unit: millisecond

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Each time the API is called, the internal buffer is checked for data. If there is data, it will be transformed as RGB24 format for return, if there is no data, return error code. As time-

consuming exists when transform the image to RGB24 format, this API may cause frame loss when the data frame rate is too high.
- Before calling this API to get image data frame, call ***MV_CC_StartGrabbing*** to start image acquisition. This API can get frame data actively, the upper layer program should control the frequency of calling this API according to the frame rate.
- This API is not supported by CameraLink device.

4.4.6 MV_CC_GetOneFrameTimeout

Get one frame of picture, support getting chunk information and setting timeout.

API Definition

```
int MV_CC_GetOneFrameTimeout(
  void                    *handle,
  unsigned char           *pData,
  unsigned int            nDataSize,
  MV_FRAME_OUT_INFO_EX    *pFrameInfo,
  unsigned int            nMsec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN] Buffer address used to save image data

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, including chunk information, see the structure ***MV_FRAME_OUT_INFO_EX*** for details.

nMsec

[IN] Waiting timeout, unit: millisecond

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Before calling this API to get image data frame, call ***MV_CC_StartGrabbing*** to start image acquisition. This API can get frame data actively, the upper layer program should control the frequency of calling this API according to the frame rate. This API supports setting timeout, SDK

will wait to return until data appears. This function will increase the streaming stability, which can be used in the situation with high stability requirement.
- This API is supported by both the USB3Vision and GIGE camera.
- This API is not supported by CameraLink device.

4.4.7 MV_CC_RegisterImageCallBackEx

Register image data callback function, supporting getting chunk information.

API Definition

```
int MV_CC_RegisterImageCallBackEx(
  void        *handle,
  void        *cbOutput,
  void        *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbOutput

[IN] Image data callback function, see the details below:

```
void(__stdcall* cbOutput)(
  unsigned char        *pData,
  MV_FRAME_OUT_INFO_EX    *pFrameInfo,
  void                 *pUser
);
```

pData

Address of buffer that saves image data

pFrameInfo

Obtained frame information, including width, height and pixel format. See the structure ***MV_FRAME_OUT_INFO_EX*** for details

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After calling ***MV_CC_CreateHandle*** , call this API to set image data callback function.
- There are two available image data acquisition modes, and cannot be used together:
  1. Call ***MV_CC_RegisterImageCallBackEx*** to set image data callback function, and then call ***MV_CC_StartGrabbing*** to start acquiring. The acquired image data is returned in the configured callback function.
  2. Call ***MV_CC_StartGrabbing*** to start acquiring, and then call ***MV_CC_GetOneFrameTimeout*** repeatedly in application layer to get frame data of specified pixel format. When getting frame data, the frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is not supported by CameraLink device.

4.4.8 MV_CC_RegisterImageCallBackForBGR

Register BGR24 image data callback function, supports getting chunk information.

API Definition

```
int MV_CC_RegisterImageCallBackForBGR(
  void        *handle,
  void        *cbOutput,
  void        *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbOutput

[IN] BGR24 image data callback function, see the details below:

```
void(__stdcall* cbOutput)(
  unsigned char          *pData,
  MV_FRAME_OUT_INFO_EX   *pFrameInfo,
  void                   *pUser
);
```

pData

Address of buffer that saves image data

pFrameInfo

Obtained information of frame with BGR24 format, including width, height, pixel format, chunk information, and so on. See the structure ***MV_FRAME_OUT_INFO_EX*** for details.

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After calling ***MV_CC_CreateHandle*** , call this API to set image data callback function.
- There are two available image data acquisition modes, and cannot be used together:
  1. Call ***MV_CC_RegisterImageCallBackForBGR*** to set BGR24 format image data callback function, and then call ***MV_CC_StartGrabbing*** to start acquiring. The acquired image data is returned in the configured callback function.
  2. Call ***MV_CC_StartGrabbing*** to start acquiring, and then call ***MV_CC_GetImageForBGR*** repeatedly in application layer to get frame data with BGR24 format. When getting frame data, the frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is not supported by CameraLink device.

See Also

MV_CC_GetImageForBGR
MV_CC_StartGrabbing

4.4.9 MV_CC_RegisterImageCallBackForRGB

Register RGB24 image data callback function, supports getting chunk information.

API Definition

```
int MV_CC_RegisterImageCallBackForRGB(
  void        *handle,
  void        *cbOutput,
  void        *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbOutput

[IN] RGB24 image data callback function, see the details below:

```
void(__stdcall* cbOutput)(
  unsigned char           *pData,
  MV_FRAME_OUT_INFO_EX    *pFrameInfo,
  void                    *pUser
);
```

pData

Address of buffer that saves image data

pFrameInfo

Obtained information of frame with RGB24 format, including width, height, pixel format, chunk information, and so on. See the structure ***MV_FRAME_OUT_INFO_EX*** for details.

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After calling ***MV_CC_CreateHandle*** , call this API to set image data callback function.
- There are two available image data acquisition modes, and cannot be used together:
  1. Call ***MV_CC_RegisterImageCallBackForRGB*** to set RGB24 format image data callback function, and then call ***MV_CC_StartGrabbing*** to start acquiring. The acquired image data is returned in the configured callback function.
  2. Call ***MV_CC_StartGrabbing*** to start acquiring, and then call ***MV_CC_GetImageForRGB*** repeatedly in application layer to get frame data with RGB24 format. When getting frame data, the frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is not supported by CameraLink device.

See Also

MV_CC_StartGrabbing
MV_CC_GetImageForRGB

4.4.10 MV_CC_StartGrabbing

Start acquiring image.

API Definition

```
int MV_CC_StartGrabbing(
  void     *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is not supported by CameraLink device.

See Also

MV_CC_StopGrabbing

4.4.11 MV_CC_StopGrabbing

Stop acquiring images.

API Definition

```
int MV_CC_StopGrabbing(
  void     *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is not supported by CameraLink device.

See Also

MV_CC_StartGrabbing

4.5 Image Processing

4.5.1 MV_CC_DisplayOneFrame

Display one image frame.

API Definition

```
int MV_CC_DisplayOneFrame(
  void                     *handle,
  MV_DISPLAY_FRAME_INFO    *pDisplayInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pDisplayInfo

[IN] Image information, see the structure ***MV_DISPLAY_FRAME_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- This API is valid for USB3Vision camera and GIGE camera.
- This API is not supported by CameraLink device.

See Also

MV_CC_GetImageBuffer

4.5.2 MV_CC_SaveImageEx2

Convert the original image data to picture and save the pictures to specific memory, supports setting JPEG encoding quality.

API Definition

```
int MV_CC_SaveImageEx2(
  void*                     handle,
  MV_SAVE_IMAGE_PARAM_EX    *pSaveParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pSaveParam

[IN] [OUT] Input and output parameters of picture data, see the structure ***MV_SAVE_IMAGE_PARAM_EX*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Once there is image data, you can call this API to convert the data.
- You can also call ***MV_CC_GetOneFrameTimeout*** or ***MV_CC_RegisterImageCallBackEx*** or ***MV_CC_GetImageBuffer*** to get one image frame and set the callback function, and then call this API to convert the format.
- Comparing with the previous API MV_CC_SaveImageEx, this API added the parameter **handle** to ensure the unity with other API.

4.5.3 MV_CC_ConvertPixelType

Convert pixel format.

API Definition

```
int MV_CC_ConvertPixelType(
  void                          *handle,
  MV_CC_PIXEL_CONVERT_PARAM     *pstCvtParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstCvtParam

[IN] [OUT] Transform input and output parameter to pixel format, see the structure ***MV_CC_PIXEL_CONVERT_PARAM*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is used to convert the collected original data to required pixel format and save to specified memory. There is no calling sequence requirement, the transformation will be executed when there is image data. First call relative API to acquire the image, then call this API to convert the format.

4.5.4 MV_CC_SetBayerCvtQuality

Set the interpolation method of Bayer format.

API Definition

```
int MV_CC_SetBayerCvtQuality(
  void                             *handle,
  unsigned int                     nBayerCvtQuality
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nBayerCvtQuality

[IN] Interpolation method: 0-nearest neighbors, 1-bilinearity, 2-optimal; the default value is "0".

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Call this API to set the Bayer interpolation quality parameter for the image conversion API (***MV_CC_ConvertPixelType*** and ***MV_CC_SaveImageEx2***).

4.5.5 MV_CC_SetBayerGammaValue

Set the Gamma value after Bayer interpolation.

API Definition

```
int MV_CC_SetBayerGammaValue(
  void                     *handle,
  float                    fBayerGammaValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

fBayerGammaValue

[IN] Gamma value, range: [0.1, 4.0]

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

After setting Gamma value by calling this API, you can call ***MV_CC_ConvertPixelType*** or ***MV_CC_SaveImageEx2*** to convert Bayer format to RGB24/BGR24/RGBA32/BGRA32.

4.5.6 MV_CC_SetBayerGammaParam

Set gamma parameters of Bayer pattern.

API Definition

```
int __stdcall MV_CC_SetBayerGammaParam(
  void                   *handle,
  MV_CC_GAMMA_PARAM      *pstGammaParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstGammaParam

[IN] Gamma parameters structure. See ***MV_CC_GAMMA_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

The configured gamma parameters take effect when you call API ***MV_CC_ConvertPixelType*** or ***MV_CC_SaveImageEx2*** to convert the format of Bayer8/10/12/16 into RGB24/48, RGBA32/64, BGR24/48, or BGRA32/64.

4.5.7 MV_CC_SetBayerCCMParam

Color correction after Bayer interpolation.

API Definition

```
int MV_CC_SetBayerCCMParam(
  void                    *handle,
  MV_CC_CCM_PARAM         *pstCCMParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstCCMParam

[IN] Color correction structure, see ***MV_CC_CCM_PARAM*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

After calling this API, you can call ***MV_CC_ConvertPixelType*** or ***MV_CC_SaveImageEx2*** to convert Bayer format to RGB24/BGR24/RGBA32/BGRA32.

4.5.8 MV_CC_SetBayerCCMParamEx

Enable/disable CCM and set CCM parameters of Bayer pattern.

API Definition

```
int __stdcall MV_CC_SetBayerCCMParamEx(
  void                     *handle,
  MV_CC_CCM_PARAM_EX       *pstCCMParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstCCMParam

[IN] CCM parameter structure. See ***MV_CC_CCM_PARAM_EX*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

- After enabling color correction and setting color correction matrix, the CCM parameters take effect when you call API ***MV_CC_ConvertPixelType*** or ***MV_CC_SaveImageEx2*** to convert the format of Bayer8/10/12/16 into RGB24/48, RGBA32/64, BGR24/48, or BGRA32/64.
- This API is available for the device, which supports the function.

4.5.9 MV_CC_LSCCalib

This API is used for LSC calibration.

API Definition

```
int __stdcall MV_CC_LSCCalib(
  void                        *handle,
  MV_CC_LSC_CALIB_PARAM       *pstLSCCalibParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstLSCCalibParam

[IN] [OUT] Structure about LSC calibration parameters. See ***MV_CC_LSC_CALIB_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

4.5.10 MV_CC_LSCCorrect

This API is used for LSC correction.

API Definition

```
int __stdcall MV_CC_LSCCorrect(
  void                          *handle,
  MV_CC_LSC_CORRECT_PARAM     *pstLSCCorrectParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstLSCCorrectParam

[IN] [OUT] Structure about LSC correction parameters. See ***MV_CC_LSC_CORRECT_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

4.5.11 MV_CC_HB_Decode

Decode lossless compression stream into raw data.

API Definition

```
int MV_CC_HB_Decode(
  void                          *handle,
  MV_CC_HB_DECODE_PARAM         *pstDecodeParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstDecodeParam

[IN] Lossless decoding parameters structure, see ***MV_CC_HB_DECODE_PARAM*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

This API supports parsing the watermark of real-time images for the current camera. If the input lossless stream is not real-time, or it does not belong the current camera, an exception may occur during watermark parsing.

4.5.12 MV_CC_RotateImage

Rotate images in MONO8/RGB24/BGR24 format.

API Definition

```
int MV_CC_RotateImage(
  void                            *handle;
  MV_CC_ROTATE_IMAGE_PARAM        *pstRotateParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstRotateParam

[IN] [OUT] Image rotation structure, see ***MV_CC_ROTATE_IMAGE_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

4.5.13 MV_CC_FlipImage

Flip images in MONO8/RGB24/BGR24 format.

API Definition

```
int MV_CC_FlipImage(
  void                          *handle;
  MV_CC_FLIP_IMAGE_PARAM        *pstFlipParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstFlipParam

[IN] [OUT] Image flipping structure, see ***MV_CC_FLIP_IMAGE_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

4.5.14 MV_CC_StartRecord

Start recording.

API Definition

```
int MV_CC_StartRecord(
  void                    *handle,
  MV_CC_RECORD_PARAM      *pstRecordParam
  );
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstRecordParam

[IN] Video parameters

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

See Also

MV_CC_StopRecord
MV_CC_InputOneFrame

4.5.15 MV_CC_InputOneFrame

Transmit video parameters.

API Definition

```
int MV_CC_InputOneFrame(
  void                          *handle,
  MV_CC_INPUT_FRAME_INFO        *pstInputFrameInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstInputFrameInfo

[IN] Video data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

4.5.16 MV_CC_StopRecord

Stop recording.

API Definition

```
int MV_CC_StopRecord(
  void          *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

See Also

MV_CC_InputOneFrame
MV_CC_StartRecord

4.5.17 MV_CC_SaveImageToFile

Save image to file. Supported image format: BMP, JPEG, PNG, and TIFF.

API Definition

```
int MV_CC_SaveImageToFile(
  void                                  *handle
  unsigned int MV_SAVE_IMG_TO_FILE_PARAM      *pstSaveFileParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstSaveFileParam

[IN] [OUT] Structure about image saving parameters, see ***MV_SAVE_IMG_TO_FILE_PARAM*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Example

The following sample code is for reference only.

```
pstSaveFileParam.enImageType = MV_Image_Tif; //Image format to save
pstSaveFileParam.enPixelType = m_stImageInfo.enPixelType; //Camera pixel type
pstSaveFileParam.nWidth      = m_stImageInfo.nWidth;      //Camera width
pstSaveFileParam.nHeight     = m_stImageInfo.nHeight;     //Camera height
pstSaveFileParam.nDataLen    = m_stImageInfo.nFrameLen;
pstSaveFileParam.pData       = m_pGrabBuf;
pstSaveFileParam.nQuality     = 80;       // JPG encoding, it is valid only
valid when saving as JPG format
pstSaveFileParam.iMethodValue = 0;
pstSaveFileParam.pImagePath = "./Test.Tif"

int nRet = MV_CC_SaveImageToFile(m_hDevHandle, pstSaveFileParam);
if(MV_OK != nRet)
{
    return nRet;
}

pstSaveFileParam.enImageType = MV_Image_Tif; //Image format
pstSaveFileParam.enPixelType = m_stImageInfo.enPixelType;  //Camera pixel type
pstSaveFileParam.nWidth      = m_stImageInfo.nWidth;         //Camera Width
pstSaveFileParam.nHeight     = m_stImageInfo.nHeight;          //Camera Height
pstSaveFileParam.nDataLen    = m_stImageInfo.nFrameLen;
pstSaveFileParam.pData       = m_pGrabBuf;
pstSaveFileParam.nQuality     = 80;       //JPG encoding, it is valid only
valid when saving as JPG format
pstSaveFileParam.iMethodValue = 0;
pstSaveFileParam.pImagePath = "./Test.Tif"

int nRet = MV_CC_SaveImageToFile(m_hDevHandle, pstSaveFileParam);
if(MV_OK != nRet)
{
    return nRet;
}
```

4.5.18 MV_CC_SavePointCloudData

Save the 3D point cloud data. Supported formats are PLY, CSV, and OBJ.

API Definition

```
int MV_CC_SavePointCloudData(
  void                                       *handle
  unsigned int MV_SAVE_POINT_CLOUD_PARAM     *pstPointDataParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstPointDataParam

[IN] [OUT] Structure about parameters of saving 3D point cloud data, see ***MV_SAVE_POINT_CLOUD_PARAM***

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Example

The following sample code is for reference only.

```
MV_SAVE_POINT_CLOUD_PARAM stSavePoCloudPar = {0};

stSavePoCloudPar.nLineNum = stOutFrame.stFrameInfo.nWidth * nImageNum;
stSavePoCloudPar.nLinePntNum = stOutFrame.stFrameInfo.nHeight;

unsigned char* pDstImageBuf = (unsigned char*)malloc(stSavePoCloudPar.nLineNum
* stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048);
if (NULL == pDstImageBuf)
{
    printf("Malloc Dst buffer fail!\n");
    break;
}
unsigned int nDstImageSize = stSavePoCloudPar.nLineNum *
stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048;

stSavePoCloudPar.enPointCloudFileType = MV_PointCloudFile_PLY;
stSavePoCloudPar.enSrcPixelType = stOutFrame.stFrameInfo.enPixelType;
stSavePoCloudPar.pSrcData = pSaveImageBuf;
stSavePoCloudPar.nSrcDataLen = nSaveDataLen;
stSavePoCloudPar.pDstBuf = pDstImageBuf;
stSavePoCloudPar.nDstBufSize = nDstImageSize;

//Save point cloud data
nRet = MV_CC_SavePointCloudData(handle, &stSavePoCloudPar);
if(MV_OK != nRet)
{
    printf("Save point cloud data failed!nRet [0x%x]\n", nRet);
    break;
}
```

4.5.19 MV_CC_DrawCircle

Draw auxiliary circle frames on the image.

API Definition

```
int MV_CC_DrawCircle(
  void                    *handle,
  MVCC_CIRCLE_INFO        *pCircleInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pCircleInfo

[IN] The information about auxiliary circle frame, see ***MVCC_CIRCLE_INFO*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

4.5.20 MV_CC_DrawLines

Draw auxiliary lines on the image.

API Definition

```
int MV_CC_DrawLines(
  void                    *handle,
  MVCC_LINES_INFO         *pLinesInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pLinesInfo

[IN] The information about auxiliary line, see ***MVCC_LINES_INFO*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

4.5.21 MV_CC_DrawRect

Draw auxiliary rectangle frames on the image.

API Definition

```
int MV_CC_DrawRect(
  void                *handle,
  MVCC_RECT_INFO      *pRectInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pRectInfo

[IN] The information about auxiliary rectangle frame, see ***MVCC_RECT_INFO*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

4.5.22 MV_CC_ImageContrast

Adjust the image contrast.

API Definition

```
int MV_CC_ImageContrast(
  void                    *handle,
  MV_CC_CONTRAST_PARAM    *pstContrastParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstContrastParam

[IN][OUT] The image contrast, see ***MV_CC_CONTRAST_PARAM*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

4.5.23 MV_CC_ReconstructImage

Reconstruct the image (it is used in the time-division exposure).

API Definition

```
int MV_CC_RestructureImage(
  void                          *handle,
  MV_RESTRUCTURE_IMAGE_PARAM    *pstReconstructParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstReconstructParam

[IN][OUT] Image reconstruction parameters, see ***MV_RECONSTRUCT_IMAGE_PARAM*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

4.5.24 MV_CC_SetBayerFilterEnable

Enable or disable the smoothing function of interpolation algorithm.

API Definition

```
int MV_CC_SetBayerFilterEnable(
  void      *handle,
  bool      bFilterEnable
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

bFilterEnable

[IN] Whether to enable the smoothing function of interpolation algorithm: true (enable), false (disable).

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

This API is used to enable or disable the smoothing function of Bayer interpolation, which is used in the APIs of ***MV_CC_ConvertPixelType*** and ***MV_CC_SaveImageEx2*** .

4.6 Advanced Settings

4.6.1 General APIs

MV_CC_InvalidateNodes

Clear GenICam node cache.

API Definition

```
int MV_CC_InvalidateNodes(
  void             *handle
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

This API is used in the situation that GenICam node is not updated in time.

4.6.2 GigE APIs

MV_GIGE_GetGvcpTimeout

Get the GVCP command timeout.

API Definition

```
int MV_GIGE_GetGvcpTimeout(
  void             *handle,
  unsigned int     *pMillisec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pMillisec

[IN] Timeout pointer. The default value is 500. Unit: millisecond.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_GIGE_SetGvcpTimeout

Set the GVCP command timeout.

API Definition

```
int MV_GIGE_SetGvcpTimeout(
  void            *handle,
  unsigned int    nMillisec
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nMillisec

[IN] Heartbeat time, which defaults to 300, range: [10,10000], unit: ms.

Return Value

Return *MV_OK(0)* for success, and return ***Error Code*** for failure.

Remarks

After the device is connected, you can call this API to set the GVCP command timeout.

MV_GIGE_GetRetryGvcpTimes

Get the number of GVCP retransmission commands.

API Definition

```
int MV_GIGE_GetRetryGvcpTimes(
  void            *handle,
  unsigned int    *pRetryGvcpTimes
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pRetryGvcpTimes

[IN] Retransmission times pointer, the default value is 3.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_GIGE_SetRetryGvcpTimes

Set the GVCP command retransmission times.

API Definition

```
int MV_GIGE_SetRetryGvcpTimes(
  void            *handle,
  unsigned int    nRetryGvcpTimes
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nRetryGvcpTimes

[IN] Retransmission times, ranges from 0 to 100.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

When GVCP packet transmission is abnormal, you can call this API to set retransmission times to avoid the camera disconnection.

MV_GIGE_ SetDiscoveryMode

Set device ACK packet type.

API Definition

```
int MV_GIGE_SetDiscoveryMode(
  unsigned int    nMode
);
```

Parameters

nMode

[IN] Packet type: 0-unicast packet, 1-broadcast packet.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

The API is supported only by GigE cameras.

4.6.3 U3V APIs

MV_USB_GetTransferSize

Get the packet size of USB3 vision device.

API Definition

```
int MV_USB_GetTransferSize(
  void              *handle,
  unsigned int      *pTransferSize
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pTransferSize

[IN] Packet size, it is 1 MB by default.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_USB_SetTransferSize

Set the packet size of USB3 vision device.

API Definition

```
int MV_USB_SetTransferSize(
  void             *handle,
  unsigned int     nTransferSize
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nTransferSize

[IN] Packet size. The value is larger than or equal to 0x0800 (2 KB), the default value is 1 MB.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Increasing the packet size can reduce the CPU usage properly, but for different computer and USB expansion cards the compatibility are different, if the packet size is too large, the image may cannot be acquired.

MV_USB_GetTransferWays

Get the number of transmission channels for USB3 vision device.

API Definition

```
int MV_USB_GetTransferWays(
  void              *handle,
  unsigned int      *pTransferWays
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pTransferWays

[OUT] The number of transmission channels, range: [1,10]

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get the number of streaming nodes, for different pixel formats, the default values are different. For example, for 2 MP camera, the default value of MONO8 is 3, YUV is 2, RGB is 1, and other pixel format is 8.

MV_USB_RegisterStreamExceptionCallBack

Register the callback function for receiving the stream exceptions.

API Definition

```
int MV_USB_RegisterStreamExceptionCallBack(
  void      *handle,
  void      *cbException,
  void      *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

cbException

[IN] Callback function for receiving stream exceptions, see the details below.

```
void(__stdcall* cbException)(
  MV_CC_STREAM_EXCEPTION_TYPE     enExceptionType,
  void                            *pUser
);
```

enExceptionType

Stream exception types, see ***MV_CC_STREAM_EXCEPTION_TYPE*** for details.

pUser

User data

pUser

[IN] User data

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

- This API is supported by the USB3.0 cameras.
- Call this API after the device is opened.

MV_USB_SetEventNodeNum

Set the number of event cache nodes for USB3.0 cameras.

API Definition

```
int MV_USB_SetEventNodeNum(
  void              *handle,
  unsigned int      nEventNodeNum
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nEventNodeNum

[IN] The number of event cache nodes, range: [1,64].

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

MV_USB_SetTransferWays

Set the number of transmission channels for USB3 vision device.

API Definition

```
int MV_USB_SetTransferWays(
  void            *handle,
  unsigned int    nTransferWays
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nTransferWays

[IN] The number of transmission channels, range: [1,10]

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to set the number of transmission channels according to the factors of computer performance, output image frame rate, image size, memory usage, and so on. But you should notice that for different computer and USB expansion cards the compatibility are different.

4.7 Camera Internal APIs

4.7.1 MV_CC_GetUpgradeProcess

Get current upgrade progress.

API Definition

```
int MV_CC_GetUpgradeProcess(
  void             *handle,
  unsigned int     *pnProcess
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pnProcess

[OUT] Current upgrade progress, from 0 to 100

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

See Also

MV_CC_LocalUpgrade

4.7.2 MV_CC_LocalUpgrade

Upgrade the device locally.

API Definition

```
int MV_CC_LocalUpgrade(
  void          *handle,
  const void    *pFilePathName
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pFilePathName

[IN] Upgrade pack path, including folder absolute path or relative path.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API to send the upgrade firmware to the device for upgrade. This API waits for return until the upgrade firmware is sent to the device, this response may take a long time.
- For CameraLink device, it keeps sending upgrade firmware continuously.

See Also

MV_CC_OpenDevice
MV_CC_GetUpgradeProcess

4.7.3 MV_XML_GetGenICamXML

Get the camera description file in XML format.

API Definition

```
int MV_XML_GetGenICamXML(
  void                *handle,
  unsigned char       *pData,
  unsigned int        nDataSize,
  unsigned int        *pnDataLen
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN][OUT] The XML file buffer address

nDataSize

[IN] The XML file buffer size

pnDataLen

[OUT] The XML file length

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- When **pData** is NULL or when the value of **nDataSize** is larger than the actual XML file size, no data will be copied, and the XML file size is returned by **pnDataLen**.
- When **pData** is valid and the buffer size is enough, the complete data will be copied and stored in the buffer, and the XML file size is returned by **pnDataLen**.

4.7.4 MV_XML_GetNodeInterfaceType

Get the current node type.

API Definition

```
int MV_XML_GetNodeInterfaceType(
  void                    *handle,
  const char              *pstrName,
  MV_XML_InterfaceType    *pInterfaceType
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstrName

[IN] Node name

pInterfaceType

[OUT] API type corresponds to each node, see the enumeration ***MV_XML_InterfaceType*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

You can call this API to get the node type before getting or setting node value.

Example

The following sample code is for reference only.

```
#include
#include
#include
#include
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

//Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        //Print current IP address and user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }

        //Select device and create handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Open device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Get the current node type
        MV_XML_InterfaceType stXmlInterfaceType = {0};
        nRet = MV_XML_GetNodeInterfaceType(handle, "Width",
&stXmlInterfaceType);
        if (MV_OK != nRet)
        {
            printf("Get node Interface type fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Close device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Destroy handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

4.7.5 MV_XML_GetNodeAccessMode

Get current node access mode.

API Definition

```
int MV_XML_GetNodeAccessMode(
  void                 *handle,
  char                 *pstrName,
  MV_XML_AccessMode    *pAccessMode
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstrName

[IN] Node name

pAccessMode

[OUT] Node access mode, see the enumeration ***MV_XML_AccessMode*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Before getting or setting node value, you can call this API to get the node read and write permission to avoid failure.

Example

The following sample code is for reference only.

```
#include <stdio.h>
#include <Windows.h>
#include "process.h"
#include "conio.h"
#include "MvCameraControl.h"

//Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        //Print current IP address and user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }

        //Select device and create handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Open device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        MV_XML_AccessMode stAccessMode = {0};

        //Get the node access mode
        nRet = MV_XML_GetNodeAccessMode(handle, "Width",&stAccessMode);
        if(nRet != MV_OK)
        {
            printf("Warning: Get node access mode fail! nRet [0x%x]!", nRet);
            break;
        }

        //en:Close device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Destroy handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

4.8 Obsolete APIs

4.8.1 MV_CC_BayerSpatialDenoise

Spatial noise reduction of images in Bayer format (except Bayer RG 10 Packed).

API Definition

```
int MV_CC_BayerSpatialDenoise(
  void                                          *handle,
  MV_CC_BAYER_SPATIAL_DENOISE_PARAM             pstSpatialDenoiseParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstNoiseEstimateParam

[IN] [OUT] Structure about spatial noise reduction parameters, see MV_CC_BAYER_SPATIAL_DENOISE_PARAM for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

This API should be supported by cameras.

4.8.2 MV_CC_ColorCorrect

This API is used for color correction (including CCM and CLUT).

API Definition

```
int __stdcall MV_CC_ColorCorrect(
  void                         *handle,
  MV_CC_COLOR_CORRECT_PARAM    *pstColorCorrectParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstColorCorrectParam

[IN] Structure about color correction parameters. See ***MV_CC_COLOR_CORRECT_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

- This API supports configuring CCM and CLUT or together. You can enable or disable CCM and CLUT by configuring the members **bCCMEnable** and **bCLUTEnable** in the corresponding structures.
- This API is available for the device, which supports color correction.

4.8.3 MV_CC_Display

Display acquired images.

API Definition

```
int MV_CC_Display(
  void    *handle,
  void    *hWnd
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

hWnd

[IN] Window handle

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Call this API after MV_CC_StartGrabbing to display acquired images.
- This API is not supported by the camera with image in JPEG format.
- This API is not supported by CameraLink device.

Example

(For Windows system) The following sample code is for reference only.

```
#include "MvCameraControl.h"

typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
PROCGETCONSOLEWINDOW GetConsoleWindowAPI;

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], 
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Start acquiring images
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }


    //Get console window handle
    HMODULE hKernel32 = GetModuleHandle("kernel32");
    GetConsoleWindowAPI = 
(PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWindow");
    HWND hWnd = GetConsoleWindowAPI(); //window handle

    //Display images
    nRet = MV_CC_Display(m_handle,hWnd);
    if (MV_OK != nRet)
    {
        printf("error: Display fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Stop acquiring images
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

(For Linux system) The following sample code is for reference only.

```
#include "MvCameraControl.h"

#define MAX_IMAGE_DATA_SIZE   (20*1024*1024)
#define NIL (0)

// Press "Enter" to end streaming or exit
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    sleep(1);
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("%s\n" , "The Pointer of pstMVDevInfoList is NULL!");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
      // Print the IP address and user name of current camera
         printf("%s %x\n" , "nCurrentIp:" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.nCurrentIp);
        printf("%s %s\n\n" , "chUserDefinedName:" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName:%s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}

int main()
{
    Window w;
    Display *dpy;

    memset(&w, 0, sizeof(Window));
    dpy = NULL;
    printf("%d\n", sizeof(Window));

    // Create the window
     dpy = XOpenDisplay(NIL);

    int whiteColor = WhitePixel(dpy, DefaultScreen(dpy));

    w = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0,752, 480, 0,
0xffff00ff, 0xff00ffff);

    // We want to get MapNotify events
     XSelectInput(dpy, w, StructureNotifyMask);

    // "Map" the window (that is, make it appear on the screen)
     XMapWindow(dpy, w);

    // Create a "Graphics Context"
     GC gc = XCreateGC(dpy, w, 0, NIL);

    // Tell the GC we draw using the white color
     XSetForeground(dpy, gc, whiteColor);

    // Wait for the MapNotify event
     for(;;)
    {
      XEvent e;
    XNextEvent(dpy, &e);
    if (e.type == MapNotify)
    {
        break;
    }
    }

    int nRet = MV_OK;

    void* handle = NULL;

    MV_CC_DEVICE_INFO_LIST stDeviceList;
    memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));

    // Enumerate devices
     nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
    if (MV_OK != nRet)
    {
      printf("MV_CC_EnumDevices fail! nRet [%x]\n", nRet);
      return -1;
    }
    unsigned int nIndex = 0;
    if (stDeviceList.nDeviceNum > 0)
    {
      for (int i = 0; i < stDeviceList.nDeviceNum; i++)
      {
        printf("[device %d]:\n", i);
        MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
        if (NULL == pDeviceInfo)
          {
             break;
          }
        PrintDeviceInfo(pDeviceInfo);
      }
    }
    else
    {
       printf("Find No Devices!\n");
       return -1;
    }

    scanf("%d", &nIndex);

    // Select device and create handle
     nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
    if (MV_OK != nRet)
    {
       printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
       return -1;
    }

    // Open device
     nRet = MV_CC_OpenDevice(handle);
    if (MV_OK != nRet)
    {
       printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
       return -1;
    }

    // Start capturing
     nRet = MV_CC_StartGrabbing(handle);
    if (MV_OK != nRet)
    {
       printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
       return -1;
    }

    nRet= MV_CC_Display(handle, (void*)w);

    if (MV_OK != nRet)
    {
       printf("MV_CC_Displayfail! nRet [%x]\n", nRet);
       return -1;
    }
    printf("Display succeed\n");

    PressEnterToExit();

    // Stop capturing
     nRet = MV_CC_StopGrabbing(handle);
    if (MV_OK != nRet)
    {
       printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
       return -1;
    }

    // Close device
     nRet = MV_CC_CloseDevice(handle);
    if (MV_OK != nRet)
    {
       printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
       return -1;
    }

    // Destroy handle
     nRet = MV_CC_DestroyHandle(handle);
    if (MV_OK != nRet)
    {
       printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
       return -1;
    }

    printf("exit\n");

    return 0;
}
```

4.8.4 MV_CC_GetImageInfo

Get the basic image information.

API Definition

```
int MV_CC_GetImageInfo(
  void                     *handle,
  MV_IMAGE_BASIC_INFO      *pstInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstInfo

[OUT] Basic image information, see the structure ***MV_IMAGE_BASIC_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After the device is connected, call this API to get basic image information, including width, height, frame rate and resolution.
- This API is not supported by CameraLink device.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }
    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    else
    {
        unsigned int nAccessMode = MV_ACCESS_Exclusive;
        unsigned short nSwitchoverKey = 0;

        //Connect device
        nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
        if (MV_OK != nRet)
        {
            printf("error: OpenDevice fail [%x]\n", nRet);
            return;
        }
        //...other processing
    }

    //Get basic image information
    MV_IMAGE_BASIC_INFO mstruBasicInfo = {0};
    nRet = MV_CC_GetImageInfo(m_handle, &mstruBasicInfo);
    if (MV_OK != nRet)
    {
        printf("error: GetImageInfo fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.5 MV_CC_GetIntValue

Get the value of camera node with integer type.

API Definition

```
int MV_CC_GetIntValue(
  void             *handle,
  const char       *strKey,
  MVCC_INTVALUE    *pIntValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name, refer to MvCameraNode for details.

pIntValue

[OUT] Obtained node value

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After the device is connected, call this interface to get specified int nodes. For value of strKey, refer to MvCameraNode. The node values of **IInteger** can be obtained through this interface, **strKey** value corresponds to the Name column.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing
    //Get int parameter
    MVCC_INTVALUE struIntValue = {0};

    nRet = MV_CC_GetIntValue(m_handle, "Width", &struIntValue);
    if (MV_OK != nRet)
    {
        printf("error: GetIntValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.6 MV_CC_GetOneFrame

Get one frame data.

API Definition

```
int MV_CC_GetOneFrame(
  void                 *handle,
  unsigned char        *pData,
  unsigned int         nDataSize,
  MV_FRAME_OUT_INFO    *pFrameInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN] Buffer address used to save image data

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, see the structure ***MV_FRAME_OUT_INFO*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Before calling this API to get image data frame, call ***MV_CC_StartGrabbing*** to start image acquisition. The API actively gets frame data. The frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is discarded, and you can use ***MV_CC_GetOneFrameTimeout*** to replace it for getting chunk information and setting timeout.
- This API is not supported by CameraLink device.

See Also

MV_CC_StartGrabbing

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

#define MAX_BUF_SIZE    (1920*1080*3)

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Start acquiring image
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }

    unsigned int    nTestFrameSize = 0;
    unsigned char*  g_pFrameBuf = NULL;
    g_pFrameBuf = (unsigned char*)malloc(MAX_BUF_SIZE);

    MV_FRAME_OUT_INFO stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO));

    //The frequency of calling this interface should be controlled by upper
layer application according to frame rate.
    //The codes are for reference only. In practical application, it is
recommended to create a new thread for image acquisition and processing.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetOneFrame(m_handle,g_pFrameBuf, MAX_BUF_SIZE, &stInfo);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //...Image data processing
            nTestFrameSize++;
        }
    }

    //...other processing

    //Stop image acquiring
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.7 MV_CC_GetOneFrameEx

Get one frame data, supporting getting chunk information.

API Definition

```
int MV_CC_GetOneFrameEx(
  void                     *handle,
  unsigned char            *pData,
  unsigned int             nDataSize,
  MV_FRAME_OUT_INFO_EX     *pFrameInfo
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pData

[IN] Buffer address used to save image data

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, see the structure ***MV_FRAME_OUT_INFO_EX*** for details.

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Before calling this API to get image data frame, call ***MV_CC_StartGrabbing*** to start image acquisition. The API actively gets frame data. The frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is discarded, and you can use ***MV_CC_GetOneFrameTimeout*** to replace it for getting chunk information and setting timeout.
- This API is not supported by CameraLink device.

See Also

MV_CC_StartGrabbing

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

#define MAX_BUF_SIZE    (1920*1080*3)

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Start image acquisition
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }

    unsigned int    nTestFrameSize = 0;
    unsigned char*  g_pFrameBuf = NULL;
    g_pFrameBuf = (unsigned char*)malloc(MAX_BUF_SIZE);

    MV_FRAME_OUT_INFO_EX stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    //The frequency of calling this interface should be controlled by upper
layer application according to frame rate.
    //The codes are for reference only. In practical application, it is
recommended to create a new thread for image acquisition and processing.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetOneFrameEx(m_handle,g_pFrameBuf, MAX_BUF_SIZE, &stInfo);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //...Image data processing
            nTestFrameSize++;
        }
    }

    //...other processing

    //Stop image acquiring
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.8 MV_GIGE_ForceIp

Force the camera IP address.

API Definition

```
int MV_GIGE_ForceIp(
  void            *handle,
  unsigned int    nIP
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

nIP

[IN] IP address

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- This function is supported only by GigEVision cameras.
- After force the IP address, you should create the device handle again.
- If device is in DHCP status, after calling this API to force camera network parameter, the device will restart.
- This API is not supported, and the API ***MV_GIGE_ForceIpEx*** is recommended.

See Also

MV_GIGE_ForceIpEx

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], 
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Set device IP address
    unsigned int nIP = 0x0a0f0636; //10.15.6.54
    nRet = MV_GIGE_ForceIp(m_handle, nIP);
    if (MV_OK != nRet)
    {
        printf("error: ForceIp fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.9 MV_CC_ImageContrast

Adjust image contrast.

API Definition

```
int __stdcall MV_CC_ImageContrast(
  void                        *handle,
  MV_CC_CONTRAST_PARAM        *pstContrastParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstContrastParam

[IN] [OUT] Contrast parameter structure. See ***MV_CC_CONTRAST_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

This API is available for the device, which supports adjusting image contrast.

4.8.10 MV_CC_ImageSharpen

Adjust image sharpness.

API Definition

```
int __stdcall MV_CC_ImageSharpen(
  void                       *handle,
  MV_CC_SHARPEN_PARAM        *pstSharpenParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstSharpenParam

[IN] [OUT] Sharpness parameter structure. See ***MV_CC_SHARPEN_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

This API is available for the device which supports the function of adjusting image sharpness.

4.8.11 MV_CC_NoiseEstimate

Estimate the noise.

API Definition

```
int MV_CC_NoiseEstimate(
  void                         *handle,
  MV_CC_NOISE_ESTIMATE_PARAM    *pstNoiseEstimateParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstNoiseEstimateParam

[IN][OUT] Noise estimation parameters, see ***MV_CC_NOISE_ESTIMATE_PARAM*** for details

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

This API is available for the device, which supports noise estimation.

4.8.12 MV_CC_RegisterEventCallBack

Register Event callback function.

API Definition

```
int MV_CC_RegisterEventCallBack(
  void       *handle,
  cbEvent    fEventCallBack,
  void       *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

fEventCallBack

Event callback function, see the details below:

```
void(__stdcall* cbEvent)(
  unsigned int    nExternalEventId,
  void            *pUser
);
```

nExternalEventId

[OUT] External output event ID, see enumeration ***MV_GIGE_EVENT*** for details.

pUser

[OUT] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Set event callback through this interface. Get acquisition and exposure information in the callback function.
- This API is not recommended, the API ***MV_CC_RegisterEventCallBackEx*** is suggested.
- This API is supported by CameraLink device only for device offline event.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

//Exception callback function
void __stdcall EventCallBack(unsigned int nExternalEventId, void* pUser)
{
    switch(nUserDefinedId)
    {
    case MV_EVENT_ExposureEnd:
        printf("ExposureEnd!\n");
        break;
    case MV_EVENT_FrameStartOvertrigger:
        printf("FrameStartOvertrigger!\n");
        break;
    case MV_EVENT_AcquisitionStartOvertrigger:
        printf("AcquisitionStartOvertrigger!\n");
        break;
    case MV_EVENT_FrameStart:
        printf("FrameStart!\n");
        break;
    case MV_EVENT_AcquisitionStart:
        printf("AcquisitionStart!\n");
        break;
    case MV_EVENT_EventOverrun:
        printf("EventOverrun!\n");
        break;
    default:
        printf("no such eventId[%d]\n", nUserDefinedId);
    }
}

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Register Event callback function
    nRet = MV_CC_RegisterEventCallBack(handle, EventCallBack, NULL);
    if (MV_OK != nRet)
    {
        printf("error: RegisterEventCallBack fail [%x]\n", nRet);
        return;
    }

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.13 MV_CC_RegisterImageCallBack

Register image data callback function.

API Definition

```
int MV_CC_RegisterImageCallBack(
  void        *handle,
  cbOutput    fOutputCallBack,
  void        *pUser
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

fOutputCallBack

[IN] Image data callback function, see the details below:

```
void(__stdcall* cbOutput)(
  unsigned char        *pData,
  MV_FRAME_OUT_INFO    *pFrameInfo,
  void                 *pUser
);
```

pData

[OUT] Address of buffer that saves image data

pFrameInfo

[OUT] Obtained frame information, including width, height and pixel format. See the structure ***MV_FRAME_OUT_INFO*** for details

pUser

[OUT] User data

pUser

[IN] User data

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- After calling ***MV_CC_CreateHandle*** , call this API to set image data callback function.
- There are two available image data acquisition modes, and cannot be used together:
  1. Call MV_CC_RegisterImageCallBack to set image data callback function, and then call ***MV_CC_StartGrabbing*** to start acquiring. The acquired image data is returned in the configured callback function.
  2. Call ***MV_CC_StartGrabbing*** to start acquiring, and then call ***MV_CC_GetOneFrameTimeout*** repeatedly in application layer to get frame data of specified pixel format. When getting frame data, the frequency of calling this API should be controlled by upper layer application according to frame rate.
- This API is not supported by CameraLink device.
- This API is not recommended, suggest to replace it with ***MV_CC_RegisterImageCallBackEx***

See Also

MV_CC_StartGrabbing

MV_CC_RegisterImageCallBackEx

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO*
pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        // Add current system time at output
        char   szInfo[128] = {0};
        SYSTEMTIME sys;
        GetLocalTime( &sys );
        sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] :
GetOneFrame succeed, width[%d], height[%d]", sys.wYear, sys.wMonth,
            sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds,
pFrameInfo->nWidth, pFrameInfo->nHeight);

        printf("%s\n", szInfo);
    }
}

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Register data callback function
    nRet = MV_CC_RegisterImageCallBack(handle, ImageCallBack, NULL);
    if (MV_OK != nRet)
    {
        printf("error: RegisterImageCallBack fail [%x]\n", nRet);
        return;
    }

    //Connect device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Start acquiring image
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }

    //...other processing


    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.14 MV_CC_SaveImage

Convert original image data into picture format and saves in specified memory.

API Definition

```
int MV_CC_SaveImage(
  MV_SAVE_IMAGE_PARAM    *pSaveParam
);
```

Parameters

pSaveParam

[IN][OUT] Picture data input and output parameter

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

Through this API, convert the original images acquired by device into JPEG or BMP format and save in specified memory, and you can save the converted data as picture file. No sequence is required for calling this interface, original image data can be directly converted. First call ***MV_CC_GetOneFrame*** or ***MV_CC_RegisterImageCallBack*** to set callback function, and get one frame of image data, and then convert into other formats through this interface.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
```

```
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect to device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Starting acquiring image
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }

    //Get the size of one frame data
    MVCC_INTVALUE stIntvalue = {0};
    nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
    if (nRet != MV_OK)
    {
        printf("Get PayloadSize failed! nRet [%x]\n", nRet);
        return;
    }
    int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size +
reserved bytes (for SDK internal processing)

    unsigned int    nTestFrameSize = 0;
    unsigned char*  g_pFrameBuf = NULL;
    g_pFrameBuf = (unsigned char*)malloc(MAX_BUF_SIZE);

    MV_FRAME_OUT_INFO stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO));

    //The frequency of calling this interface should be controlled by upper
layer application according to frame rate.
    //The codes are for reference only. In practical application, it is
recommended to create a new thread for image acquisition and processing.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetOneFrame(m_handle,g_pFrameBuf, nBufSize, &stInfo);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //Picture data input and output parameters
            MV_SAVE_IMAGE_PARAM stParam;

            //Source data
            stParam.pData         = g_pFrameBuf;              //Original image
data
            stParam.nDataLen      = stFrameInfo.nFrameLen;    //Original image
data length
            stParam.enPixelType   = stFrameInfo.enPixelType;  //Original image
data pixel format
            stParam.nWidth        = stFrameInfo.nWidth;       //Image width
            stParam.nHeight       = stFrameInfo.nHeight;      //Image height

            //Target data
            stParam.enImageType   = MV_Image_Jpeg;            //Image type to
be saved, converting to JPEG format
            stParam.nBufferSize   = MAX_BUF_SIZE;             //Storage node
size
            unsigned char* pImage = (unsigned char*)malloc(MAX_BUF_SIZE);
            stParam.pImageBuffer  = pImage;                   //Output data
buffer, saving converted picture data
```

```
            nRet = MV_CC_SaveImage(&stParam);
            if(MV_OK != nRet)
            {
                break;
            }

            //Save converted picture data as file
            FILE* fp = fopen("image", "wb");
            fwrite(pImage, 1, stParam.nImageLen, fp);
            fclose(fp);
            free(pImage);

            //...other image data processing
            nTestFrameSize++;
        }
    }
    free(g_pFrameBuf);

    //...other processing

    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut down device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.15 MV_CC_SaveImageEx

Convert original image data into picture format and saves in specified memory, supporting setting JPEG encoding quality.

API Definition

```
int MV_CC_SaveImageEx(
  MV_SAVE_IMAGE_PARAM_EX    *pSaveParam
);
```

Parameters

pSaveParam

[IN][OUT] Picture data input and output parameter

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

- Through this interface, convert the original images acquired by device into JPEG or BMP format and save in specified memory, and you can save the converted data as picture file. No sequence is required for calling this interface, original image data can be directly converted. First call ***MV_CC_GetOneFrameTimeout*** or ***MV_CC_RegisterImageCallBackEx*** or ***MV_CC_GetImageBuffer*** to set callback function, and get one frame of image data, and then convert into other formats through this API.
- Suggest to replace this API with ***MV_CC_SaveImageEx2*** , frequently use of this API results in fragmentation.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Start acquiring image
    nRet = MV_CC_StartGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StartGrabbing fail [%x]\n", nRet);
        return;
    }

    //Get the size of one frame data
    MVCC_INTVALUE stIntvalue = {0};
    nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
    if (nRet != MV_OK)
    {
        printf("Get PayloadSize failed! nRet [%x]\n", nRet);
        return;
    }
    int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size +
reserved bytes (for SDK internal processing)

    unsigned int    nTestFrameSize = 0;
    unsigned char*  g_pFrameBuf = NULL;
    g_pFrameBuf = (unsigned char*)malloc(MAX_BUF_SIZE);

    MV_FRAME_OUT_INFO_EX stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    //The frequency of calling this interface should be controlled by upper
layer application according to frame rate.
    //The codes are for reference only. In practical application, it is
recommended to create a new thread for image acquisition and processing.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetOneFrameTimeout(m_handle,g_pFrameBuf, nBufSize,
&stInfo, 1000);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //Picture data input and output parameters
            MV_SAVE_IMAGE_PARAM_EX stParam;

            //Source data
            stParam.pData         = g_pFrameBuf;              //Original image
data
            stParam.nDataLen      = stFrameInfo.nFrameLen;    //Original image
data length
            stParam.enPixelType   = stFrameInfo.enPixelType;  //Original image
data pixel format
            stParam.nWidth        = stFrameInfo.nWidth;       //Image width
            stParam.nHeight       = stFrameInfo.nHeight;      //Image height
            stParam.nJpgQuality   = 70;                          //JPEG picture
encoding quality

            //Target data
            stParam.enImageType   = MV_Image_Jpeg;            //Image type to
be saved, converting to JPEG format
            stParam.nBufferSize   = MAX_BUF_SIZE;             //Storage node
size
            unsigned char* pImage = (unsigned char*)malloc(MAX_BUF_SIZE);
            stParam.pImageBuffer  = pImage;                   //Output data
buffer, saving converted picture data

            nRet = MV_CC_SaveImageEx(&stParam);
            if(MV_OK != nRet)
            {
                break;
            }

            //Save converted picture data as file
            FILE* fp = fopen("image", "wb");
            fwrite(pImage, 1, stParam.nImageLen, fp);
            fclose(fp);
            free(pImage);

            //...other image data processing
            nTestFrameSize++;
        }
    }
    free(g_pFrameBuf);

    //...other processing

    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.16 MV_CC_SetBayerCLUTParam

Enable/disable CLUT and set CLUT parameters of Bayer pattern.

API Definition

```
int __stdcall MV_CC_SetBayerCLUTParam(
  void                  *handle,
  MV_CC_CLUT_PARAM      *pstCLUTParam
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstCLUTParam

CLUT parameters structure. See ***MV_CC_CLUT_PARAM*** for details.

Return Value

Return *MV_OK* for success, and return ***Error Code*** for failure.

Remarks

- After enabling CLUT and setting CLUT parameters, the parameters will take effect when you call API ***MV_CC_ConvertPixelType*** or ***MV_CC_SaveImageEx2*** to convert the format of Bayer8/10/12/16 into RGB24/48, RGBA32/64, BGR24/48, or BGRA32/64.
- This API is available for the device, which supports the function.

4.8.17 MV_CC_SetIntValue

Set the value of camera node with integer type.

API Definition

```
int MV_CC_SetIntValue(
  void            *handle,
  const char      *strKey,
  unsigned int    nValue
);
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

strKey

[IN] Node name

nValue

[IN] Node Value

Return Value

Return *MV_OK(0)* on success, and return ***Error Code*** on failure.

Remarks

After the device is connected, call this API to set specified int nodes. For value of **strKey**, see "MvCameraNode.xlsx" sheet. The node values of **IInteger** can be set through this API, **strKey** value corresponds to the Name column.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;

    //Enumerate all devices corresponding to specified transport protocol
within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
        return;
    }

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return;
    }

    //Select the first found online device and create device handle
    int nDeviceIndex = 0;

    MV_CC_DEVICE_INFO m_stDevInfo = {0};
    memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex],
sizeof(MV_CC_DEVICE_INFO));

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

    if (MV_OK != nRet)
    {
        printf("error: CreateHandle fail [%x]\n", nRet);
        return;
    }

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...other processing

    //Set int parameters
    unsigned int nValue = 752;

    nRet = MV_CC_SetIntValue(m_handle, "Width", nValue);
    if (MV_OK != nRet)
    {
        printf("error: SetIntValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

4.8.18 MV_CC_SpatialDenoise

This API is used for spatial denoising.

API Definition

```
int MV_CC_SpatialDenoise(
  void                            *handle,
  MV_CC_SPATIAL_DENOISE_PARAM     *pstSpatialDenoiseParam
 )
```

Parameters

handle

[IN] Device handle, which is returned by ***MV_CC_CreateHandle*** or ***MV_CC_CreateHandleWithoutLog*** .

pstSpatialDenoiseParam

Spatial denoising parameters, see ***MV_CC_SPATIAL_DENOISE_PARAM*** for details.

Return Value

Return *MV_OK* on success, and return ***Error Code*** on failure.

Remarks

This API is available for the device, which supports spatial denoising.

Chapter 5 Data Structure and Enumeration

5.1 Data Structure

5.1.1 MV_ACTION_CMD_INFO

Command information structure

Structure Definition

```
struct{
  unsigned int            nDeviceKey;
  unsigned int            nGroupKey;
  unsigned int            nGroupMask;
  unsigned int            bActionTimeEnable;
  int64_t                 nActionTime;
  const char              *pBroadcastAddress;
  unsigned int            nTimeOut;
  unsigned int            nReserved[16];
}MV_ACTION_CMD_INFO_T;
```

Members

nDeviceKey

Device password

nGroupKey

Group key

nGroupMask

Group mask

bActionTimeEnable

Enable scheduled time or not: 1-enable

nActionTime

Scheduled time, it is valid only when **bActionTimeEnable** values "1", it is related to the clock rate.

pBroadcastAddress

Broadcast address

nTimeOut

ACK timeout, 0 indicates no need for acknowledgment

nReserved

Reserved.

5.1.2 MV_ACTION_CMD_RESULT

Structure about returned information of command

Structure Definition

```
struct{
  unsigned char            strDeviceAddress[12 + 3 + 1];
  int                      nStatus;
  unsigned int             nReserved[4];
}MV_ACTION_CMD_RESULT;
```

Members

strDeviceAddress

Device IP address

nStatus

Status code

nReserved

Reserved.

See Also

MV_ACTION_CMD_RESULT_LIST

5.1.3 MV_ACTION_CMD_RESULT_LIST

Structure about returned information list of command

Structure Definition

```
struct{
  unsigned int               nNumResults;
  MV_ACTION_CMD_RESULT       *pResults;
}MV_ACTION_CMD_RESULT_LIST;
```

Members

nNumResults

The number of returned results

pResults

Returned information of command, see the structure ***MV_ACTION_CMD_RESULT*** for details.

5.1.4 MV_ALL_MATCH_INFO

Structure about different matching type information

Structure Definition

```
struct{
  unsigned int    nType;
  void            *pInfo;
  unsigned int    nInfoSize;
}MV_ALL_MATCH_INFO;
```

Members

nType

Outputted information type

pInfo

Outputted information buffer, which is allocated by application layer.

nInfoSize

Information buffer size

Remarks

The outputted structure corresponding to **pInfo** are different according to different , see the table below:

nType Macro Definition	Value	Description	pInfo Structure
MV_MATCH_TYPE_NET_DETECT	0x00000001	Network flow and packet loss information	***MV_MATCH_INFO_NET_DETECT***
MV_MATCH_TYPE_USB_DETECT	0x00000002	Total byte number of USB3Vision camera received by host	***MV_MATCH_INFO_USB_DETECT***

Related API

MV_CC_GetAllMatchInfo

5.1.5 MV_CamL_DEV_INFO

Structure about CameraLink device information

Structure Definition

```
struct{
  unsigned char     chPortID[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chModelName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chFamilyName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chDeviceVersion[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chManufacturerName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chSerialNumber[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned int      nReserved[38];
}MV_CamL_DEV_INFO;
```

Members

chPortID

Port No.

chModelName

Device model name

chFamilyName

Device family name

chDeviceVersion

Version No.

chManufacturerName

Manufacturer name

chSerialNumber

Serial No.

nReserved

Reserved.

See Also

MV_CC_DEVICE_INFO

5.1.6 MV_CC_CCM_PARAM

Structure about Color Correction Parameters

Member	Data Type	Description
bCCMEnable	bool	Whether to enable color correction.
nCCMat	int	Color correction matrix, range: (-8192, 8192)
nRes	Array of unsigned int	Reserved.

5.1.7 MV_CC_CCM_PARAM_EX

CCM Parameter Structure

Member	Data Type	Description
bCCMEnable	bool	Whether to enable CCM.
nCCMat	int[]	Color correction matrix, range: (-65536,65536). The maximum length is 9 bytes.
nCCMScale	unsigned int	Quantitative scale (integer power of 2), the maximum value: 65536
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.8 MV_CC_CONTRAST_PARAM

Structure About Contrast Adjustment Parameters

Member	Data Type	Description
nWidth	unsigned int	The image width, the minimum value is 8.
nHeight	unsigned int	The image height, the minimum value is 8.
pSrcBuf	unsigned char*	The input data buffer
nSrcBufLen	unsigned int	The input data length
enPixelType	enum ***MvGvspPixelType***	The pixel format

Member	Data Type	Description
pDstBuf	unsigned char*	The output data buffer
nDstBufSize	unsigned int	The size of output buffer
nDstBufLen	unsigned int	The length of output data
nContrastFactor	unsigned int	The contract value, range: [1,10000]
nRes[8]	unsigned int	Reserved.

5.1.9 MV_CC_CLUT_PARAM

CLUT Parameter Structure

Member	Data Type	Description
bCLUTEnable	bool	Whether to enable CLUT.
nCLUTScale	unsigned int	Quantitative scale (integer power of 2). The maximum value: 65536, recommended value: 1024
nCLUTSize	unsigned int	CLUT size, default value: 17.
pCLUTBuf	unsigned char*	CLUT buffer
nCLUTBufLen	unsigned int	CLUT buffer length (nCLUTSize*nCLUTSize*nCLUTSize*sizeof(int)*3)
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.10 MV_CC_COLOR_CORRECT_PARAM

Structure about Color Correction Parameters

Member	Data Type	Description
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height

Member	Data Type	Description
pSrcBuf	unsigned char*	Input data buffer
nSrcBufLen	unsigned int	Input data length
enPixelType	enum ***MvGvspPixelType***	Pixel format
pDstBuf	unsigned char*	Output data buffer
nDstBufSize	unsigned int	Size of output buffer
nDstBufLen	unsigned int	Output data length
nImageBit	unsigned int	Image bit depth: 8, 10,12, or 16
stGammaParam	***MV_CC_GAMMA_PARAM***	Gamma parameters
stCCMParam	***MV_CC_CCM_PARAM_EX***	CCM parameters
stCLUTParam	***MV_CC_CLUT_PARAM***	CLUT parameters
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.11 MV_CC_CONTRAST_PARAM

Contrast Parameter Structure

Member	Data Type	Description
nWidth	unsigned int	Image width. Minimum value: 8.
nHeight	unsigned int	Image height. Minimum value: 8.
pSrcBuf	unsigned char*	Input data buffer
nSrcBufLen	unsigned int	Length of input data
enPixelType	enum ***MvGvspPixelType***	Pixel format
pDstBuf	unsigned char*	Output data buffer
nDstBufSize	unsigned int	Size of the provided output buffer
nDstBufLen	unsigned int	Length of output data

Member	Data Type	Description
nContrastFactor	unsigned int	Contrast. Range: [1,10000].
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.12 MV_CC_DEVICE_INFO

Device information structure.

Structure Definition

```
struct{
  unsigned short              nMajorVer;
  unsigned short              nMinorVer;
  unsigned int                nMacAddrHigh;
  unsigned int                nMacAddrLow;
  unsigned int                nTLayerType;
  unsigned int                nReserved[4];
  union
  {
      MV_GIGE_DEVICE_INFO     stGigEInfo;
      MV_USB3_DEVICE_INFO     stUsb3VInfo;
      MV_CamL_DEV_INFO        stCamLInfo;
  }SpecialInfo;
}MV_CC_DEVICE_INFO;
```

Members

nMajorVer

Major version No.

nMinorVer

Minor version No.

nMacAddrHigh

High MAC address

nMacAddrLow

Low MAC address

nTLayerType

Transport layer type, see the definitions in the table below.

Macro Definition	Value	Description
MV_UNKNOW_DEVICE	0x00000000	Unknown device type
MV_GIGE_DEVICE	0x00000001	GigE device
MV_1394_DEVICE	0x00000002	1394-a/b device
MV_USB_DEVICE	0x00000004	USB3.0 device
MV_CAMERALINK_DEVICE	0x00000008	CameraLink device

nReserved

Reserved.

stGigEInfo

GIGE device information, it is valid when **nTLayerType** is "MV_GIGE_DEVICE", (different transport layers corresponds to different device information). See the structure ***MV_GIGE_DEVICE_INFO*** for details.

stUsb3VInfo

USB device information, it is valid when **nTLayerType** is "MV_USB_DEVICE" (different transport layers corresponds to different device information). See the structure ***MV_USB3_DEVICE_INFO*** for details.

stCamLInfo

CameraLink device information, it is valid when **nTLayerType** is "MV_CAMERALINK_DEVICE" (different transport layers corresponds to different device information). See the structure ***MV_CamL_DEV_INFO*** for details.

See Also

MV_CC_DEVICE_INFO_LIST

Related API

MV_CC_CreateHandle
MV_CC_IsDeviceAccessible
MV_CC_GetDeviceInfo

5.1.13 MV_CC_DEVICE_INFO_LIST

Structure about device information list

Structure Definition

```
struct{
  unsigned int          nDeviceNum;
  MV_CC_DEVICE_INFO    *pDeviceInfo[MV_MAX_DEVICE_NUM/*256*/];
}MV_CC_DEVICE_INFO_LIST;
```

Members

nDeviceNum

The number of online devices

pDeviceInfo

Online device information, each array indicates a device, and up to 256 devices are supported. See the structure ***MV_CC_DEVICE_INFO*** for details.

Related API

MV_CC_EnumDevices

5.1.14 MV_CC_FILE_ACCESS

File information structure

Structure Definition

```
struct{
  const char       *pDevFileName;
  const char       *pUserFileName;
  unsigned int     nReserved[32];
}MV_CC_FILE_ACCESS;
```

Members

pDevFileName

Device file name

pUserFileName

User file name

nReserved

Reserved.

5.1.15 MV_CC_FILE_ACCESS_PROGRESS

Structure about parameters loading progress

Structure Definition

```
struct{
  int64_t          nCompleted;
  int64_t          nTotal;
  unsigned int     nRes[8];
}MV_CC_FILE_ACCESS_PROGRESS;
```

Members

nCompleted

Completed size

nTotal

Total size

nRes

Reserved.

5.1.16 MV_CC_FLIP_IMAGE_PARAM

Structure about Image Flipping

Member	Data Type	Description
enPixelType	enum ***MvGvspPixelType***	Pixel format
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height
pSrcData	public IntPtr	Buffer of input data
nSrcDataLen	unsigned int	Size of input data
pDstBuf	public IntPtr	Buffer of output data
nDstBufLen	unsigned int	Size of output data
nDstBufSize	unsigned int	Size of the output buffer
enFlipType	***MV_IMG_FLIP_TYPE***	Flip type
nRes	Array of unsigned int	Reserved.

5.1.17 MV_CC_FRAME_SPEC_INFO

Structure about Watermark Information

Member	Data Type	Description
nSecondCount	unsigned int	Seconds
nCycleCount	unsigned int	The number of cycles
nCycleOffset	unsigned int	Cycle offset
fGain	float	Gain
fExposureTime	unsigned int	Exposure Time
nAverageBrightness	unsigned int	Average brightness
nRed	unsigned int	Red
nGreen	unsigned int	Green
nBlue	unsigned int	Blue
nFrameCounter	unsigned int	The total number of frames
nTriggerIndex	unsigned int	Trigger index
nInput	unsigned int	Input
nOutput	unsigned int	Output
nOffsetX	unsigned short	Horizontal offset
nOffsetY	unsigned short	Vertical offset
nFrameWidth	unsigned short	Watermark width
nFrameHeight	unsigned short	Watermark height
nReserved	unsigned int	Reserved.

5.1.18 MV_CC_GAMMA_PARAM

Gamma Parameter Structure

Member	Data Type	Description
enGammaType	***MV_CC_GAMMA_TYPE***	Gamma type
fGammaValue	float	Gamma value, range: [0.1,4.0]
pGammaCurveBuf	unsigned char*	Gamma curve buffer

Member	Data Type	Description
nGammaCurveBufLen	unsigned int	Size of gamma curve
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.19 MV_CC_HB_DECODE_PARAM

Structure about Lossless Decoding Parameters

Member	Data Type	Description
pSrcBuf	unsigned char*	Buffer of input data
nSrcLen	unsigned int	Size of input data
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height
pDstBuf	unsigned char*	Buffer of output data
nDstBufLen	unsigned int	Size of output data
nDstBufSize	unsigned int	Size of the output buffer
enDstPixelType	***MvGvspPixelType***	Pixel format
stFrameSpecInfo	***MV_CC_FRAME_SPEC_INFO***	Watermark information
nRes	Array of unsigned int	Reserved.

5.1.20 MV_CC_INPUT_FRAME_INFO

Structure about Video Data

Member	Data Type	Description
pData	unsigned char*	Image data pointer
nDataLen	unsigned int	Image size
nRes	Array of unsigned int	Reserved.

5.1.21 MV_CC_LSC_CALIB_PARAM

Structure about LSC Calibration Parameters

Member	Data Type	Description
nWidth	unsigned int	Image width. Range: [16,65535].
nHeight	unsigned int	Image height. Range: [16,65535].
enPixelType	enum ***MvGvspPixelType***	Pixel format
pSrcBuf	unsigned char*	Input data buffer
nSrcBufLen	unsigned int	Length of input data
pCalibBuf	unsigned char*	Output calibration data file buffer
nCalibBufSize	unsigned int	Size of the provided calibration data file buffer
nCalibBufLen	unsigned int	Length of calibration data file buffer
nSecNumW	unsigned int	Number of width sections
nSecNumH	unsigned int	Number of height sections
nPadCoef	unsigned int	Padding coefficient. Range: [1,5].
nCalibMethod	unsigned int	Calibration method: 0 (use the center as the reference) 1 (use the brightest area as the reference) 2 (adjust to the target brightness)
nTargetGray	unsigned int	Target brightness: 8bit. Range: [0,255] 10bit. Range: [0,1023] 12bit. Range: [0,4095]

Member	Data Type	Description
		16bit. Range: [0,65535]
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.22 MV_CC_LSC_CORRECT_PARAM

Structure about LSC Correction Parameters

Member	Data Type	Description
nWidth	unsigned int	Image width. Range: [16,65535].
nHeight	unsigned int	Image height. Range: [16,65535].
enPixelType	enum ***MvGvspPixelType***	Pixel format
pSrcBuf	unsigned char*	Input data buffer
nSrcBufLen	unsigned int	Length of input data
pDstBuf	unsigned char*	Output data buffer
nDstBufSize	unsigned int	Size of the provided output buffer
nDstBufLen	unsigned int	Length of output data
pCalibBuf	unsigned char*	Input calibration data file buffer
nCalibBufLen	unsigned int	Length of input calibration data file buffer
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.23 MV_CC_NOISE_ESTIMATE_PARAM

Structure about Noise Estimation Parameters

Member	Data Type	Description
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height
enPixelType	enum ***MvGvspPixelType***	Pixel format
pSrcBuf	unsigned char*	Buffer of input data
nSrcBufLen	unsigned int	Input data size
pstROIRect	***MV_CC_RECT_I*** *	ROI information
nROINum	unsigned int	The number of ROIs
nNoiseThreshold	unsigned int	Noise threshold, range: [0,4095] **Note** This node is the Bayer noise estimation parameter, and it is invalid for MONO8/ RGB.
pNoiseProfile	unsigned char*	Output noise feature **Note** This node is the Bayer noise estimation parameter, and it is invalid for MONO8/ RGB.
nNoiseProfileSize	unsigned int	Output buffer size **Note** This node is the Bayer noise estimation parameter, and it is invalid for MONO8/ RGB.
nNoiseProfileLen	unsigned int	Output noise feature length **Note** This node is the Bayer noise estimation parameter, and it is invalid for MONO8/ RGB.
nRes	unsigned int	Reserved. Maximum: 8 bytes.

5.1.24 MV_CC_PIXEL_CONVERT_PARAM

Structure about image conversion parameters

Structure Definition

```
struct{
  unsigned short          nWidth;
  unsigned short          nHeight;
  MvGvspPixelType         enSrcPixelType;
  unsigned char           *pSrcData;
  unsigned int            nSrcDataLen;
  MvGvspPixelType         enDstPixelType;
  unsigned char           *pDstBuffer;
  unsigned int            nDstLen;
  unsigned int            nDstBufferSize;
  unsigned int            nRes[4];
}MV_CC_PIXEL_CONVERT_PARAM;
```

Members

nWidth

Image width

nHeight

Image Height

enSrcPixelType

Source pixel format, see the enumeration type ***MvGvspPixelType*** for details.

pSrcData

Original image data

nSrcDataLen

Length of original image data

enDstPixelType

Target pixel format, see the enumeration type ***MvGvspPixelType*** for details.

pDstBuffer

Outputted data buffer, used to save the converted target data.

nDstLen

Converted target data length

nDstBufferSize

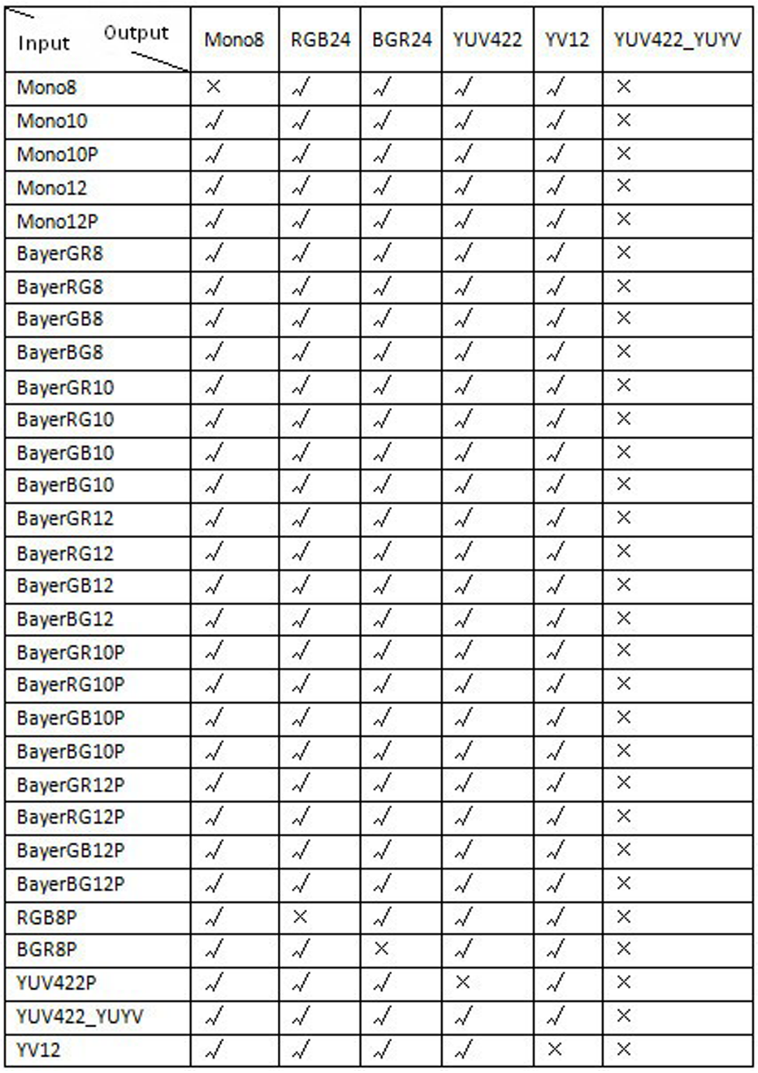
Outputted data buffer size

nRes

Reserved.

Remarks

The supported inputted and outputted pixel formats after conversion are shown below:



Input Output	Mono8	RGB24	BGR24	YUV422	YV12	YUV422_YUYV
Mono8	×	✓	✓	✓	✓	×
Mono10	✓	✓	✓	✓	✓	×
Mono10P	✓	✓	✓	✓	✓	×
Mono12	✓	✓	✓	✓	✓	×
Mono12P	✓	✓	✓	✓	✓	×
BayerGR8	✓	✓	✓	✓	✓	×
BayerRG8	✓	✓	✓	✓	✓	×
BayerGB8	✓	✓	✓	✓	✓	×
BayerBG8	✓	✓	✓	✓	✓	×
BayerGR10	✓	✓	✓	✓	✓	×
BayerRG10	✓	✓	✓	✓	✓	×
BayerGB10	✓	✓	✓	✓	✓	×
BayerBG10	✓	✓	✓	✓	✓	×
BayerGR12	✓	✓	✓	✓	✓	×
BayerRG12	✓	✓	✓	✓	✓	×
BayerGB12	✓	✓	✓	✓	✓	×
BayerBG12	✓	✓	✓	✓	✓	×
BayerGR10P	✓	✓	✓	✓	✓	×
BayerRG10P	✓	✓	✓	✓	✓	×
BayerGB10P	✓	✓	✓	✓	✓	×
BayerBG10P	✓	✓	✓	✓	✓	×
BayerGR12P	✓	✓	✓	✓	✓	×
BayerRG12P	✓	✓	✓	✓	✓	×
BayerGB12P	✓	✓	✓	✓	✓	×
BayerBG12P	✓	✓	✓	✓	✓	×
RGB8P	✓	×	✓	✓	✓	×
BGR8P	✓	✓	×	✓	✓	×
YUV422P	✓	✓	✓	×	✓	×
YUV422_YUYV	✓	✓	✓	✓	✓	×
YV12	✓	✓	✓	✓	×	×

Related API

MV_CC_ConvertPixelType

5.1.25 MV_CC_RECORD_PARAM

Video parameters structure

Structure Definition

```
struct{
  enum MvGvspPixelType              enPixelType;
  unsigned short                    nWidth;
  unsigned short                    nHeight;
  float                             fFrameRate;
  unsigned int                      nBitRate;
  MV_RECORD_FORMAT_TYPE             enRecordFmtType;
  char                              *strFilePath;
  unsigned int                      nRes[8];
}MV_CC_RECORD_PARAM;
```

Members

enPixelType

Pixel format

nWidth

Image width, it should be multiples of eight

nHeight

Image height, it should be multiples of eight

fFrameRate

Frame rate, unit: fps, from 1/16 to 120

nBitRate

Bit rate, unit: kbps, from 128kbps to 16Mbps

enRecordFmtType

Video format, see the structure ***MV_RECORD_FORMAT_TYPE*** for details.

strFilePath

Video storage path

nRes

Reserved.

5.1.26 MV_CC_RECT_I

Structure about ROI Rectangle Information

Member	Data Type	Description
nX	unsigned int	X-coordinate of rectangle upper left corner
nY	unsigned int	Y-coordinate of rectangle upper left corner
nWidth	unsigned int	Rectangle width
nHeight	unsigned int	Rectangle height

5.1.27 MV_CC_ROTATE_IMAGE_PARAM

Structure about Image Rotation

Member	Data Type	Description
public **enPixelType**	enum ***MvGvspPixelType***	Pixel format
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height
pSrcData	unsigned char*	Buffer of input data
nSrcDataLen	unsigned int	Size of input data
pDstBuf	unsigned char*	Buffer of output data
nDstBufLen	unsigned int	Size of output data
nDstBufSize	unsigned int	Size of the output buffer
enRotationAngle	***MV_IMG_ROTATION_ANGLE***	Rotation angle
nRes	Array of unsigned int	Reserved.

5.1.28 MV_CC_SHARPEN_PARAM

Sharpness Parameter Structure

Member	Data Type	Description
nWidth	unsigned int	Image width. Minimum value: 8.
nHeight	unsigned int	Image height. Minimum value: 8.
pSrcBuf	unsigned char*	Input data buffer
nSrcBufLen	unsigned int	Length of input data
enPixelType	enum ***MvGvspPixelType***	Pixel format
pDstBuf	unsigned char*	Output data buffer
nDstBufSize	unsigned int	Size of the provided output buffer
nDstBufLen	unsigned int	Length of output data
nSharpenAmount	unsigned int	Sharpness. Range: [0,500].
nSharpenRadius	unsigned int	Radius of the adjustment area. Range: [1,21].
nSharpenThreshold	unsigned int	Sharpness threshold. Range: [0,255].
nRes	unsigned int[]	Reserved. The maximum length is 8 bytes.

5.1.29 MV_CC_SPATIAL_DENOISE_PARAM

Structure about Spatial Denoising Parameters

Member	Data Type	Description
nWidth	unsigned int	Image width
nHeight	unsigned int	Image height
enPixelType	enum ***MvGvspPixelType***	Pixel format
pSrcBuf	unsigned char*	Buffer of input data
nSrcBufLen	unsigned int	Input data size

Member	Data Type	Description
pDstBuf	unsigned char*	Output denoised data
nDstBufSize	unsigned int	Buffer size of output data
nDstBufLen	unsigned int	Output denoised data length
pNoiseProfile	unsigned char*	Input noise features
nNoiseProfileLen	unsigned int	Input noise features length
nBayerDenoiseStrength	unsigned int	Denoising strength, range: [0,100]. **Note** This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nBayerSharpenStrength	unsigned int	Sharpening strength, range: [0,32]. **Note** This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nBayerNoiseCorrect	unsigned int	Noise correction factor, range: [0,1280]. **Note** This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nNoiseCorrectLum	unsigned int	Luminance correction factor, range: [1,2000]. **Note** This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nNoiseCorrectChrom	unsigned int	Hue correction factor, range: [1,2000].

Member	Data Type	Description
		Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthLum	unsigned int	Luminance denoising strength, range: [0,100]. **Note** This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthChrom	unsigned int	Hue denoising strength, range: [0,100]. **Note** This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthSharpen	unsigned int	Sharpening strength, range: [1,1000]. **Note** This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nRes	unsigned int	Reserved. Maximum: 8 bytes.

5.1.30 MV_DISPLAY_FRAME_INFO

Image displaying structure

Structure Definition

```
struct{
  void                    *hWnd;
  unsigned char           *pData;
  unsigned int            nDataLen;
  unsigned short          nWidth;
  unsigned short          nHeight;
  MvGvspPixelType         enPixelType;
  unsigned int            nRes[4];
}MV_DISPLAY_FRAME_INFO;
```

Members

hWnd

Window handle

pData

Image data

nDataLen

Image data size

nWidth

Image width

nHeight

Image height

enPixelType

Original image pixel format, see the enumeration type ***MvGvspPixelType*** for details.

nRes

Reserved.

Related API

MV_CC_DisplayOneFrame

5.1.31 MV_EVENT_OUT_INFO

Output event information structure

Structure Definition

```
struct{
  char                 EventName[MAX_EVENT_NAME_SIZE/*128*/];
  unsigned short       nEventID;
  unsigned short       nStreamChannel;
  unsigned int         nBlockIdHigh;
  unsigned int         nBlockIdLow;
  unsigned int         nTimestampHigh;
  unsigned int         nTimestampLow;
  void                 *pEventData;
  unsigned int         nEventDataSize;
  unsigned int        nReserved[16];
}MV_EVENT_OUT_INFO;
```

Members

EventName

Event name

nEventID

Event ID

nStreamChannel

Stream channel ID

nBlockIdHigh

High bit of frame number

nBlockIdLow

Low bit of frame number

nTimestampHigh

Timestamp high bit

nTimestampLow

Timestamp low bit

pEventData

Event data

nEventDataSize

Event data size

nReserved

Reserved

5.1.32 MV_FRAME_OUT

Structure about picture data and picture information

Structure Definition

```
struct{
  unsigned char           *pBufAddr;
  MV_FRAME_OUT_INFO_EX    stFrameInfo;
  unsigned int            nRes[16];
}MV_FRAME_OUT;
```

Members

pBufAddr

Picture data

stFrameInfo

Picture information, see the structure ***MV_FRAME_OUT_INFO_EX*** for details.

nRes

Reserved.

5.1.33 MV_FRAME_OUT_INFO

Output frame information structure

Structure Definition

```
struct{
  unsigned short     nWidth;
  unsigned short     nHeight;
  MvGvspPixelType    enPixelType;
  unsigned int       nFrameNum;
  unsigned int       nDevTimeStampHigh;
  unsigned int       nDevTimeStampLow;
  unsigned int       nReserved0;
  int64_t            nHostTimeStamp;
  unsigned int       nFrameLen;
  unsigned int       nReserved[3];
}MV_FRAME_OUT_INFO;
```

Members

nWidth

Image width

nHeight

Image height

enPixelType

Pixel format, see the enumeration ***MvGvspPixelType*** for details.

nFrameNum

Frame number

nDevTimeStampHigh

Timestamp generated by camera, high-order 32-bits

nDevTimeStampLow

Timestamp generated by camera, low-order 32-bits

nReserved0

Reserved (align 8 bytes)

nHostTimeStamp

Timestamp generated by host

nFrameLen

Frame length

nReserved

Reserved.

5.1.34 MV_FRAME_OUT_INFO_EX

Output frame information structure

Structure Definition

```
struct{
  unsigned short     nWidth;
  unsigned short     nHeight;
  MvGvspPixelType    enPixelType;
  unsigned int       nFrameNum;
  unsigned int       nDevTimeStampHigh;
  unsigned int       nDevTimeStampLow;
  unsigned int       nReserved0;
  int64              nHostTimeStamp;
  unsigned int       nFrameLen;
  unsigned int       nSecondCount;
  unsigned int       nCycleCount;
  unsigned int       nCycleOffset;
  float              fGain;
  float              fExposureTime;
  unsigned int       nAverageBrightness;
  unsigned int       nRed;
  unsigned int       nGreen;
  unsigned int       nBlue;
  unsigned int       nFrameCounter;
  unsigned int       nTriggerIndex;
  unsigned int       nInput;
  unsigned int       nOutput;
  unsigned int       nLostPacket;
  unsigned short     nOffsetX;
  unsigned short     nOffsetY;
  unsigned int       nReserved[41];
}MV_FRAME_OUT_INFO_EX;
```

Members

nWidth

Image width

nHeight

Image height

enPixelType

Pixel format, see the enumeration ***MvGvspPixelType*** for details.

nFrameNum

Frame number

nDevTimeStampHigh

Timestamp generated by camera, high-order 32-bits

nDevTimeStampLow

Timestamp generated by camera, low-order 32-bits

nReserved0

Reserved (align 8 bytes)

nHostTimeStamp

Timestamp generated by host

nFrameLen

Frame length

nSecondCount

Seconds, increase by second

nCycleCount

Clock period counting, increase by 125 us, reset in every 1 second.

nCycleOffset

Clock period offset, reset in every 125 us.

fGain

Gain

fExposureTime

Exposure time

nAverageBrightness

Average brightness

nRed

WB red

nGreen

WB green

nBlue

WB blue

nFrameCounter

The number of frames

nTriggerIndex

Trigger counting

nInput

Line input

nOutput

Line output

nLostPacket

The number of lost packets

nOffsetX

X value of ROI area offset

nOffsetY

Y value of ROI area offset

nReserved

Reserved.

5.1.35 MV_GENTL_DEV_INFO

Structure about information of device enumerated via GenTL

Structure Definition

```
struct{
  unsigned char      chInterfaceID[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chDeviceID[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chVendorName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chModelName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chTLType[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chDisplayName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chUserDefinedName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chSerialNumber[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char      chDeviceVersion[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned int       nCtiIndex;
  unsigned int       nReserved[8];
}MV_GENTL_DEV_INFO;
```

Members

chInterfaceID

Interface ID

chDeviceID

Device ID

chVendorName

Vendor name

chModelName

Model name

chTLType

Transport layer type

chDisplayName

Device name

chUserDefinedName

User defined name

chSerialNumber

Serial number

chDeviceVersion

Device version

nCtiIndex

CTI file index

nReserved

Reserved.

5.1.36 MV_GENTL_DEV_INFO_LIST

Structure about list of devices enumerated via GenTL

Structure Definition

```
struct{
  unsigned int          nDeviceNum;
  MV_GENTL_DEV_INFO    *pDeviceInfo[MV_MAX_GENTL_DEV_NUM/*256*/];
}MV_GENTL_DEV_INFO_LIST;
```

Members

nDeviceNum

The number of online devices

pDeviceInfo

Device information, see the structure ***MV_GENTL_DEV_INFO*** for details. Up to 256 devices are supported.

5.1.37 MV_GENTL_IF_INFO

Structure about information of interface enumerated via GenTL

Structure Definition

```
struct{
  unsigned char          chInterfaceID[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char          chTLType[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char          chDisplayName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned int           nCtiIndex;
  unsigned int           nReserved[8];
}MV_GENTL_IF_INFO;
```

Members

chInterfaceID

Interface ID

chTLType

Transport layer type

chDisplayName

Device name

nCtiIndex

CTI file index

nReserved

Reserved.

5.1.38 MV_GENTL_IF_INFO_LIST

Structure about list of interfaces enumerated via GenTL

Structure Definition

```
struct{
  unsigned int           nInterfaceNum;
  MV_GENTL_IF_INFO       *pIFInfo[MV_MAX_GENTL_IF_NUM/*256*/];
}MV_GENTL_IF_INFO_LIST;
```

Members

nInterfaceNum

The number of interfaces.

pIFInfo

The interface information, see ***MV_GENTL_IF_INFO*** for details. Up to 256 interfaces are supported.

5.1.39 MV_GIGE_DEVICE_INFO

Structure about GIGE device information

Structure Definition

```
struct{
  unsigned int     nIpCfgOption;
  unsigned int     nIpCfgCurrent;
  unsigned int     nCurrentIp;
  unsigned int     nCurrentSubNetMask;
  unsigned int     nDefultGateWay;
  unsigned char    chManufacturerName[32];
  unsigned char    chModelName[32];
  unsigned char    chDeviceVersion[32];
  unsigned char    chManufacturerSpecificInfo[48];
  unsigned char    chSerialNumber[16];
  unsigned char    chUserDefinedName[16];
  unsigned int     nNetExport;
  unsigned int     nReserved[4];
}MV_GIGE_DEVICE_INFO;
```

Members

nIpCfgOption

IP configuration options

nIpCfgCurrent

Current IP configuration

nCurrentIp

Current device IP

nCurrentSubNetMask

Current subnet mask

nDefultGateWay

Default gateway

chManufacturerName

Manufacturer name

chModelName

Model name

chDeviceVersion

Device version

chManufacturerSpecificInfo

Manufacturing batch information

chSerialNumber

Serial No.

chUserDefinedName

Custom name

nNetExport

Network port IP address

nReserved

Reserved.

See Also

MV_CC_DEVICE_INFO

5.1.40 MV_IMAGE_BASIC_INFO

Image basic information structure

Structure Definition

```
struct{
  unsigned short     nWidthValue;
  unsigned short     nWidthMin;
  unsigned short     nWidthMax;
  unsigned short     nWidthInc;
  unsigned short     nHeightValue;
  unsigned short     nHeightMin;
  unsigned short     nHeightMax;
  unsigned short     nHeightInc;
  float              fFrameRateValue;
  float              fFrameRateMin;
  float              fFrameRateMax;
  MvGvspPixelType    enPixelType;
  unsigned int       nSupportedPixelFmtNum;
  MvGvspPixelType    enPixelList[MV_MAX_XML_SYMBOLIC_NUM/*64*/];
  unsigned int       nReserved[8];
}MV_IMAGE_BASIC_INFO;
```

Members

nWidthValue

Image width

nWidthMin

Minimum image width

nWidthMax

Maximum image width

nWidthInc

Step-by-step value of image width

nHeightValue

Image height

nHeightMin

Minimum image height

nHeightMax

Maximum image height

nHeightInc

Step-by-step value of image height

fFrameRateValue

Frame rate

fFrameRateMin

Minimum frame rate

fFrameRateMax

Maximum frame rate

enPixelType

Current pixel format, see the enumeration ***MvGvspPixelType*** for details.

nSupportedPixelFmtNum

Supported pixel format types

enPixelList

Supported pixel format list, see the enumeration ***MvGvspPixelType*** for details.

nReserved

Reserved.

5.1.41 MV_MATCH_INFO_NET_DETECT

Structure about network flow and packet loss information

Structure Definition

```
struct{
  int64           nReviceDataSize;
  int64           nLostPacketCount;
  unsigned int    nLostFrameCount;
  unsigned int    nNetRecvFrameCount;
  int64           nRequestResendPacketCount;
  int64           nResendPacketCount;
}MV_MATCH_INFO_NET_DETECT;
```

Members

nReviceDataSize

Received data size (data statistics between StartGrabbing and StopGrabbing)

nLostPacketCount

The number of lost packets

nLostFrameCount

The number of lost frames

nNetRecvFrameCount

The number of received frames

nRequestResendPacketCount

The number of packets, which are requested to resend

nResendPacketCount

The number of resent packets

See Also

MV_ALL_MATCH_INFO

5.1.42 MV_MATCH_INFO_USB_DETECT

Structure about the total number of bytes host received from USB3 vision camera

Structure Definition

```
struct{
  int64            nReceiveDataSize;
  unsigned int     nReceivedFrameCount;
  unsigned int     nErrorFrameCount;
  unsigned int     nReserved[2];
}MV_MATCH_INFO_USB_DETECT
```

Members

nReceiveDataSize

Received data size (data statistics between OpenDevicce and CloseDevice)

nReceivedFrameCount

The number of received frames

nErrorFrameCount

The number of error frames

nReserved

Reserved.

5.1.43 MV_NETTRANS_INFO

Network transport information structure

Structure Definition

```
struct{
  int64                nReviceDataSize;
  int                  nThrowFrameCount;
  unsigned int         nNetRecvFrameCount;
  int64                nRequestResendPacketCount;
  int64                nResendPacketCount;
}MV_NETTRANS_INFO;
```

Members

nReviceDataSize

Received data size

nThrowFrameCount

The number of lost frames

nNetRecvFrameCount

The number of received frames

nRequestResendPacketCount

The number of packets, which request for resend

nResendPacketCount

The number of resent packets

5.1.44 MV_OUTPUT_IMAGE_INFO

Structure about Image List After Reconstruction

Member	Data Type	Description
nWidth	unsigned int	The source image width
nHeight	unsigned int	The source image height
enPixelType	enum ***MvGvspPixelType***	The pixel format
pBuf	unsigned char*	The output data buffer
nBufLen	unsigned int	The output data length
nBufSize	unsigned int	The size of provided output buffer
nRes[8]	unsigned int	Reserved.

5.1.45 MV_RECORD_FORMAT_TYPE

Video format structure

Structure Definition

```
struct{
  MV_FormatType_Undefined      = 0,
  MV_FormatType_AVI            = 1,
}MV_RECORD_FORMAT_TYPE;
```

Members

MV_FormatType_Undefined

Undefined format

MV_FormatType_AVI

AVI format

5.1.46 MV_RECONSTRUCT_IMAGE_PARAM

Structure About Image Reconstruction Parameters

Member	Data Type	Description
nWidth	unsigned int	The source image width
nHeight	unsigned int	The source image height
enPixelType	enum ***MvGvspPixelType***	The pixel format
pSrcData	unsigned char*	The input data buffer
nSrcDataLen	unsigned int	The length of input data
nExposureNum	unsigned int	The number of exposures, range: (1,8]
enRestructureMode	***MV_IMAGE_RECONSTRUCTION_METHOD***	The image reconstruction mode
stDstBufList	***MV_OUTPUT_IMAGE_INFO***	The information about output data buffer
nRes[4]	unsigned int	Reserved.

5.1.47 MV_SAVE_IMAGE_PARAM

Structure about parameters of converting picture format

Structure Definition

```
struct{
  unsigned char         *pData;
  unsigned int          nDataLen;
  MvGvspPixelType       enPixelType;
  unsigned short        nWidth;
  unsigned short        nHeight;
  unsigned char         *pImageBuffer;
  unsigned int          nImageLen;
  unsigned int          nBufferSize;
  MV_SAVE_IAMGE_TYPE    enImageType;
}MV_SAVE_IMAGE_PARAM;
```

Members

pData

Original image data

nDataLen

Original image data length

enPixelType

Pixel format of original image data, see the enumeration ***MvGvspPixelType*** for details.

nWidth

Image width

nHeight

Image height

pImageBuffer

Output data buffer, used for storing converted picture

nImageLen

Converted picture data length

nBufferSize

The size of output data buffer

enImageType

Output picture format, see the enumeration ***MV_SAVE_IAMGE_TYPE*** for details.

5.1.48 MV_SAVE_IMAGE_PARAM_EX

Structure about parameters of converting picture format

Structure Definition

```
struct{
  unsigned char         *pData;
  unsigned int          nDataLen;
  MvGvspPixelType       enPixelType;
  unsigned short        nWidth;
  unsigned short        nHeight;
  unsigned char         *pImageBuffer;
  unsigned int          nImageLen;
  unsigned int          nBufferSize;
  MV_SAVE_IAMGE_TYPE    enImageType;
  unsigned int          nJpgQuality;
  unsigned int          nReserved[4];
}MV_SAVE_IMAGE_PARAM_EX;
```

Members

pData

Original image data

nDataLen

Original image data length

enPixelType

Pixel format of original image data, see the enumeration ***MvGvspPixelType*** for details.

nWidth

Image width

nHeight

Image height

pImageBuffer

Output data buffer, used for storing converted picture data

nImageLen

Converted picture data length

nBufferSize

The size of output data buffer

enImageType

Output picture format, see the enumeration ***MV_SAVE_IAMGE_TYPE*** for details.

nJpgQuality

Encoding quality, range: (50,99]

nReserved

Reserved.

5.1.49 MV_SAVE_IMG_TO_FILE_PARAM

Structure about image saving parameters.

Structure Definition

```
struct{
  enum MvGvspPixelType       enPixelType;
  unsigned char*             pData;
  unsigned int               nDataLen;
  unsigned short             nWidth;
  unsigned short             nHeight;
  enum MV_SAVE_IAMGE_TYPE    enImageType;
  unsigned int               nQuality;
  char                       pImagePath[256];
  int                        iMethodValue;
  unsigned int               nReserved[8];
}MV_SAVE_IMG_TO_FILE_PARAM;
```

Members

enPixelType

The pixel format of the input data, see the enumeration definition ***MvGvspPixelType*** for details.

pData

Input data buffer

nDataLen

Input data size

nWidth

Image width

nHeigh

Image height

enImageType

Input image format, see the enumeration definition ***MV_SAVE_IAMGE_TYPE*** for details.

nQuality

JPG encoding quality: (50-99]; PNG encoding quality: [0-9]

pImagePath

Input file path

iMethodValue

Interpolation method of converting Bayer to RGB24: 0-nearest neighbor 1-bilinearity 2-Hamilton

nReserved

Reserved.

5.1.50 MV_SAVE_POINT_CLOUD_PARAM

Structure about parameters of saving 3D point cloud data

Structure Definition

```
struct{
  unsigned int                     nLinePntNum;
  unsigned int                     nLineNum;
  MvGvspPixelType                  enSrcPixelType;
  unsigned char                    *pSrcData;
  unsigned int                     nSrcDataLen;
  unsigned char                    *pDstBuf;
  unsigned int                     nDstBufSize;
  unsigned int                     nDstBufLen;
  MV_SAVE_POINT_CLOUD_FILE_TYPE    enPointCloudFileType;
  unsigned int                        nReserved[8];
}MV_SAVE_POINT_CLOUD_PARAM;
```

Members

nLinePntNum

The number of points in each row, which is the image width

nLineNum

The number of rows, which is the image height

enSrcPixelType

The pixel format of the input data, see the enumeration definition ***MvGvspPixelType*** for details.

pSrcData

Input data buffer

nSrcDataLen

Input data size

pDstBuf

Output pixel data buffer

nDstBufSize

Provided output buffer size, the value is (nLinePntNum * nLineNum * (16*3 + 4) + 2048)

nDstBufLen

Buffer size of output pixel data

enPointCloudFileType

Provided file type of output point data, see the enumeration definition ***MV_SAVE_POINT_CLOUD_FILE_TYPE*** for details.

nReserved

Reserved.

5.1.51 MV_TRANSMISSION_TYPE

Structure about transmission modes.

Structure Definition

```
struct{
  MV_GIGE_TRANSMISSION_TYPE           enTransmissionType;
  unsigned int                        nDestIp;
  unsigned short                      nDestPort;
  unsigned int                        nReserved[32];
}MV_TRANSMISSION_TYPE;
```

Members

enTransmissionType

Transmission mode, see the enumeration type ***MV_GIGE_TRANSMISSION_TYPE*** for details.

nDestIp

Target IP, it is valid when transmission mode is multicast.

nDestPort

Target port, it is valid when transmission mode is multicast.

nReserved

Reserved.

5.1.52 MV_USB3_DEVICE_INFO

Structure about USB3 device information

Structure Definition

```
struct{
  unsigned char     CrtlInEndPoint;
  unsigned char     CrtlOutEndPoint;
  unsigned char     StreamEndPoint;
  unsigned char     EventEndPoint;
  unsigned short    idVendor;
  unsigned short    idProduct;
  unsigned int      nDeviceNumber;
  unsigned char     chDeviceGUID[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chVendorName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chModelName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chFamilyName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chDeviceVersion[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chManufacturerName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chSerialNumber[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned char     chUserDefinedName[INFO_MAX_BUFFER_SIZE/*64*/];
  unsigned int      nbcdUSB;
  unsigned int      nReserved[3];
}MV_USB3_DEVICE_INFO;
```

Members

CrtlInEndPoint

Control input port

CrtlOutEndPoint

Control output port

StreamEndPoint

Stream port

EventEndPoint

Event port

idVendor

Supplier ID

nDeviceNumber

Device No.

chDeviceGUID

Device GUID No.

chVendorName

Supplier name

chModelName

Model name

chFamilyName

Family name

chDeviceVersion

Device version

chManufacturerName

Manufacturer name

chSerialNumber

Serial No.

chUserDefinedName

Custom name

nbcdUSB

Supported USB protocol

nReserved

Reserved.

See Also

MV_CC_DEVICE_INFO

5.1.53 MV_XML_FEATURE_Base

IBase node information structure

Structure Definition

```
struct{
  MV_XML_AccessMode    enAccessMode;
}MV_XML_FEATURE_Base;
```

Members

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

5.1.54 MV_XML_FEATURE_Boolean

IBoolean node information structure

Structure Definition

```
struct{
  char                 strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                 strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                 strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                 strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  MV_XML_Visibility    enVisivility;
  MV_XML_AccessMode    enAccessMode;
  int                  bIsLocked;
  bool                 bValue;
  unsigned int         nReserved[4];
}MV_XML_FEATURE_Boolean;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

bValue

Current value

nReserved

Reserved.

5.1.55 MV_XML_FEATURE_Category

ICategory node information structure

Structure Definition

```
struct{
  char                          strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                          strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                          strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                          strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility        enVisivility;
  unsigned int                  nReserved[4];
}MV_XML_FEATURE_Category;
```

Members

strDescription

Node description, not supported now, reserved

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

nReserved

Reserved.

5.1.56 MV_XML_FEATURE_Command

ICommand node information structure

Structure Definition

```
struct{
  char                        strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                        strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility      enVisivility;
  enum MV_XML_AccessMode      enAccessMode;
  int                         bIsLocked;
  unsigned int                nReserved[4];
}MV_XML_FEATURE_Command;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nReserved

Reserved.

5.1.57 MV_XML_FEATURE_EnumEntry

IEnumEntry node information structure

Structure Definition

```
struct{
  char                         strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                         strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  int                          bIsImplemented;
  int                          nParentsNum;
  MV_XML_NODE_FEATURE          stParentsList[MV_MAX_XML_PARENTS_NUM/*8*/];
  enum MV_XML_Visibility       enVisivility;
  int64_t                      nValue;
  enum MV_XML_AccessMode       enAccessMode;
  int                          bIsLocked;
  int                          nReserved[8];
}MV_XML_FEATURE_EnumEntry;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved

strToolTip

Prompt

bIsImplemented

Take effect or not: 0-no, 1-yes

nParentsNum

The number of parent nodes

stParentsList

Parent nodes list

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

nValue

Current value

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nReserved

Reserved.

5.1.58 MV_XML_FEATURE_Enumeration

IEnumeration node information structure

Structure Definition

```
struct{
  enum MV_XML_Visibility     enVisivility;
  char                       strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                       strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                       strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                       strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  int                        nSymbolicNum;
  char                       strCurrentSymbolic[MV_MAX_XML_SYMBOLIC_STRLEN_C/
*64*/];
  char                       strSymbolic[MV_MAX_XML_SYMBOLIC_NUM/*64*/]
[MV_MAX_XML_SYMBOLIC_STRLEN_C/*64*/];
  enum MV_XML_AccessMode     enAccessMode;
  int                        bIsLocked;
  int64_t                    nValue;
  unsigned int               nReserved[4];
}MV_XML_FEATURE_Enumeration;
```

Members

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

strDescription

Node description, not supported now, reserved

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

nSymbolicNum

The number of symbols

strCurrentSymbolic

Current symbol index, refer to MvCameraNode for details.

strSymbolic

Symbol information

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nValue

Current value

nReserved

Reserved.

5.1.59 MV_XML_FEATURE_Float

IFloat node information structure

Structure Definition

```
struct{
  char                        strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                        strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility      enVisivility;
  enum MV_XML_AccessMode      enAccessMode;
  int                         bIsLocked;
  double                      dfValue;
  double                      dfMinValue;
  double                      dfMaxValue;
  double                      dfIncrement;
  unsigned int                 nReserved[4];
}MV_XML_FEATURE_Float;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved.

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

dfValue

Current value

dfMinValue

Minimum value

dfMaxValue

Maximum value

dfIncrement

Increment

nReserved

Reserved.

5.1.60 MV_XML_FEATURE_Integer

IInteger node information structure

Structure Definition

```
struct{
  char                     strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                     strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                     strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                     strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility   enVisivility;
  enum MV_XML_AccessMode   enAccessMode;
  int                      bIsLocked;
  int64_t                  nValue;
  int64_t                  nMinValue;
  int64_t                  nMaxValue;
  int64_t                  nIncrement;
  unsigned int             nReserved[4];
}MV_XML_FEATURE_Integer;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved.

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nValue

Current value

nMinValue

Minimum value

nMaxValue

Maximum value

nIncrement

Increment

nReserved

Reserved.

5.1.61 MV_XML_FEATURE_Port

IPort node information structure

Structure Definition

```
struct{
  enum MV_XML_Visibility    enVisivility;
  char                      strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                      strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                      strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                      strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_AccessMode    enAccessMode;
  int                       bIsLocked;
  unsigned int              nReserved[4];
}MV_XML_FEATURE_Port;
```

Members

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

strDescription

Node description, not supported now, reserved.

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nReserved

Reserved.

5.1.62 MV_XML_FEATURE_Register

IRegister node information structure

Structure Definition

```
struct _MV_XML_FEATURE_Register_{
  char                        strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                        strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                        strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility      enVisivility;
  enum MV_XML_AccessMode      enAccessMode;
  int                         bIsLocked;
  int64_t                     nAddrValue;
  unsigned int                nReserved[4];
}MV_XML_FEATURE_Register;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved.

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

nAddrValue

Current value

nReserved

Reserved.

5.1.63 MV_XML_FEATURE_String

IString node information structure

Structure Definition

```
struct{
  char                     strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                     strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                     strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                     strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  enum MV_XML_Visibility   enVisivility;
  enum MV_XML_AccessMode   enAccessMode;
  int                      bIsLocked;
  char                     strValue[MV_MAX_XML_STRVALUE_STRLEN_C/*64*/];
  unsigned int             nReserved[4];
}MV_XML_FEATURE_String;
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, not supported now, reserved.

strToolTip

Prompt

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

enAccessMode

Accessing mode, see the enumeration ***MV_XML_AccessMode*** for details.

bIsLocked

Lock or not (not supported, reserved): 0-unlock, 1-lock

strValue

Current value

nReserved

Reserved.

5.1.64 MV_XML_FEATURE_Value

IValue node information structure

Structure Definition

```
struct{
  enum MV_XML_InterfaceType    enType;
  char                         strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                         strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  unsigned int                 nReserved[4];
}MV_XML_FEATURE_Value;
```

Members

enType

Node types, see the enumeration ***MV_XML_InterfaceType*** for details.

strDescription

Node description, not supported now, reserved.

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

nReserved

Reserved.

5.1.65 MV_XML_NODE_FEATURE

Single node basic attribute

Structure Definition

```
struct{
  enum MV_XML_InterfaceType    enType;
  enum MV_XML_Visibility       enVisivility;
  char                         strDescription[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  char                         strDisplayName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strName[MV_MAX_XML_NODE_STRLEN_C/*64*/];
  char                         strToolTip[MV_MAX_XML_DISC_STRLEN_C/*512*/];
  unsigned int                 nReserved[4];
}MV_XML_NODE_FEATURE;
```

Members

enType

Node types, see the enumeration ***MV_XML_InterfaceType*** for details.

enVisivility

Visible or not, see the enumeration ***MV_XML_Visibility*** for details.

strDescription

Node description, not supported now, reserved.

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

nReserved

Reserved.

5.1.66 MV_XML_NODES_LIST

Node list structure

Structure Definition

```
struct{
  unsigned int          nNodeNum;
  MV_XML_NODE_FEATURE   stNodes[MV_MAX_XML_NODE_NUM_C/*128*];
}MV_XML_NODES_LIST;
```

Members

nNodeNum

The number of nodes

stNodes

Single node information, see the structure ***MV_XML_NODE_FEATURE*** for details.

5.1.67 MVCC_CIRCLE_INFO

Structure About Circle Area Information

Member	Data Type	Description
stCenterPoint	***MVCC_POINTF***	The circle center information
fR1	float	The width radius, which depends on the image relative position. The value range: [0,1.0]. The radius is related to the circle center. The circle needs to be drawn within the display frame range, otherwise, an error occurs.
fR2	float	The height radius, which depends on the image relative position. The value range: [0,1.0]. The radius is related to the circle center. The circle needs to be drawn within the display frame range, otherwise, an error occurs.
stColor	***MVCC_COLORF***	The color of auxiliary line
nLineWidth	unsigned int	The auxiliary line width. The value can be 1 or 2.
nReserved[4]	unsigned int	Reserved.

5.1.68 MVCC_COLORF

Structure about Color of Auxiliary Line

Member	Data Type	Description
fR	float	R value, range: [0.0,1.0]. It refers to the relative color of pixel.
fG	float	G value, range: [0.0,1.0]. It refers to the relative color of pixel.
fB	float	B value, range: [0.0,1.0]. It refers to the relative color of pixel.
fAlpha	float	Alpha value, range: [0.0,1.0]. It refers to the relative transparency of pixel.
nReserved[4]	unsigned int	Reserved.

5.1.69 MVCC_ENUMENTRY

Structure about Enumerated Type

Member	Data Type	Description
nValue	unsigned int	The assigned value
chSymbolic	char	The enumerator name corresponding to the assigned value. The maximum value is 64 (defined by the macro definition MV_MAX_ SYMBOLIC_LEN)
nReserved[4]	unsigned int	Reserved.

5.1.70 MVCC_ENUMVALUE

Enumeration type parameters structure

Structure Definition

```
struct{
  unsigned int    nCurValue;
  unsigned int    nSupportedNum;
  unsigned int    nSupportValue[MV_MAX_XML_SYMBOLIC_NUM/*64*/];
  unsigned int    nReserved[4];
}MVCC_ENUMVALUE;
```

Members

nCurValue

Current value

nSupportedNum

The number of valid data

nSupportValue

Supported enumeration types, each array indicates one type, up to **nSupportedNum** types are supported.

nReserved

Reserved.

5.1.71 MVCC_FLOATVALUE

Structure about float type parameter value

Structure Definition

```
struct{
  float           fCurValue;
  float           fMax;
  float           fMin;
  unsigned int    nReserved[4];
}MVCC_FLOATVALUE;
```

Members

fCurValue

Current value

fMax

Maximum value

fMin

Minimum value

nReserved

Reserved.

5.1.72 MVCC_INTVALUE

Structure about int type parameter value

Structure Definition

```
struct{
  unsigned int    nCurValue;
  unsigned int    nMax;
  unsigned int    nMin;
  unsigned int    nInc;
  unsigned int    nReserved[4];
}MVCC_INTVALUE;
```

Members

nCurValue

Current value

nMax

The maximum value

nMin

The minimum value

nInc

Increment

nReserved

Reserved.

5.1.73 MVCC_INTVALUE_EX

Structure about 64-bit int type parameter value

Structure Definition

```
struct{
  int64_t         nCurValue;
  int64_t         nMax;
  int64_t         nMin;
  int64_t         nInc;
  unsigned int    nReserved[16];
}MVCC_INTVALUE_EX;
```

Members

nCurValue

Current value

nMax

The maximum value

nMin

The minimum value

nInc

Increment

nReserved

Reserved

5.1.74 MVCC_LINES_INFO

Structure About Auxiliary Line Information

Member	Data Type	Description
stStartPoint	***MVCC_POINTF***	The start point of auxiliary line
stEndPoint	***MVCC_POINTF***	The end point of auxiliary line
stColor	***MVCC_COLORF***	The auxiliary line color
nLineWidth	unsigned int	The auxiliary line width. The value can be 1 or 2.
nReserved[4]	unsigned int	Reserved.

5.1.75 MVCC_POINTF

Structure about Custom Point Information

Member	Data Type	Description
fX	float	The point X-coordinate, value range: [0.0,1.0]. It refers to the relative position of the point in the image.
fY	float	The point Y-coordinate, value range: [0.0,1.0]. It refers to the relative position of the point in the image.
nReserved[4]	unsigned int	Reserved.

5.1.76 MVCC_RECT_INFO

Structure about Rectangle Frame Information

Member	Data Type	Description
fTop	float	The Y-coordinate of the top side of the rectangle, value range: [0.0,1.0].
fBottom	float	The Y-coordinate of the bottom side of the rectangle, value range: [0.0,1.0].
fLeft	float	The X coordinate of the left side of the rectangle, value range: [0.0,1.0].
fRight	float	The X coordinate of the right side of the rectangle, value range: [0.0,1.0].
stColor	***MVCC_COLORF***	The color of auxiliary line
nLineWidth	unsigned int	The auxiliary line width. The value can be 1 or 2.
nReserved[4]	unsigned int	Reserved.

5.1.77 MVCC_STRINGVALUE

Structure about string type parameter value

Structure Definition

```
struct{
  char            chCurValue[256];
  unsigned int    nReserved[4];
}MVCC_STRINGVALUE;
```

Members

chCurValue

Current value

nReserved

Reserved.

5.2 Enumeration

5.2.1 MV_CAM_ACQUISITION_MODE

Acquisition mode enumeration

Enumeration Definition

```
enum{
  MV_ACQ_MODE_SINGLE      = 0,
  MV_ACQ_MODE_MUTLI       = 1,
  MV_ACQ_MODE_CONTINUOUS  = 2
}MV_CAM_ACQUISITION_MODE
```

Members

MV_ACQ_MODE_SINGLE

Single frame mode

MV_ACQ_MODE_MUTLI

Multi-frame mode

MV_ACQ_MODE_CONTINUOUS

Continuous acquisition mode

5.2.2 MV_CAM_BALANCEWHITE_AUTO

Enumeration of auto white balance mode

Enumeration Definition

```
enum{
  MV_BALANCEWHITE_AUTO_OFF         = 0,
  MV_BALANCEWHITE_AUTO_CONTINUOUS  = 1,
  MV_BALANCEWHITE_AUTO_ONCE        = 2
}MV_CAM_BALANCEWHITE_AUTO
```

Members

MV_BALANCEWHITE_AUTO_OFF

Disable

MV_BALANCEWHITE_AUTO_CONTINUOUS

Continuous

MV_BALANCEWHITE_AUTO_ONCE

Once

5.2.3 MV_CAM_EXPOSURE_AUTO_MODE

Enumeration of auto exposure mode enumeration

Enumeration Definition

```
enum{
  MV_EXPOSURE_AUTO_MODE_OFF          = 0,
  MV_EXPOSURE_AUTO_MODE_ONCE         = 1,
  MV_EXPOSURE_AUTO_MODE_CONTINUOUS   = 2
}MV_CAM_EXPOSURE_AUTO_MODE
```

Members

MV_EXPOSURE_AUTO_MODE_OFF

Disable

MV_EXPOSURE_AUTO_MODE_ONCE

Once

MV_EXPOSURE_AUTO_MODE_CONTINUOUS

Continuous

5.2.4 MV_CAM_GAIN_MODE

Gain mode enumeration

Enumeration Definition

```
enum{
  MV_GAIN_MODE_OFF          = 0,
  MV_GAIN_MODE_ONCE         = 1,
  MV_GAIN_MODE_CONTINUOUS   = 2
}MV_CAM_GAIN_MODE
```

Members

MV_GAIN_MODE_OFF

Disable

MV_GAIN_MODE_ONCE

Once

MV_GAIN_MODE_CONTINUOUS

Continuous

5.2.5 MV_CAM_GAMMA_SELECTOR

Gamma type enumeration

Enumeration Definition

```
enum{
  MV_GAMMA_SELECTOR_USER        = 1,
  MV_GAMMA_SELECTOR_SRGB        = 2
}MV_CAM_GAMMA_SELECTOR
```

Members

MV_GAMMA_SELECTOR_USER

Custom

MV_GAMMA_SELECTOR_SRGB

sRGB type

5.2.6 MV_CAM_TRIGGER_MODE

Triggering mode enumeration

Enumeration Definition

```
enum{
  MV_TRIGGER_MODE_OFF    = 0,
  MV_TRIGGER_MODE_ON     = 1
}MV_CAM_TRIGGER_MODE
```

Members

MV_TRIGGER_MODE_OFF

Disable

MV_TRIGGER_MODE_ON

Enable

5.2.7 MV_CAM_TRIGGER_SOURCE

Triggering source

Enumeration Definition

```
enum{
  MV_TRIGGER_SOURCE_LINE0       = 0,
  MV_TRIGGER_SOURCE_LINE1        = 1,
  MV_TRIGGER_SOURCE_LINE2        = 2,
  MV_TRIGGER_SOURCE_LINE3        = 3,
  MV_TRIGGER_SOURCE_COUNTER0     = 4,
  MV_TRIGGER_SOURCE_SOFTWARE     = 7
}MV_CAM_TRIGGER_SOURCE
```

Members

MV_TRIGGER_SOURCE_LINE0

LINE0 hardware trigger

MV_TRIGGER_SOURCE_LINE1

LINE1 hardware trigger

MV_TRIGGER_SOURCE_LINE2

LINE2 hardware trigger

MV_TRIGGER_SOURCE_LINE3

LINE3 hardware trigger

MV_TRIGGER_SOURCE_COUNTER0

COUNTER0 hardware trigger

MV_TRIGGER_SOURCE_SOFTWARE

Software trigger

5.2.8 MV_CC_BAYER_NOISE_FEATURE_TYPE

Enumeration about Noise Characteristics

Member	Marco Definition Value	Description
MV_CC_BAYER_NOISE_ FEATURE_TYPE_INVALID	0	Invalid
MV_CC_BAYER_NOISE_ FEATURE_TYPE_PROFILE	1	Noise curve
MV_CC_BAYER_NOISE_ FEATURE_TYPE_LEVEL	2	Noise level
MV_CC_BAYER_NOISE_ FEATURE_TYPE_DEFAULT	3	Default value

5.2.9 MV_CC_GAMMA_TYPE

Enumeration about Gamma Type

Enumeration Type	Macro Definition Value	Description
MV_CC_GAMMA_TYPE_NONE	0	Disable.
MV_CC_GAMMA_TYPE_VALUE	1	Gamma value
MV_CC_GAMMA_TYPE_USER_CURVE	2	Gamma curve: 8bit. Required length: 256*sizeof(unsigned char) 10bit. Required length: 1024*sizeof(unsigned short) 12bit. Required length: 4096*sizeof(unsigned short) 16bit. Required length: 65536*sizeof(unsigned short)
MV_CC_GAMMA_TYPE_LRGB2SRGB	3	Linear RGB to sRGB.
MV_CC_GAMMA_TYPE_SRGB2LRGB	4	sRGB to linear RGB. **Note** This parameter is valid for color interpolation only, it is invalid for color correction.

5.2.10 MV_CC_STREAM_EXCEPTION_TYPE

Enumeration About Stream Exceptions of USB3.0

Enumeration Type	Macro Definition Value	Description
MV_CC_STREAM_EXCEPTION_ABNORMAL_IMAGE	0x4001	Abnormal image. The frame is dropped.
MV_CC_STREAM_EXCEPTION_LIST_OVERFLOW	0x4002	Buffer overflow. Clear the oldest frame.
MV_CC_STREAM_EXCEPTION_LIST_EMPTY	0x4003	Buffer is empty. The frame is dropped.

Enumeration Type	Macro Definition Value	Description
MV_CC_STREAM_EXCEPTION_ RECONNECTION	0x4004	Disconnection restored.
MV_CC_STREAM_EXCEPTION_ DISCONNECTED	0x4005	Disconnection restoring failed. Streaming paused.
MV_CC_STREAM_EXCEPTION_ DEVICE	0x4006	Device exception. Streaming paused.

5.2.11 MV_GIGE_EVENT

Event enumeration type

Enumeration Definition

```
enum{
  MV_EVENT_ExposureEnd                  = 1,
  MV_EVENT_FrameStartOvertrigger        = 2,
  MV_EVENT_AcquisitionStartOvertrigger = 3,
  MV_EVENT_FrameStart                   = 4,
  MV_EVENT_AcquisitionStart             = 5,
  MV_EVENT_EventOverrun                 = 6
}MV_GIGE_EVENT
```

Members

MV_EVENT_ExposureEnd

The end of each frame exposure, not support

MV_EVENT_FrameStartOvertrigger

Frame starts over-trigger (the next frame is triggered before the end of the previous frame trigger), not support

MV_EVENT_AcquisitionStartOvertrigger

Streaming start over-trigger（the streaming signal is sent too often), not support

MV_EVENT_FrameStart

Start each frame, not support

MV_EVENT_AcquisitionStart

Start streaming (continuous or single frame mode), not support

MV_EVENT_EventOverrun

Event over-trigger (the event is sent too often), not support

5.2.12 MV_GIGE_TRANSMISSION_TYPE

Enumeration of transmission modes, including unicast mode, multicast mode, and so on.

Enumeration Definition

```
enum{
  MV_GIGE_TRANSTYPE_UNICAST                = 0x0,
  MV_GIGE_TRANSTYPE_MULTICAST              = 0x1,
  MV_GIGE_TRANSTYPE_LIMITEDBROADCAST       = 0x2,
  MV_GIGE_TRANSTYPE_SUBNETBROADCAST        = 0x3,
  MV_GIGE_TRANSTYPE_CAMERADEFINED          = 0x4,
  MV_GIGE_TRANSTYPE_UNICAST_DEFINED_PORT   = 0x5,
  MV_GIGE_TRANSTYPE_UNICAST_WITHOUT_RECV   = 0x00010000,
  MV_GIGE_TRANSTYPE_MULTICAST_WITHOUT_RECV = 0x00010001,
}MV_GIGE_TRANSMISSION_TYPE;
```

Members

MV_GIGE_TRANSTYPE_UNICAST
Unicast

MV_GIGE_TRANSTYPE_MULTICAST
Multicast

MV_GIGE_TRANSTYPE_LIMITEDBROADCAST
LAN broadcast

MV_GIGE_TRANSTYPE_SUBNETBROADCAST
Subnet broadcast

MV_GIGE_TRANSTYPE_CAMERADEFINED
Get from camera

MV_GIGE_TRANSTYPE_UNICAST_DEFINED_PORT
Port No. of getting image data

MV_GIGE_TRANSTYPE_UNICAST_WITHOUT_RECV
Unicast mode, but not receive image data

MV_GIGE_TRANSTYPE_MULTICAST_WITHOUT_RECV
Multiple mode, but not receive image data

5.2.13 MV_GRAB_STRATEGY

Strategy enumeration definition

Enumeration Definition

```
enum _MV_GRAB_STRATEGY_{
  MV_GrabStrategy_OneByOne          = 0,
  MV_GrabStrategy_LatestImagesOnly  = 1,
  MV_GrabStrategy_LatestImages      = 2,
  MV_GrabStrategy_UpcomingImage     = 3,
}MV_GRAB_STRATEGY;
```

Members

MV_GrabStrategy_OneByOne

Get image frames one by one in the chronological order, it is the default strategy.

MV_GrabStrategy_LatestImagesOnly

Only get the latest one frame in the list, and clear the rest images in the list.

MV_GrabStrategy_LatestImages

Get the latest image in the list, and the quantity of frames depends on the parameter **OutputQueueSize**, value range: [1,ImageNodeNum]. If the **OutputQueueSize** values 1, the strategy is same to "LatestImagesOnly", and if the **OutputQueueSize** values "ImageNodeNum", the strategy is same to "OneByOne".

MV_GrabStrategy_UpcomingImage

Wait for the upcoming frame.

5.2.14 MV_IMAGE_RECONSTRUCTION_METHOD

Enumeration About Image Processing Modes of Time-Division Exposure

Enumeration Type	Macro Definition Value	Description
MV_SPLIT_BY_LINE	1	Split the source image into multiple images by lines.

5.2.15 MV_IMG_FLIP_TYPE

Enumeration about Flip Types

Member	Marco Definition Value	Description
MV_FLIP_VERTICAL	1	Vertical
MV_FLIP_HORIZONTAL	2	Horizontal

5.2.16 MV_IMG_ROTATION_ANGLE

Enumeration about Rotation Angle

Member	Marco Definition Value	Description
MV_IMAGE_ROTATE_90	1	90°
MV_IMAGE_ROTATE_180	2	180°
MV_IMAGE_ROTATE_270	3	270°

5.2.17 MV_SAVE_IAMGE_TYPE

Picture format type enumeration

Enumeration Definition

```
enum{
  MV_Image_Undefined  = 0,
  MV_Image_Bmp        = 1,
  MV_Image_Jpeg       = 2,
  MV_Image_Png        = 3,
  MV_Image_Tif        = 4,
}MV_SAVE_IAMGE_TYPE
```

Members

MV_Image_Undefined

Undefined

MV_Image_Bmp

BMP picture

MV_Image_Jpeg

JPEG picture

MV_Image_Png

PNG picture

MV_Image_Tif

TIF picture

5.2.18 MV_SAVE_POINT_CLOUD_FILE_TYPE

The saved 3D data formats

Enumeration Definition

```
enum MV_SAVE_POINT_CLOUD_FILE_TYPE{
  MV_PointCloudFile_Undefined           = 0,
  MV_PointCloudFile_PLY                 = 1,
  MV_PointCloudFile_CSV                 = 2,
  MV_PointCloudFile_OBJ                 = 3,
};
```

Members

MV_PointCloudFile_Undefined

Undefined point cloud format

MV_PointCloudFile_PLY

The point cloud format named PLY

MV_PointCloudFile_CSV

The point cloud format named CSV

MV_PointCloudFile_OBJ

The point cloud format named OBJ

5.2.19 MV_SORT_METHOD

Enumeration About Sorting Modes

Enumeration Type	Macro Definition Value	Description
SortMethod_SerialNumber	0	Sort by the serial number
SortMethod_UserID	1	Sort by the user-defined name
SortMethod_CurrentIP_ASC	2	Sort by the current IP address in ascending order
SortMethod_CurrentIP_DESC	3	Sort by the current IP address in descending order

5.2.20 MV_XML_AccessMode

Accessing mode enumeration

Enumeration Definition

```
enum{
  AM_NI,
  AM_NA,
  AM_WO,
  AM_RO,
  AM_RW,
  AM_Undefined,
  AM_CycleDetect
}MV_XML_AccessMode
```

Members

AM_NI

Not implemented

AM_NA

Not available

AM_WO

Write only

AM_RO

Read only

AM_RW

Read and write

AM_Undefined

Object is not yet initialized

AM_CycleDetect

Used internally for AccessMode cycle detection

5.2.21 MV_XML_InterfaceType

Interface type, to which each node corresponds.

Enumeration Definition

```
enum MV_XML_InterfaceType{
  IFT_IValue,
  IFT_IBase,
  IFT_IInteger,
  IFT_IBoolean,
  IFT_ICommand,
  IFT_IFloat,
  IFT_IString,
  IFT_IRegister,
  IFT_ICategory,
  IFT_IEnumeration,
  IFT_IEnumEntry,
  IFT_IPort
}MV_XML_InterfaceType
```

Members

IFT_IValue

IValue interface

IFT_IBase

IBase interface

IFT_IInteger

IInteger interface

IFT_IBoolean

IBoolean interface

IFT_ICommand

ICommand interface

IFT_IFloat

IFloat interface

IFT_IString

IString interface

IFT_IRegister

IRegister interface

IFT_ICategory

IInteger interface

IFT_IEnumeration

IEnumeration interface

IFT_IEnumEntry

IEnumEntry interface

IFT_IPort

IPort interface

5.2.22 MV_XML_Visibility

Visible mode enumeration

Enumeration Definition

```
enum{
  V_Beginner       = 0,
  V_Expert         = 1,
  V_Guru           = 2,
  V_Invisible      = 3,
  V_Undefined      = 99
}MV_XML_Visibility
```

Members

V_Beginner

Always visible

V_Expert

Visible for experts or Gurus

V_Guru

Visible for Gurus

V_Invisible

Not Visible

V_Undefined

Object is not yet initialized

5.2.23 MvGvspPixelType

Enumeration of GigE protocol pixel types

```
enum{
  PixelType_Gvsp_Undefined                        = -1,
    // Mono buffer format defines
  PixelType_Gvsp_Mono1p                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(1) | 0x0037),
  PixelType_Gvsp_Mono2p                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(2) | 0x0038),
  PixelType_Gvsp_Mono4p                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(4) | 0x0039),
  PixelType_Gvsp_Mono8                            =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8) | 0x0001),
  PixelType_Gvsp_Mono8_Signed                     =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8)  | 0x0002),
  PixelType_Gvsp_Mono10                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0003),
  PixelType_Gvsp_Mono10_Packed                    =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0004),
  PixelType_Gvsp_Mono12                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0005),
  PixelType_Gvsp_Mono12_Packed                    =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0006),
  PixelType_Gvsp_Mono14                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0025),
  PixelType_Gvsp_Mono16                           =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0007),
    // Bayer buffer format defines
  PixelType_Gvsp_BayerGR8                         =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8) | 0x0008),
  PixelType_Gvsp_BayerRG8                         =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8) | 0x0009),
  PixelType_Gvsp_BayerGB8                         =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8) | 0x000A),
  PixelType_Gvsp_BayerBG8                         =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(8) | 0x000B),
  PixelType_Gvsp_BayerGR10                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x000C),
  PixelType_Gvsp_BayerRG10                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x000D),
  PixelType_Gvsp_BayerGB10                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x000E),
  PixelType_Gvsp_BayerBG10                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x000F),
  PixelType_Gvsp_BayerGR12                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0010),
  PixelType_Gvsp_BayerRG12                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0011),
  PixelType_Gvsp_BayerGB12                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0012),
  PixelType_Gvsp_BayerBG12                        =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0013),
  PixelType_Gvsp_BayerGR10_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0026),
  PixelType_Gvsp_BayerRG10_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0027),
  PixelType_Gvsp_BayerGB10_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0028),
  PixelType_Gvsp_BayerBG10_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x0029),
  PixelType_Gvsp_BayerGR12_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x002A),
  PixelType_Gvsp_BayerRG12_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x002B),
  PixelType_Gvsp_BayerGB12_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x002C),
  PixelType_Gvsp_BayerBG12_Packed                 =   (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(12) | 0x002D),
  PixelType_Gvsp_BayerGR16                      =    (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x002E),
  PixelType_Gvsp_BayerRG16                      =    (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x002F),
  PixelType_Gvsp_BayerGB16                      =    (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0030),
  PixelType_Gvsp_BayerBG16                      =    (MV_GVSP_PIX_MONO |
MV_PIXEL_BIT_COUNT(16) | 0x0031),
    // RGB Packed buffer format defines
  PixelType_Gvsp_RGB8_Packed                    =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x0014),
  PixelType_Gvsp_BGR8_Packed                    =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x0015),
  PixelType_Gvsp_RGBA8_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(32) | 0x0016),
  PixelType_Gvsp_BGRA8_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(32) | 0x0017),
  PixelType_Gvsp_RGB10_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0018),
  PixelType_Gvsp_BGR10_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0019),
  PixelType_Gvsp_RGB12_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x001A),
  PixelType_Gvsp_BGR12_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x001B),
  PixelType_Gvsp_RGB16_Packed                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0033),
  PixelType_Gvsp_RGB10V1_Packed                 =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(32) | 0x001C),
  PixelType_Gvsp_RGB10V2_Packed                 =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(32) | 0x001D),
  PixelType_Gvsp_RGB12V1_Packed                 =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(36) | 0X0034),
  PixelType_Gvsp_RGB565_Packed                  =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0035),
  PixelType_Gvsp_BGR565_Packed                  =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0X0036),
    // YUV Packed buffer format defines
  PixelType_Gvsp_YUV411_Packed                  =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(12) | 0x001E),
  PixelType_Gvsp_YUV422_Packed                  =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x001F),
  PixelType_Gvsp_YUV422_YUYV_Packed             =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0032),
  PixelType_Gvsp_YUV444_Packed                  =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x0020),
  PixelType_Gvsp_YCBCR8_CBYCR                   =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x003A),
  PixelType_Gvsp_YCBCR422_8                     =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x003B),
  PixelType_Gvsp_YCBCR422_8_CBYCRY              =    (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0043),
  PixelType_Gvsp_YCBCR411_8_CBYYCRYY              =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(12) | 0x003C),
  PixelType_Gvsp_YCBCR601_8_CBYCR                 =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x003D),
  PixelType_Gvsp_YCBCR601_422_8                   =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x003E),
  PixelType_Gvsp_YCBCR601_422_8_CBYCRY            =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0044),
  PixelType_Gvsp_YCBCR601_411_8_CBYYCRYY          =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(12) | 0x003F),
  PixelType_Gvsp_YCBCR709_8_CBYCR                 =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x0040),
  PixelType_Gvsp_YCBCR709_422_8                   =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0041),
  PixelType_Gvsp_YCBCR709_422_8_CBYCRY            =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(16) | 0x0045),
  PixelType_Gvsp_YCBCR709_411_8_CBYYCRYY          =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(12) | 0x0042),
    // RGB Planar buffer format defines
  PixelType_Gvsp_RGB8_Planar                      =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(24) | 0x0021),
  PixelType_Gvsp_RGB10_Planar                     =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0022),
  PixelType_Gvsp_RGB12_Planar                     =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0023),
  PixelType_Gvsp_RGB16_Planar                     =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(48) | 0x0024),
  // Custom picture format
  PixelType_Gvsp_Jpeg                             =   (MV_GVSP_PIX_CUSTOM |
MV_PIXEL_BIT_COUNT(24) | 0x0001)
  PixelType_Gvsp_Coord3D_ABC32f                   =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(96) | 0x00C0),//0x026000C0
  PixelType_Gvsp_Coord3D_ABC32f_Planar            =   (MV_GVSP_PIX_COLOR |
MV_PIXEL_BIT_COUNT(96) | 0x00C1),//0x026000C1
  //Lossless decoding pixel format
  PixelType_Gvsp_HB_Mono8                         =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(8) | 0x0001),
  PixelType_Gvsp_HB_Mono10                        =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0003),
  PixelType_Gvsp_HB_Mono10_Packed                 =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0004),
  PixelType_Gvsp_HB_Mono12                        =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0005),
  PixelType_Gvsp_HB_Mono12_Packed                 =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0006),
  PixelType_Gvsp_HB_Mono16                        =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0007),
  PixelType_Gvsp_HB_BayerGR8                      =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(8) | 0x0008),
  PixelType_Gvsp_HB_BayerRG8                      =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(8) | 0x0009),
  PixelType_Gvsp_HB_BayerGB8                      =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(8) | 0x000A),
  PixelType_Gvsp_HB_BayerBG8                    =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(8) | 0x000B),
  PixelType_Gvsp_HB_BayerGR10                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x000C),
  PixelType_Gvsp_HB_BayerRG10                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x000D),
  PixelType_Gvsp_HB_BayerGB10                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x000E),
  PixelType_Gvsp_HB_BayerBG10                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x000F),
  PixelType_Gvsp_HB_BayerGR12                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0010),
  PixelType_Gvsp_HB_BayerRG12                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0011),
  PixelType_Gvsp_HB_BayerGB12                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0012),
  PixelType_Gvsp_HB_BayerBG12                   =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(16) | 0x0013),
  PixelType_Gvsp_HB_BayerGR10_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0026),
  PixelType_Gvsp_HB_BayerRG10_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0027),
  PixelType_Gvsp_HB_BayerGB10_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0028),
  PixelType_Gvsp_HB_BayerBG10_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x0029),
  PixelType_Gvsp_HB_BayerGR12_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x002A),
  PixelType_Gvsp_HB_BayerRG12_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x002B),
  PixelType_Gvsp_HB_BayerGB12_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x002C),
  PixelType_Gvsp_HB_BayerBG12_Packed            =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_MONO | MV_PIXEL_BIT_COUNT(12) | 0x002D),
  PixelType_Gvsp_HB_YUV422_Packed               =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(16) | 0x001F),
  PixelType_Gvsp_HB_YUV422_YUYV_Packed          =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(16) | 0x0032),
  PixelType_Gvsp_HB_RGB8_Packed                 =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(24) | 0x0014),
  PixelType_Gvsp_HB_BGR8_Packed                 =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(24) | 0x0015),
  PixelType_Gvsp_HB_RGBA8_Packed                =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(32) | 0x0016),
  PixelType_Gvsp_HB_BGRA8_Packed                =   (MV_GVSP_PIX_CUSTOM |
MV_GVSP_PIX_COLOR | MV_PIXEL_BIT_COUNT(32) | 0x0017),
}MvGvspPixelType
```

Remarks

The macro definitions of enumeration types are listed below:

Macro Definition	Value
MV_GVSP_PIX_MONO	0x01000000
MV_GVSP_PIX_COLOR	0x02000000
MV_PIXEL_BIT_COUNT(n)	((n) << 16)

Chapter 6 FAQ (Frequently Asked Question)

6.1 GigE Vision Camera

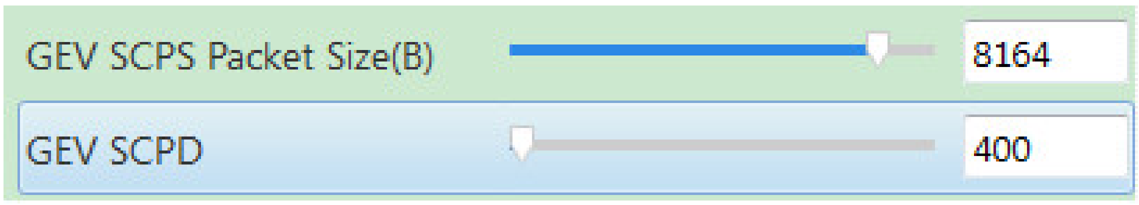
6.1.1 Why is there packet loss?

Cause

The abnormal network transmission environment causes the packet loss of data transmission.

Solution

1. Check if the bandwidth is sufficient.
2. Enable the NIC jumbo frame.
3. Disable firewall.
4. Increase the SCPD gradually till no packet loss.


GEV SCPS Packet Size(B) 8164
GEV SCPD 400

6.1.2 Why does link error occur in the normal compiled Demo?

Cause

No administrator permission for Demo directory will make it unable to write the .exe file.

Solution

Change the Demo directory to the directory with administrator permission.

6.1.3 Why can't I set the static IP under DHCP?

Cause

The camera with unpublished version limit the gateway, the 0.0.0.0 will display failed.

Solution

Upgrade firmware again.

6.1.4 Why do I failed to perform the software trigger command when calling SDK?

Cause

The trigger source is not set to software trigger.

Solution

Before performing software trigger command, make sure the camera is in software trigger mode and the trigger source is set to software trigger.

6.1.5 Why does the camera often be offline?

Cause 1

The NIC card is in sleep status.

Solution 1

Set the power option of operating system to avoid the computer going to the sleep status.

Cause 2

The network port may be not plugged in.

Solution 2

Check the network port status.

6.1.6 Why is no permission returned when calling API MV_CC_OpenDevice?

Cause 1

The camera is occupied.

Solution 1

Check if the camera is occupied or connected by other application.

Cause 2

The configured heartbeat timeout is too long, and the program exits abnormally without executing the API of shutting down device or destroying device handle. So the device remains occupied.

Solution 2

Wait till the heartbeat timed out or unplug the camera.

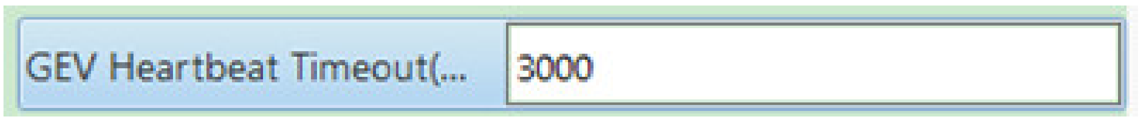
6.1.7 Why is there error code returned during debug process?

Cause

Debug will cause heartbeat sending timeout.

Solution

Lengthen the heartbeat time (example: 30s, and set the value to 3000). The default heartbeat time is 3s, see the picture below:



GEV Heartbeat Timeout(... 3000

6.1.8 Why is no data error returned when calling API MV_CC_GetOneFrameTimeout?

Cause

This API adopts active search method, and no data can be obtained when calling for only once.

Solution

Increase the timeout.

6.1.9 Why is there always no data when calling MV_CC_GetOneFrameTimeout?

Cause

Image registration callback function has been called at the same time. These two functions cannot be called at the same time.

Solution

Stop calling the registration callback function.

6.1.10 Why can't open the camera after finishing debugging abnormally?

Cause

To avoid the heartbeat timeout under debug process, the default value of camera heartbeat timeout is 60000ms (60s). So sometimes the camera cannot be opened after finishing debugging abnormally.

Solution

Shut down camera before exiting debugging.

6.2 USB3 Vision Camera

6.2.1 Why can't the MVS get the data or why is the frame rate far smaller the actual frame rare?

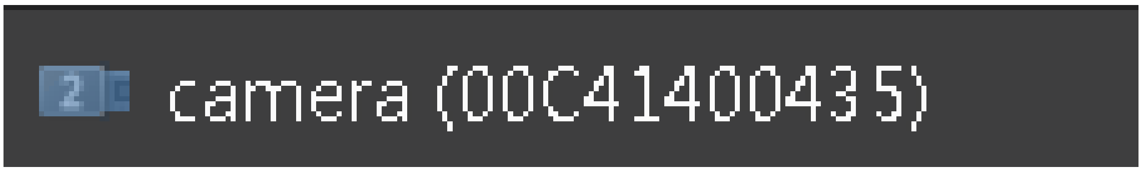
Cause

The USB connected with camera is in Version 2.0, and the bandwidth is not enough.

Solution

Make sure the USB connected with camera is in Version 3.0. You can check the USB version information by the following methods:

1. Check the digit of the icon in front of camera name in the device list.



2. Check whether the value of **USB Speed Mode** in the device property is **Highspeed** (USB 2.0) or **SuperSpeed** (USB 3.0).

Appendix A. Error Code

The error may occurred during the MVC SDK integration are listed here for reference. You can search for the error description according to returned error codes or name.

Error Type	Error Code	Description
General Error Codes: From 0x80000000 to 0x800000FF		
MV_E_HANDLE	0x80000000	Error or invalid handle.
MV_E_SUPPORT	0x80000001	Not supported function.
MV_E_BUFOVER	0x80000002	Buffer is full.
MV_E_CALLORDER	0x80000003	Incorrect calling order
MV_E_PARAMETER	0x80000004	Incorrect parameter.
MV_E_RESOURCE	0x80000006	Applying resource failed.
MV_E_NODATA	0x80000007	No data.
MV_E_PRECONDITION	0x80000008	Precondition error, or the running environment changed.
MV_E_VERSION	0x80000009	Version mismatches.
MV_E_NOENOUGH_BUF	0x8000000A	Insufficient memory.
MV_E_ABNORMAL_IMAGE	0x8000000B	Abnormal image. Incomplete image caused by packet loss.
MV_E_LOAD_LIBRARY	0x8000000C	Importing DLL (Dynamic Link Library) failed.
MV_E_NOOUTBUF	0x8000000D	No buffer node can be outputted.
MV_E_ENCRYPT	0x8000000E	Encryption error.
MV_E_UNKNOW	0x800000FF	Unknown error.
GenICam Series Error Codes: RFrom 0x80000100 to 0x800001FF		
MV_E_GC_GENERIC	0x80000100	Generic error.
MV_E_GC_ARGUMENT	0x80000101	Illegal parameters.
MV_E_GC_RANGE	0x80000102	The value is out of range.
MV_E_GC_PROPERTY	0x80000103	Attribute error
MV_E_GC_RUNTIME	0x80000104	Running environment error.

Error Type	Error Code	Description
MV_E_GC_LOGICAL	0x80000105	Incorrect logic
MV_E_GC_ACCESS	0x80000106	Node accessing condition error.
MV_E_GC_TIMEOUT	0x80000107	Timed out.
MV_E_GC_DYNAMICCAST	0x80000108	Conversion exception.
MV_E_GC_UNKNOW	0x800001FF	GenICam unknown error.
GigE Error Codes: From 0x80000200 to 0x800002FF, 0x80000221		
MV_E_NOT_IMPLEMENTED	0x80000200	The command is not supported by the device.
MV_E_INVALID_ADDRESS	0x80000201	The target address being accessed does not exist.
MV_E_WRITE_PROTECT	0x80000202	The target address is not writable.
MV_E_ACCESS_DENIED	0x80000203	The device has no access permission.
MV_E_BUSY	0x80000204	Device is busy, or the network disconnected.
MV_E_PACKET	0x80000205	Network packet error.
MV_E_NETER	0x80000206	Network error.
MV_E_IP_CONFLICT	0x80000221	Device IP address conflicted.
USB_STATUS Error Codes: From 0x80000300 to 0x800003FF		
MV_E_USB_READ	0x80000300	Reading USB error.
MV_E_USB_WRITE	0x80000301	Writing USB error.
MV_E_USB_DEVICE	0x80000302	Device exception.
MV_E_USB_GENICAM	0x80000303	GenICam error.
MV_E_USB_BANDWIDTH	0x80000304	Insufficient bandwidth.
MV_E_USB_UNKNOW	0x800003FF	USB unknown error.
Upgrade Error Codes: From 0x80000400 to 0x800004FF		
MV_E_UPG_FILE_MISMATCH	0x80000400	Firmware mismatches
MV_E_UPG_LANGUSGE_MISMATCH	0x80000401	Firmware language mismatches.
MV_E_UPG_CONFLICT	0x80000402	Upgrading conflicted (repeated upgrading requests during device upgrade).
MV_E_UPG_INNER_ERR	0x80000403	Camera internal error during upgrade.
MV_E_UPG_UNKNOW	0x800004FF	Unknown error during upgrade.

Error Type	Error Code	Description
Exception Error Codes: From 0x00008001 to 0x00008002		
MV_EXCEPTION_DEV_DISCONNECT	0x00008001	Device disconnected.
MV_EXCEPTION_VERSION_CHECK	0x00008002	SDK doesn't match the driver version.

Algorithm Error Codes

Error Type	Error Code	Description
General Error Codes		
MV_ALG_OK	0x00000000	OK
MV_ALG_ERR	0x00000000	Unknown error
Capability Related Error Codes		
MV_ALG_E_ABILITY_ARG	0x10000001	Invalid parameters of capabilities
Memory Related Error Codes (From 0x10000002 to 0x10000006)		
MV_ALG_E_MEM_NULL	0x10000002	The memory address is empty.
MV_ALG_E_MEM_ALIGN	0x10000003	The memory alignment is not satisfactory.
MV_ALG_E_MEM_LACK	0x10000004	No enough memory space.
MV_ALG_E_MEM_SIZE_ALIGN	0x10000005	The memory space does not meet the requirement of alignment.
MV_ALG_E_MEM_ADDR_ ALIGN	0x10000006	The memory address does not meet the requirement of alignment.
Image Related Error Codes (From 0x10000007 to 0x1000000A)		
MV_ALG_E_IMG_FORMAT	0x10000007	Incorrect image format or the image format is not supported.
MV_ALG_E_IMG_SIZE	0x10000008	Invalid image width and height.
MV_ALG_E_IMG_STEP	0x10000009	The image width/height and step parameters mismatched.
MV_ALG_E_IMG_DATA_NULL	0x1000000A	The storage address of image is empty.
Input/Output Related Error Codes (From 0x1000000B to 0x10000010)		
MV_ALG_E_CFG_TYPE	0x1000000B	Incorrect type for setting/getting parameters.

Error Type	Error Code	Description
MV_ALG_E_CFG_SIZE	0x1000000C	Incorrect size for setting/getting parameters.
MV_ALG_E_PRC_TYPE	0x1000000D	Incorrect processing type.
MV_ALG_E_PRC_SIZE	0x1000000E	Incorrect parameter size for processing.
MV_ALG_E_FUNC_TYPE	0x1000000F	Incorrect sub-process type.
MV_ALG_E_FUNC_SIZE	0x100000010	Incorrect parameter size for sub-processing.
Operation Parameters Related Error Codes (From 0x10000011 to 0x10000013)		
MV_ALG_E_PARAM_INDEX	0x100000011	Incorrect index parameter.
MV_ALG_E_PARAM_VALUE	0x100000012	Incorrect or invalid value parameter.
MV_ALG_E_PARAM_NUM	0x100000013	Incorrect param_num parameter.
API Calling Related Error Codes (From 0x10000014 to 0x10000016)		
MV_ALG_E_NULL_PTR	0x100000014	Pointer to function is empty.
MV_ALG_E_OVER_MAX_MEM	0x100000015	The maximum memory reached.
MV_ALG_E_CALL_BACK	0x100000016	Callback function error.
Algorithm Library Encryption Related Error Codes (0x10000017 and 0x10000018)		
MV_ALG_E_ENCRYPT	0x100000017	Encryption error.
MV_ALG_E_EXPIRE	0x100000018	Incorrect algorithm library service life.
Basic Errors of Inner Module (From 0x10000019 and 0x1000001B)		
MV_ALG_E_BAD_ARG	0x100000019	Incorrect value range of the parameter.
MV_ALG_E_DATA_SIZE	0x1000001A	Incorrect data size.
MV_ALG_E_STEP	0x1000001B	Incorrect data step.
Other Error Codes		
MV_ALG_E_CPUID	0x1000001C	The instruction set of optimized code does not supported by the CPU.
MV_ALG_WARNING	0x1000001D	Warning.
MV_ALG_E_TIME_OUT	0x1000001E	Algorithm library timed out.
MV_ALG_E_LIB_VERSION	0x1000001F	Algorithm version No. error.
MV_ALG_E_MODEL_VERSION	0x10000020	Model version No. error.

Error Type	Error Code	Description
MV_ALG_E_GPU_MEM_ALLOC	0x10000021	GUP memory allocation error.
MV_ALG_E_FILE_NON_EXIST	0x10000022	The file does not exist.
MV_ALG_E_NONE_STRING	0x10000023	The string is empty.
MV_ALG_E_IMAGE_CODEC	0x10000024	Image decoder error.
MV_ALG_E_FILE_OPEN	0x10000025	Opening file failed.
MV_ALG_E_FILE_READ	0x10000026	Reading file failed.
MV_ALG_E_FILE_WRITE	0x10000027	Writing to file failed.
MV_ALG_E_FILE_READ_SIZE	0x10000028	Incorrect file read size.
MV_ALG_E_FILE_TYPE	0x10000029	Incorrect file type.
MV_ALG_E_MODEL_TYPE	0x1000002A	Incorrect model type.
MV_ALG_E_MALLOC_MEM	0x1000002B	Memory allocation error.
MV_ALG_E_BIND_CORE_ FAILED	0x1000002C	Binding thread to core failed.
Denoising Related Error Codes (From 0x10402001 to 0x1040200f)		
MV_ALG_E_DENOISE_NE_ IMG_FORMAT	0x10402001	Incorrect image format of noise characteristics.
MV_ALG_E_DENOISE_NE_ FEATURE_TYPE	0x10402002	Incorrect noise characteristics type.
MV_ALG_E_DENOISE_NE_ PROFILE_NUM	0x10402003	Incorrect number of noise characteristics.
MV_ALG_E_DENOISE_NE_ GAIN_NUM	0x10402004	Incorrect number of noise characteristics gain.
MV_ALG_E_DENOISE_NE_ GAIN_VAL	0x10402005	Incorrect noise curve gain value.
MV_ALG_E_DENOISE_NE_BIN_ NUM	0x10402006	Incorrect number of noise curves.
MV_ALG_E_DENOISE_NE_ INIT_GAIN	0x10402007	Incorrect settings of noise initial gain.
MV_ALG_E_DENOISE_NE_ NOT_INIT	0x10402008	The noise is uninitialized.

Error Type	Error Code	Description
MV_ALG_E_DENOISE_COLOR_MODE	0x10402009	Incorrect color mode.
MV_ALG_E_DENOISE_ROI_NUM	0x1040200a	Incorrect number of ROIs.
MV_ALG_E_DENOISE_ROI_ORI_PT	0x1040200b	Incorrect ROI origin.
MV_ALG_E_DENOISE_ROI_SIZE	0x1040200c	Incorrect ROI size.
MV_ALG_E_DENOISE_GAIN_NOT_EXIST	0x1040200d	The camera gain does not exist (The maximum number of gains reached).
MV_ALG_E_DENOISE_GAIN_BEYOND_RANGE	0x1040200e	Invalid camera gain.
MV_ALG_E_DENOISE_NP_BUF_SIZE	0x1040200f	Incorrect noise characteristics memory size.

Appendix B. Sample Code

B.1 Perform Basic Functions of CamLink Cameras

Perform the basic functions of CamLink cameras, including connecting cameras, acquiring images, setting parameters, and so on.

CamLBasicDemo.cpp

```
#include "MvCameraControl.h"
#include <stdio.h>
#include <Windows.h>
#include <conio.h>


void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}


bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_CAMERALINK_DEVICE)
    {
        printf("chPortID: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chPortID);
        printf("chModelName: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chModelName);
        printf("chFamilyName: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chFamilyName);
        printf("chDeviceVersion: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chDeviceVersion);
        printf("chManufacturerName: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chManufacturerName);
        printf("Serial Number: [%s]\n", pstMVDevInfo-
>SpecialInfo.stCamLInfo.chSerialNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}


int GetParameters(void* handle)
{
    if ( NULL == handle )
    {
        return MV_E_PARAMETER;
    }

    int nRet = MV_OK;

    MVCC_INTVALUE stIntVal;
    memset(&stIntVal, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(handle, "Width", &stIntVal);
    if (MV_OK != nRet)
    {
        printf("Get Width fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    printf("Current Width [%d]\n", stIntVal.nCurValue);

    MVCC_ENUMVALUE stEnumVal;
    memset(&stEnumVal, 0, sizeof(MVCC_ENUMVALUE));
    nRet = MV_CC_GetEnumValue(handle, "TriggerMode", &stEnumVal);
    if (MV_OK != nRet)
    {
        printf("Get Trigger Mode fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    printf("Current TriggerMode [%d]\n", stEnumVal.nCurValue);

    MVCC_FLOATVALUE stFloatVal;
    memset(&stFloatVal, 0, sizeof(MVCC_FLOATVALUE));
    nRet = MV_CC_GetFloatValue(handle, "AcquisitionFrameRate", &stFloatVal);
    if (MV_OK != nRet)
    {
        printf("Get AcquisitionFrameRate fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    printf("Current AcquisitionFrameRate [%f] Fps\n", stFloatVal.fCurValue);

    bool bBoolVal = false;
    nRet = MV_CC_GetBoolValue(handle, "AcquisitionFrameRateEnable", &bBoolVal);
    if (MV_OK != nRet)
    {
        printf("Get AcquisitionFrameRateEnable fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    printf("Current AcquisitionFrameRateEnable [%d]\n", bBoolVal);

    MVCC_STRINGVALUE stStrVal;
    memset(&stStrVal, 0, sizeof(MVCC_STRINGVALUE));
    nRet = MV_CC_GetStringValue(handle, "DeviceUserID", &stStrVal);
    if (MV_OK != nRet)
    {
        printf("Get DeviceUserID fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    printf("Current DeviceUserID [%s]\n", stStrVal.chCurValue);

    return MV_OK;
}


int SetParameters(void* handle)
{
    if ( NULL == handle )
    {
        return MV_E_PARAMETER;
    }

    int nRet = MV_OK;

    nRet = MV_CC_SetIntValue(handle, "Width", 200);
    if (MV_OK != nRet)
    {
        printf("Set Width fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    nRet = MV_CC_SetFloatValue(handle, "AcquisitionFrameRate", 8.8f);
    if (MV_OK != nRet)
    {
        printf("Set AcquisitionFrameRate fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    nRet = MV_CC_SetBoolValue(handle, "AcquisitionFrameRateEnable", true);
    if (MV_OK != nRet)
    {
        printf("Set AcquisitionFrameRateEnable fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    nRet = MV_CC_SetStringValue(handle, "DeviceUserID", "UserIDChanged");
    if (MV_OK != nRet)
    {
        printf("Set DeviceUserID fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_ON);
    if (MV_OK != nRet)
    {
        printf("Set TriggerMode fail! nRet [0x%x]\n", nRet);
        return nRet;
    }
    nRet = MV_CC_SetEnumValue(handle, "TriggerSource",
MV_TRIGGER_SOURCE_SOFTWARE);
    if (MV_OK != nRet)
    {
        printf("Set TriggerSource fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    nRet = MV_CC_SetCommandValue(handle, "TriggerSoftware");
    if (MV_OK != nRet)
    {
        printf("Execute TriggerSoftware fail! nRet [0x%x]\n", nRet);
        return nRet;
    }

    return MV_OK;
}


void __stdcall ExceptionCallBack(unsigned int nMsgType, void* pUser)
{
    if(nMsgType == MV_EXCEPTION_DEV_DISCONNECT)
    {
        printf("MV_EXCEPTION_DEV_DISCONNECT");
    }
    else
    {
        printf("Unknown exception 0x[%x] ", nMsgType);
    }
}


int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    bool bDevConnected = false;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_CAMERALINK_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = true;

        nRet = MV_CC_RegisterExceptionCallBack(handle, ExceptionCallBack, 
handle);
        if (MV_OK != nRet)
        {
            printf("RegisterExceptionCallBack fail! nRet [0x%x]\n", nRet);
            break;
        }

        /******************characteristic interfaces for CameraLink 
device********************/
        unsigned int nBaudrateAblity = 0;
        nRet = MV_CAML_GetSupportBauderates(handle, &nBaudrateAblity);
        if (MV_OK != nRet)
        {
            printf("Get supported bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Current device supported bauderate [0x%x]\n", nBaudrateAblity);

        nRet = MV_CAML_SetDeviceBauderate(handle, MV_CAML_BAUDRATE_115200);
        if (MV_OK != nRet)
        {
            printf("Set Device Bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nCurrentBaudrate = 0;
        nRet = MV_CAML_GetDeviceBauderate(handle, &nCurrentBaudrate);
        if (MV_OK != nRet)
        {
            printf("Get device bauderate fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Current device bauderate [0x%x]\n", nCurrentBaudrate);

        /***************************properties 
configuration*********************************/
        nRet = GetParameters(handle);
        if (MV_OK != nRet)
        {
            printf("Get parameters fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = SetParameters(handle);
        if (MV_OK != nRet)
        {
            printf("Set parameters fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }
        bDevConnected = false;

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Device successfully closed.\n");
    } while (0);


    if (nRet != MV_OK)
    {
        if ( bDevConnected )
        {
            MV_CC_CloseDevice(handle);
            bDevConnected = false;
        }
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.2 Get Chunk Information

The sample code below shows how to enable the ChunkData function, confure ChunkData parameters and get ChunkData information.

ChunkData.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;

void WaitForKeyPress(void)
```

```
{
   while(!_kbhit())
   {
      Sleep(10);
   }
   _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
   if (NULL == pstMVDevInfo)
   {
      printf("The Pointer of pstMVDevInfo is NULL!\n");
      return false;
   }
   if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
   {
      int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
      int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
      int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
      int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x000000ff);

      printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
      printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
   }
   else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
   {
      printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
      printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
      printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
   }
   else
   {
      printf("Not support.\n");
   }

   return true;
}

// Get Image buffer function, you can get the chunk information from frame
information
void __stdcall ImageCallBackEx(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser)
{
   if (pFrameInfo)
   {
      printf("ImageCallBack:FrameNum[%d], ExposureTime[%f], SecondCount[%d],
CycleCount[%d], CycleOffset[%d]\n",
        pFrameInfo->nFrameNum, pFrameInfo->fExposureTime, pFrameInfo-
>nSecondCount, pFrameInfo->nCycleCount, pFrameInfo->nCycleOffset);

      MV_CHUNK_DATA_CONTENT* pUnparsedChunkContent = pFrameInfo-
>UnparsedChunkList.pUnparsedChunkContent;
      for(unsigned int i = 0;i < pFrameInfo->nUnparsedChunkNum; i++)
      {
        printf("ChunkInfo[%d]: ChunkID[0x%x], ChunkLen[%d]\n", i,
pUnparsedChunkContent->nChunkID, pUnparsedChunkContent->nChunkLen);
        pUnparsedChunkContent++;
      }
      printf("***********************************\n");
   }
}

int main()
{
   int nRet = MV_OK;
   void* handle = NULL;

   do
   {
      MV_CC_DEVICE_INFO_LIST stDeviceList;
      memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
      nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE, &stDeviceList);
      if (MV_OK != nRet)
      {
        printf("Enum Devices fail! nRet [0x%x]\n", nRet);
        break;
      }

      if (stDeviceList.nDeviceNum > 0)
      {
        for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
        {
           printf("[device %d]:\n", i);
           MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
           if (NULL == pDeviceInfo)
           {
              break;
           }
           PrintDeviceInfo(pDeviceInfo);
        }
      }
      else
      {
        printf("Find No Devices!\n");
        break;
      }
```

```
        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
          printf("Input error!\n");
          break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
          printf("Create Handle fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
          printf("Open Device fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBackEx, handle);
        if (MV_OK != nRet)
        {
          printf("Register Image CallBack fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_SetBoolValue(handle, "ChunkModeActive", true);
        if (MV_OK != nRet)
        {
          printf("Set Chunk Mode fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_SetEnumValueByString(handle, "ChunkSelector", "Exposure");
        if (MV_OK != nRet)
        {
          printf("Set Exposure Chunk fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_SetBoolValue(handle, "ChunkEnable", true);
        if (MV_OK != nRet)
        {
          printf("Set Chunk Enable fail! nRet [0x%x]\n", nRet);
          break;
        }
        nRet = MV_CC_SetEnumValueByString(handle, "ChunkSelector", "Timestamp");
        if (MV_OK != nRet)
        {
          printf("Set Timestamp Chunk fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_SetBoolValue(handle, "ChunkEnable", true);
        if (MV_OK != nRet)
        {
          printf("Set Chunk Enable fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
          printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
          printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
          break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
          printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
          printf("ClosDevice fail! nRet [0x%x]\n", nRet);
          break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
```

```
        printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
        break;
      }
   } while (0);


   if (nRet != MV_OK)
   {
      if (handle != NULL)
      {
        MV_CC_DestroyHandle(handle);
        handle = NULL;
      }
   }

   printf("Press a key to exit.\n");
   WaitForKeyPress();

   return 0;
}
```

B.3 Connect to Cameras via IP Address

Connect to cameras via its IP address and the related NIC's IP address.

ConnectSpecCamera.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;

// Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT stImageInfo = {0};
    while(1)
    {
        nRet = MV_CC_GetImageBuffer(pUser, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get Image Buffer: Width[%d], Height[%d], FrameNum[%d]\n",
                stImageInfo.stFrameInfo.nWidth,
stImageInfo.stFrameInfo.nHeight, stImageInfo.stFrameInfo.nFrameNum);

            nRet = MV_CC_FreeImageBuffer(pUser, &stImageInfo);
            if(nRet != MV_OK)
            {
                printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
            }
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    MV_CC_DEVICE_INFO stDevInfo = {0};
    MV_GIGE_DEVICE_INFO stGigEDev = {0};

    // The camera IP that needs to be connected (based on actual padding)
    printf("Please input Current Camera Ip : ");
    char nCurrentIp[128];
    scanf("%s", &nCurrentIp);
    // The PC IP that needs to be connected (based on actual padding)
    printf("Please input Net Export Ip : ");
    char nNetExport[128];
    scanf("%s", &nNetExport);
    unsigned int nIp1, nIp2, nIp3, nIp4, nIp;

    sscanf_s(nCurrentIp, "%d.%d.%d.%d", &nIp1, &nIp2, &nIp3, &nIp4);
    nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;
    stGigEDev.nCurrentIp = nIp;

    sscanf_s(nNetExport, "%d.%d.%d.%d", &nIp1, &nIp2, &nIp3, &nIp4);
    nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;
    stGigEDev.nNetExport = nIp;
    stDevInfo.nTLayerType = MV_GIGE_DEVICE;// Only support GigE camera
    stDevInfo.SpecialInfo.stGigEInfo = stGigEDev;

    do
    {
        // Select a device and create a handle
        nRet = MV_CC_CreateHandle(&handle, &stDevInfo);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet[0x%x]\n", nRet);
            break;
        }

        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Detect the optimal network packet (It only works for the GigE camera)
        if (stDevInfo.nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet = 
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!", 
nPacketSize);
            }
        }

        // Set trigger mode to off
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , 
handle, 0 , &nThreadID );
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Close the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
        handle = NULL;
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.4 Convert Pixel Format

Convert images to the desired pixel format, such as Mono, Bayer.

ConvertPixelType.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include "MvCameraControl.h"

// Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        // Print current IP address and user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

bool IsColor(MvGvspPixelType enType)
{
    switch(enType)
    {
    case PixelType_Gvsp_BGR8_Packed:
    case PixelType_Gvsp_YUV422_Packed:
    case PixelType_Gvsp_YUV422_YUYV_Packed:
    case PixelType_Gvsp_BayerGR8:
    case PixelType_Gvsp_BayerRG8:
    case PixelType_Gvsp_BayerGB8:
    case PixelType_Gvsp_BayerBG8:
    case PixelType_Gvsp_BayerGB10:
    case PixelType_Gvsp_BayerGB10_Packed:
    case PixelType_Gvsp_BayerBG10:
    case PixelType_Gvsp_BayerBG10_Packed:
    case PixelType_Gvsp_BayerRG10:
    case PixelType_Gvsp_BayerRG10_Packed:
    case PixelType_Gvsp_BayerGR10:
    case PixelType_Gvsp_BayerGR10_Packed:
    case PixelType_Gvsp_BayerGB12:
    case PixelType_Gvsp_BayerGB12_Packed:
    case PixelType_Gvsp_BayerBG12:
    case PixelType_Gvsp_BayerBG12_Packed:
    case PixelType_Gvsp_BayerRG12:
    case PixelType_Gvsp_BayerRG12_Packed:
    case PixelType_Gvsp_BayerGR12:
    case PixelType_Gvsp_BayerGR12_Packed:
        return true;
    default:
        return false;
    }
```

```
}

bool IsMono(MvGvspPixelType enType)
{
    switch(enType)
    {
    case PixelType_Gvsp_Mono10:
    case PixelType_Gvsp_Mono10_Packed:
    case PixelType_Gvsp_Mono12:
    case PixelType_Gvsp_Mono12_Packed:
        return true;
    default:
        return false;
    }
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char *pConvertData = NULL;
    unsigned int nConvertDataSize = 0;

    do
    {
        // Enumerate devices
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        // Select a device and create a handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Detect the optimal network packet size (It only works for the GigE
camera)
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        MV_FRAME_OUT stImageInfo = {0};

        nRet = MV_CC_GetImageBuffer(handle, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.stFrameInfo.nWidth,
stImageInfo.stFrameInfo.nHeight, stImageInfo.stFrameInfo.nFrameNum);

            MvGvspPixelType enDstPixelType = PixelType_Gvsp_Undefined;
            unsigned int nChannelNum = 0;
            char chFileName[MAX_PATH] = {0};
            //If it is color, convert it to RGB8
            if (IsColor(stImageInfo.stFrameInfo.enPixelType))
            {
                nChannelNum = 3;
                enDstPixelType = PixelType_Gvsp_RGB8_Packed;
                sprintf(chFileName, "AfterConvert.rgb");
            }
            //If it is Mono, convert it to Mono8
            else if (IsMono(stImageInfo.stFrameInfo.enPixelType))
            {
                nChannelNum = 1;
                enDstPixelType = PixelType_Gvsp_Mono8;
                sprintf(chFileName, "AfterConvert.gray");
            }
            else
            {
                printf("Don't need to convert!\n");
            }

            if (enDstPixelType != PixelType_Gvsp_Undefined)
            {
                pConvertData = (unsigned
char*)malloc(stImageInfo.stFrameInfo.nWidth * stImageInfo.stFrameInfo.nHeight *
nChannelNum);
                if (NULL == pConvertData)
                {
                    printf("malloc pConvertData fail!\n");
                    nRet = MV_E_RESOURCE;
                    break;
                }
                nConvertDataSize = stImageInfo.stFrameInfo.nWidth * 
stImageInfo.stFrameInfo.nHeight * nChannelNum;

                // Convert pixel format
                MV_CC_PIXEL_CONVERT_PARAM stConvertParam = {0};

                stConvertParam.nWidth = 
stImageInfo.stFrameInfo.nWidth;                 // Image width
                stConvertParam.nHeight = 
stImageInfo.stFrameInfo.nHeight;               // Image height
                stConvertParam.pSrcData = 
stImageInfo.pBufAddr;                         // Input data buffer
                stConvertParam.nSrcDataLen = 
stImageInfo.stFrameInfo.nFrameLen;         // Input data size
                stConvertParam.enSrcPixelType = 
stImageInfo.stFrameInfo.enPixelType;    // Input pixel format
                stConvertParam.enDstPixelType = 
enDstPixelType;                         // Output pixel format
                stConvertParam.pDstBuffer = 
pConvertData;                               // Output data buffer
                stConvertParam.nDstBufferSize = 
nConvertDataSize;                       // Output buffer size
                nRet = MV_CC_ConvertPixelType(handle, &stConvertParam);
                if (MV_OK != nRet)
                {
                    printf("Convert Pixel Type fail! nRet [0x%x]\n", nRet);
                    break;
                }

                FILE* fp = NULL;
                errno_t err = fopen_s(&fp, chFileName, "wb");
                if (0 != err || NULL == fp)
                {
                    printf("Open file failed\n");
                    nRet = MV_E_RESOURCE;
                    break;
                }
                fwrite(stConvertParam.pDstBuffer, 1, stConvertParam.nDstLen, 
fp);
                fclose(fp);
                printf("Convert pixeltype succeed\n");
            }
            MV_CC_FreeImageBuffer(handle, &stImageInfo);
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }

        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Close the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (pConvertData)
    {
        free(pConvertData);
        pConvertData = NULL;
    }

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.5 Call APIs Dynamically

The sample code below shows how to call APIs dynamically to implement functions.

DynamicallyLoadDLL.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "CameraParams.h"
#include "MvErrorDefine.h"

typedef unsigned int    (__stdcall * DLL_GetSDKVersion)         ();
typedef int             (__stdcall * DLL_EnumerateTls)          ();
typedef int             (__stdcall * DLL_EnumDevices)           (IN unsigned
int nTLayerType, IN OUT MV_CC_DEVICE_INFO_LIST * pstDevList);
typedef int             (__stdcall * DLL_EnumDevicesEx)         (IN unsigned
int nTLayerType, IN OUT MV_CC_DEVICE_INFO_LIST* pstDevList, IN const char*
pManufacturerName);
typedef bool            (__stdcall * DLL_IsDeviceAccessible)    (IN
MV_CC_DEVICE_INFO* pstDevInfo, IN unsigned int nAccessMode);
typedef int             (__stdcall * DLL_CreateHandle)          (OUT void **
handle, IN const MV_CC_DEVICE_INFO* pstDevInfo);
typedef int             (__stdcall * DLL_CreateHandleWithoutLog)(OUT void **
handle, IN const MV_CC_DEVICE_INFO* pstDevInfo);
typedef int             (__stdcall * DLL_DestroyHandle)         (IN void *
handle);
typedef int             (__stdcall * DLL_OpenDevice)            (IN void*
handle, IN unsigned int nAccessMode, IN unsigned short nSwitchoverKey);
typedef int             (__stdcall * DLL_CloseDevice)           (IN void*
handle);
typedef int             (__stdcall * DLL_RegisterImageCallBackEx)(void* handle,
void(__stdcall* cbOutput)(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser),void* pUser);
typedef int             (__stdcall * DLL_RegisterImageCallBackForRGB)(void*
handle, void(__stdcall* cbOutput)(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser), void* pUser);
typedef int             (__stdcall * DLL_RegisterImageCallBackForBGR)(void*
handle, void(__stdcall* cbOutput)(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser), void* pUser);
typedef int             (__stdcall * DLL_StartGrabbing)         (IN void*
handle);
typedef int             (__stdcall * DLL_StopGrabbing)          (IN void*
handle);
typedef int             (__stdcall * DLL_GetImageForRGB)        (IN void*
handle, IN OUT unsigned char * pData , IN unsigned int nDataSize, IN OUT
MV_FRAME_OUT_INFO_EX* pFrameInfo, int nMsec);
typedef int             (__stdcall * DLL_GetImageForBGR)        (IN void*
handle, IN OUT unsigned char * pData , IN unsigned int nDataSize, IN OUT
MV_FRAME_OUT_INFO_EX* pFrameInfo, int nMsec);
typedef int             (__stdcall * DLL_GetImageBuffer)        (IN void*
handle, MV_FRAME_OUT* pFrameInfo, int nMsec);
typedef int             (__stdcall * DLL_FreeImageBuffer)       (IN void*
handle, MV_FRAME_OUT* pFrameInfo);
```

```
typedef int              (__stdcall * DLL_GetOneFrameTimeout)    (IN void*
handle, IN OUT unsigned char * pData , IN unsigned int nDataSize, IN OUT
MV_FRAME_OUT_INFO_EX* pFrameInfo, int nMsec);
typedef int              (__stdcall * DLL_Display)               (IN void*
handle, void* hWnd);
typedef int              (__stdcall * DLL_DisplayOneFrame)       (IN void*
handle, IN MV_DISPLAY_FRAME_INFO* pDisplayInfo);
typedef int              (__stdcall * DLL_SetImageNodeNum)       (IN void*
handle, unsigned int nNum);
typedef int              (__stdcall * DLL_GetDeviceInfo)         (IN void *
handle, IN OUT MV_CC_DEVICE_INFO* pstDevInfo);
typedef int              (__stdcall * DLL_GetAllMatchInfo)       (IN void*
handle, IN OUT MV_ALL_MATCH_INFO* pstInfo);

typedef int              (__stdcall * DLL_GetIntValueEx)         (IN void*
handle,IN const char* strKey,OUT MVCC_INTVALUE_EX *pIntValue);
typedef int              (__stdcall * DLL_SetIntValueEx)         (IN void*
handle,IN const char* strKey,IN int64_t nValue);
typedef int              (__stdcall * DLL_GetEnumValue)          (IN void*
handle,IN const char* strValue,OUT MVCC_ENUMVALUE *pEnumValue);
typedef int              (__stdcall * DLL_SetEnumValue)          (IN void*
handle,IN const char* strValue,IN unsigned int nValue);
typedef int              (__stdcall * DLL_SetEnumValueByString)  (IN void*
handle,IN const char* strValue,IN const char* sValue);
typedef int              (__stdcall * DLL_GetFloatValue)         (IN void*
handle,IN const char* strValue,OUT MVCC_FLOATVALUE *pFloatValue);
typedef int              (__stdcall * DLL_SetFloatValue)         (IN void*
handle,IN const char* strValue,IN float fValue);
typedef int              (__stdcall * DLL_GetBoolValue)          (IN void*
handle,IN const char* strValue,OUT bool *pBoolValue);
typedef int              (__stdcall * DLL_SetBoolValue)          (IN void*
handle,IN const char* strValue,IN bool bValue);
typedef int              (__stdcall * DLL_GetStringValue)        (IN void*
handle,IN const char* strKey,OUT MVCC_STRINGVALUE *pStringValue);
typedef int              (__stdcall * DLL_SetStringValue)        (IN void*
handle,IN const char* strKey,IN const char * sValue);
typedef int              (__stdcall * DLL_SetCommandValue)       (IN void*
handle,IN const char* strValue);
typedef int              (__stdcall * DLL_LocalUpgrade)          (IN void*
handle, const void *pFilePathName);
typedef int              (__stdcall * DLL_GetUpgradeProcess)     (IN void*
handle, unsigned int* pnProcess);
typedef int              (__stdcall * DLL_GetOptimalPacketSize)  (IN void*
handle);
typedef int              (__stdcall * DLL_ReadMemory)            (IN void*
handle , void *pBuffer, int64_t nAddress, int64_t nLength);
typedef int              (__stdcall * DLL_WriteMemory)           (IN void*
handle , const void *pBuffer, int64_t nAddress, int64_t nLength);
typedef int              (__stdcall * DLL_RegisterExceptionCallBack)(IN void*
handle, void(__stdcall* cbException)(unsigned int nMsgType, void* pUser), void*
pUser);
typedef int              (__stdcall * DLL_RegisterEventCallBackEx)(void* handle,
const char* pEventName, void(__stdcall* cbEvent)(MV_EVENT_OUT_INFO *
pEventInfo, void* pUser),void* pUser);
typedef int             (__stdcall * DLL_RegisterAllEventCallBack)(void*
handle, void(__stdcall* cbEvent)(MV_EVENT_OUT_INFO * pEventInfo, void*
pUser),void* pUser);

typedef int             (__stdcall * DLL_ForceIpEx)             (IN void*
handle, unsigned int nIP, unsigned int nSubNetMask, unsigned int
nDefaultGateWay);
typedef int             (__stdcall * DLL_SetIpConfig)           (IN void*
handle, unsigned int nType);
typedef int             (__stdcall * DLL_SetNetTransMode)       (IN void*
handle, unsigned int nType);
typedef int             (__stdcall * DLL_GetNetTransInfo)       (IN void*
handle, MV_NETTRANS_INFO* pstInfo);
typedef int             (__stdcall * DLL_SetGvcpTimeout)        (IN void*
handle, unsigned int nMillisec);
typedef int                (__stdcall * DLL_SetResend)             (IN void*
handle, unsigned int bEnable, unsigned int nMaxResendPercent, unsigned int
nResendTimeout);
typedef int                (__stdcall * DLL_SetTransmissionType)   (IN void*
handle, MV_TRANSMISSION_TYPE * pstTransmissionType);
typedef int                (__stdcall * DLL_IssueActionCommand)    (IN
MV_ACTION_CMD_INFO* pstActionCmdInfo, OUT MV_ACTION_CMD_RESULT_LIST*
pstActionCmdResults);

typedef int                (__stdcall * DLL_SetDeviceBauderate)    (IN void*
handle, unsigned int nBaudrate);
typedef int                (__stdcall * DLL_GetDeviceBauderate)    (IN void*
handle,unsigned int* pnCurrentBaudrate);
typedef int                (__stdcall * DLL_GetSupportBauderates)  (IN void*
handle,unsigned int* pnBaudrateAblity);
typedef int                (__stdcall * DLL_SetGenCPTimeOut)       (IN void*
handle, unsigned int nMillisec);

typedef int             (__stdcall * DLL_GetGenICamXML)         (IN void*
handle, IN OUT unsigned char* pData, IN unsigned int nDataSize, OUT unsigned
int* pnDataLen);
typedef int             (__stdcall * DLL_SaveImageEx2)          (IN void*
handle, IN OUT MV_SAVE_IMAGE_PARAM_EX* pSaveParam);
typedef int             (__stdcall * DLL_ConvertPixelType)      (IN void*
handle, IN OUT MV_CC_PIXEL_CONVERT_PARAM* pstCvtParam);
typedef int                (__stdcall * DLL_SetBayerCvtQuality)    (IN void*
handle, IN unsigned int BayerCvtQuality);
typedef int                (__stdcall * DLL_FeatureSave)           (IN void*
handle, IN const char* pFileName);
typedef int                (__stdcall * DLL_FeatureLoad)           (IN void*
handle, IN const char* pFileName);
typedef int                (__stdcall * DLL_FileAccessRead)        (IN void*
handle, IN MV_CC_FILE_ACCESS * pstFileAccess);
typedef int                (__stdcall * DLL_FileAccessWrite)       (IN void*
handle, IN MV_CC_FILE_ACCESS * pstFileAccess);
```

```
typedef int                (__stdcall * DLL_GetFileAccessProgress) (IN void*
handle, OUT MV_CC_FILE_ACCESS_PROGRESS * pstFileAccessProgress);
typedef int                (__stdcall * DLL_StartRecord)           (IN void*
handle, IN MV_CC_RECORD_PARAM* pstRecordParam);
typedef int                (__stdcall * DLL_InputOneFrame)         (IN void*
handle, IN MV_CC_INPUT_FRAME_INFO * pstInputFrameInfo);
typedef int                (__stdcall * DLL_StopRecord)            (IN void*
handle);

bool g_bExit = false;

struct MultiThrParam
{
    void *pUser;
    HINSTANCE hDll;
};

// Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        // Print current IP address and the user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall WorkThread(void* stMuthreadPar)
{
    struct MultiThrParam *stMulThrPar = (struct MultiThrParam*)stMuthreadPar;
    int nRet = MV_OK;
    MV_FRAME_OUT stOutFrame = {0};

    DLL_GetImageBuffer DLLGetImageBuffer = 
(DLL_GetImageBuffer)GetProcAddress(stMulThrPar->hDll, "MV_CC_GetImageBuffer");
    DLL_FreeImageBuffer DLLFreeImageBuffer = 
(DLL_FreeImageBuffer)GetProcAddress(stMulThrPar->hDll, "MV_CC_FreeImageBuffer");

    while(true)
    {

        nRet = DLLGetImageBuffer(stMulThrPar->pUser, &stOutFrame, 1000);
        if (nRet == MV_OK)
        {
            printf("Get Image Buffer: Width[%d], Height[%d], nFrameNum[%d]\n",
                stOutFrame.stFrameInfo.nWidth, stOutFrame.stFrameInfo.nHeight, 
stOutFrame.stFrameInfo.nFrameNum);

            nRet = DLLFreeImageBuffer(stMulThrPar->pUser, &stOutFrame);
            if(nRet != MV_OK)
            {
                printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
            }
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    HINSTANCE MvCamCtrlDll = NULL;
    // Dynamic link library default path (System Disk:\Program Files
(x86)\Common Files\MVS\Runtime)
    MvCamCtrlDll = LoadLibrary("MvCameraControl.dll");
    if (MvCamCtrlDll)
    {
        do
        {
            // Enumerate devices
            MV_CC_DEVICE_INFO_LIST stDeviceList;
            memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
            DLL_EnumDevices DLLEnumDevices = (DLL_EnumDevices) GetProcAddress
(MvCamCtrlDll,"MV_CC_EnumDevices");
            nRet = DLLEnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE,
&stDeviceList);
            if (MV_OK != nRet)
            {
                printf("Enum Devices fail! nRet [0x%x]\n", nRet);
                break;
            }

            if (stDeviceList.nDeviceNum > 0)
            {
                for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
                {
                    printf("[device %d]:\n", i);
                    MV_CC_DEVICE_INFO* pDeviceInfo =
stDeviceList.pDeviceInfo[i];
                    if (NULL == pDeviceInfo)
                    {
                        break;
                    }
                    PrintDeviceInfo(pDeviceInfo);
                }
            }
            else
            {
                printf("Find No Devices!\n");
                break;
            }

            printf("Please Input camera index:");
            unsigned int nIndex = 0;
            scanf_s("%d", &nIndex);
            if (nIndex >= stDeviceList.nDeviceNum)
            {
                printf("Input error!\n");
                break;
            }

            // Select a device and create a handle
            DLL_CreateHandle DLLCreateHandle =
(DLL_CreateHandle)GetProcAddress(MvCamCtrlDll, "MV_CC_CreateHandle");
            nRet = DLLCreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
            if (MV_OK != nRet)
            {
                printf("Create Handle fail! nRet [0x%x]\n", nRet);
                break;
            }

            // Open the device
            DLL_OpenDevice DLLOpenDevice =
(DLL_OpenDevice)GetProcAddress(MvCamCtrlDll, "MV_CC_OpenDevice");
            nRet = DLLOpenDevice(handle, MV_ACCESS_Exclusive, 0);
            if (MV_OK != nRet)
            {
                printf("Open Device fail! nRet [0x%x]\n", nRet);
                break;
            }

            // Set trigger mode to off
            DLL_SetEnumValue DLLSetEnumValue =
(DLL_SetEnumValue)GetProcAddress(MvCamCtrlDll, "MV_CC_SetEnumValue");
            nRet = DLLSetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
            if (MV_OK != nRet)
            {
                printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
                break;
            }

            // Start grabbing image
            DLL_StartGrabbing DLLStartGrabbing =
(DLL_StartGrabbing)GetProcAddress(MvCamCtrlDll, "MV_CC_StartGrabbing");
            nRet = DLLStartGrabbing(handle);
            if (MV_OK != nRet)
            {
                printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
                break;
            }

            unsigned int nThreadID = 0;
            MultiThrParam stMuthreadPar;
            stMuthreadPar.pUser = handle;
            stMuthreadPar.hDll = MvCamCtrlDll;
            void* hThreadHandle = (void*) _beginthreadex( NULL , 0 ,
WorkThread , (void*)&stMuthreadPar, 0 , &nThreadID );
            if (NULL == hThreadHandle)
            {
                break;
            }

            printf("Press a key to stop grabbing.\n");
            WaitForKeyPress();

            g_bExit = true;
            Sleep(1000);

            // Stop grabbing image
            DLL_StopGrabbing DLLStopGrabbing =
(DLL_StartGrabbing)GetProcAddress(MvCamCtrlDll, "MV_CC_StopGrabbing");
            nRet = DLLStopGrabbing(handle);
            if (MV_OK != nRet)
            {
                printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
                break;
            }

            // Close the device
            DLL_CloseDevice DLLCloseDevice =
(DLL_CloseDevice)GetProcAddress(MvCamCtrlDll, "MV_CC_CloseDevice");
            nRet = DLLCloseDevice(handle);
            if (MV_OK != nRet)
            {
                printf("ClosDevice fail! nRet [0x%x]\n", nRet);
                break;
            }

            // Destroy the handle
            DLL_DestroyHandle DLLDestroyHandle =
(DLL_DestroyHandle)GetProcAddress(MvCamCtrlDll, "MV_CC_DestroyHandle");
            nRet = DLLDestroyHandle(handle);
            if (MV_OK != nRet)
            {
                printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
                break;
            }
        } while (0);


        if (nRet != MV_OK)
        {
            if (handle != NULL)
            {
                DLL_DestroyHandle DLLDestroyHandle =
(DLL_DestroyHandle)GetProcAddress(MvCamCtrlDll, "MV_CC_DestroyHandle");
                nRet = DLLDestroyHandle(handle);
                handle = NULL;
            }
        }
        FreeLibrary(MvCamCtrlDll);
        printf("Press a key to exit.\n");
        WaitForKeyPress();
    }
    else
    {
        DWORD errCode = 0;
        errCode = GetLastError();
        printf("error code %ld!\n",errCode);
        printf("Press a key to exit.\n");
        WaitForKeyPress();
        return -1;
    }
    return 0;
}
```

B.6 Get Camera Events

The sample code below show how to configure camera events, register the event callback function and handle events in callback function.

Events.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include "MvCameraControl.h"

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
```

```
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

void __stdcall EventCallBack(MV_EVENT_OUT_INFO * pEventInfo, void* pUser)
{
    if (pEventInfo)
    {
        __int64 nBlockId = pEventInfo->nBlockIdHigh;
        nBlockId = (nBlockId << 32) + pEventInfo->nBlockIdLow;

        __int64 nTimestamp = pEventInfo->nTimestampHigh;
        nTimestamp = (nTimestamp << 32) + pEventInfo->nTimestampLow;

        printf("EventName[%s], EventID[%u], BlockId[%I64d], Timestamp[%I64d]
\n",
            pEventInfo->EventName, pEventInfo->nEventID,nBlockId,nTimestamp);
    }
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetEnumValueByString(handle,"EventSelector","ExposureEnd");
        if (MV_OK != nRet)
        {
            printf("Set Event Selector fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetEnumValueByString(handle,"EventNotification","On");
        if (MV_OK != nRet)
        {
            printf("Set Event Notification fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_RegisterEventCallBackEx(handle, "ExposureEnd",
EventCallBack, handle);
        if (MV_OK != nRet)
        {
            printf("Register Event CallBack fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.7 Get Images Via Precision Time Protocol

he sample code below shows how to get images via precision time protocol.

Grab_ActionCommand.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;
bool g_bReceive = false;
unsigned int g_nPayloadSize = 0;
unsigned int g_DeviceKey = 1;
unsigned int g_GroupKey = 1;
unsigned int g_GroupMask= 1;

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}

static  unsigned int __stdcall ActionCommandWorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_ACTION_CMD_INFO stActionCmdInfo = {0};
    MV_ACTION_CMD_RESULT_LIST stActionCmdResults = {0};

    stActionCmdInfo.nDeviceKey = g_DeviceKey;
    stActionCmdInfo.nGroupKey = g_GroupKey;
    stActionCmdInfo.nGroupMask = g_GroupMask;
    stActionCmdInfo.pBroadcastAddress = "255.255.255.255";
    stActionCmdInfo.nTimeOut = 100;
    stActionCmdInfo.bActionTimeEnable = 0;

    while(!g_bExit)
    {
        nRet = MV_GIGE_IssueActionCommand(&stActionCmdInfo,&stActionCmdResults);
        if (MV_OK != nRet)
        {
            printf("Issue Action Command fail! nRet [0x%x]\n", nRet);
            continue;
        }
        printf("NumResults = %d\r\n",stActionCmdResults.nNumResults);

        MV_ACTION_CMD_RESULT* pResults = stActionCmdResults.pResults;
        for (unsigned int i = 0;i < stActionCmdResults.nNumResults;i++)
        {
            printf("Ip == %s, Status == 0x%x\r\n",pResults-
>strDeviceAddress,pResults->nStatus);
            pResults++;
        }
    }

    return 0;
}

static  unsigned int __stdcall ReceiveImageWorkThread(void* pUser)
{
    int nRet = MV_OK;

    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) *
(g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }

    free(pData);

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet = 
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!", 
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 1);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerSource", 9);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Source fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetIntValue(handle, "ActionDeviceKey", g_DeviceKey);
        if (MV_OK != nRet)
        {
            printf("Set Action Device Key fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetIntValue(handle, "ActionGroupKey", g_GroupKey);
        if (MV_OK != nRet)
        {
            printf("Set Action Group Key fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_SetIntValue(handle, "ActionGroupMask", g_GroupMask);
        if (MV_OK != nRet)
        {
            printf("Set Action Group Mask fail! nRet [0x%x]\n", nRet);
            break;
        }

        MVCC_INTVALUE stParam;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        g_nPayloadSize = stParam.nCurValue;

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nReceiveImageThreadID = 0;
        void* hReceiveImageThreadHandle = (void*) _beginthreadex( NULL , 0 ,
ReceiveImageWorkThread , handle, 0 , &nReceiveImageThreadID );
        if (NULL == hReceiveImageThreadHandle)
        {
            break;
        }

        unsigned int nActionCommandThreadID = 0;
        void* hActionCommandThreadHandle = (void*) _beginthreadex( NULL , 0 ,
ActionCommandWorkThread , NULL, 0 , &nActionCommandThreadID );
        if (NULL == hActionCommandThreadHandle)
        {
            return 0;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.8 Get Images in Callback Function

The sample code below shows how to get images by registering the image callback function.

Grab_Callback.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include "MvCameraControl.h"

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

void __stdcall ImageCallBackEx(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
            pFrameInfo->nWidth, pFrameInfo->nHeight, pFrameInfo->nFrameNum);
    }
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet = 
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!", 
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBackEx, handle);
        if (MV_OK != nRet)
        {
            printf("Register Image CallBack fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.9 Get Images Directly

The sample code below shows how to get images directly.

GrabImage.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;

// Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);
```

```
        // Print current IP address and the user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT stOutFrame = {0};

    while(true)
    {
        nRet = MV_CC_GetImageBuffer(pUser, &stOutFrame, 1000);
        if (nRet == MV_OK)
        {
            printf("Get Image Buffer: Width[%d], Height[%d], FrameNum[%d]\n",
                stOutFrame.stFrameInfo.nWidth, stOutFrame.stFrameInfo.nHeight,
stOutFrame.stFrameInfo.nFrameNum);

            nRet = MV_CC_FreeImageBuffer(pUser, &stOutFrame);
            if(nRet != MV_OK)
            {
                printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
            }
        }
        else
        {
            printf("Get Image fail! nRet [0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        // Enumerate devices
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index(0-%d):", stDeviceList.nDeviceNum-1);
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        // Select the device and create a handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Detect the network optimal package size (It only works for the GigE
camera)
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        // Set the trigger mode to off
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        //Get the enumerator name according to the node name and assigned value
        MVCC_ENUMVALUE stEnumValue = {0};
        MVCC_ENUMENTRY stEnumEntry = {0};
        nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &stEnumValue);
        if (MV_OK != nRet)
        {
            printf("Get PixelFormat's value fail! nRet [0x%x]\n", nRet);
            break;
        }
        stEnumEntry.nValue = stEnumValue.nCurValue;
        nRet = MV_CC_GetEnumEntrySymbolic(handle, "PixelFormat", &stEnumEntry);
        if (MV_OK != nRet)
        {
            printf("Get PixelFormat's symbol fail! nRet [0x%x]\n", nRet);
            break;
        }
        else
        {
            printf("PixelFormat:%s\n", stEnumEntry.chSymbolic);
        }

        // Start acquiring images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , 
handle, 0 , &nThreadID );
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        // Stop acquiring image(s)
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Close the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.10 Get Images Directly with High Performance

The sample code shows how to get images directly with high performance.

GrabImage_HighPerformance.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;

    MV_FRAME_OUT stOutFrame = {0};
    memset(&stOutFrame, 0, sizeof(MV_FRAME_OUT));

    while(1)
    {
        nRet = MV_CC_GetImageBuffer(pUser, &stOutFrame, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                stOutFrame.stFrameInfo.nWidth, stOutFrame.stFrameInfo.nHeight,
stOutFrame.stFrameInfo.nFrameNum);
```

```
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(NULL != stOutFrame.pBufAddr)
        {
            nRet = MV_CC_FreeImageBuffer(pUser, &stOutFrame);
            if(nRet != MV_OK)
            {
                printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
            }
        }
        if(g_bExit)
        {
            break;
        }
    }

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        MVCC_INTVALUE stParam;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        g_nPayloadSize = stParam.nCurValue;

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread ,
handle, 0 , &nThreadID );
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.11 Display the Acquired Images

Display each acquired image.

GrabImage_Display.cpp

```
#include <stdio.h>
#include <process.h>
#include <conio.h>
#include "windows.h"
#include "MvCameraControl.h"

HWND g_hwnd = NULL;
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch(msg)
    {
    case WM_DESTROY:
        PostQuitMessage(0);
        g_hwnd = NULL;
        break;
    }

    return DefWindowProc(hWnd, msg, wParam, lParam);
}

static  unsigned int __stdcall CreateRenderWindow(void* pUser)
{
    HINSTANCE hInstance = ::GetModuleHandle(NULL);
    WNDCLASSEX wc;
    wc.cbSize           = sizeof(wc);
    wc.style            = CS_HREDRAW | CS_VREDRAW;
    wc.cbClsExtra       = 0;
    wc.cbWndExtra       = 0;
    wc.hInstance        = hInstance;
    wc.hIcon            = ::LoadIcon(NULL, IDI_APPLICATION);
    wc.hIconSm          = ::LoadIcon( NULL, IDI_APPLICATION );
    wc.hbrBackground    = (HBRUSH)( COLOR_WINDOW + 1);
    wc.hCursor          = ::LoadCursor(NULL, IDC_ARROW);
    wc.lpfnWndProc      = WndProc;
    wc.lpszMenuName     = NULL;
    wc.lpszClassName    = "RenderWindow";

    if(!RegisterClassEx(&wc))
    {
        return 0;
    }

    DWORD style = WS_OVERLAPPEDWINDOW;
    DWORD styleEx = WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
    RECT rect = {0, 0, 640, 480};

    AdjustWindowRectEx(&rect, style, false, styleEx);

    HWND hWnd = CreateWindowEx(styleEx, "RenderWindow", "Display", style, 0, 0,
        rect.right - rect.left, rect.bottom - rect.top, NULL, NULL, hInstance,
NULL);
    if(hWnd == NULL)
    {
        return 0;
    }

    ::UpdateWindow(hWnd);
    ::ShowWindow(hWnd, SW_SHOW);

    g_hwnd = hWnd;

    MSG msg = {0};
    while(msg.message != WM_QUIT)
    {
        if(PeekMessage(&msg, 0, 0, 0, PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return 0;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    MV_DISPLAY_FRAME_INFO stDisplayInfo = {0};
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * 
(g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;

    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 
1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n", 
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);

            if (g_hwnd)
            {
                stDisplayInfo.hWnd = g_hwnd;
                stDisplayInfo.pData = pData;
                stDisplayInfo.nDataLen = stImageInfo.nFrameLen;
                stDisplayInfo.nWidth = stImageInfo.nWidth;
                stDisplayInfo.nHeight = stImageInfo.nHeight;
                stDisplayInfo.enPixelType = stImageInfo.enPixelType;

                MV_CC_DisplayOneFrame(pUser, &stDisplayInfo);
            }
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }

    free(pData);

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet = 
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!", 
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        MVCC_INTVALUE stParam = {0};
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        g_nPayloadSize = stParam.nCurValue;

        unsigned int nThreadID = 0;
        void* hCreateWindow = (void*) _beginthreadex( NULL , 0 , 
CreateRenderWindow , handle, 0 , &nThreadID);
        if (NULL == hCreateWindow)
        {
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread ,
handle, 0 , &nThreadID);
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        WaitForSingleObject(hThreadHandle, INFINITE);
        CloseHandle(hThreadHandle);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.12 Get Images by Strategy

The sample code shows how to get image by different strategies.

GrabStrategies.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }

    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
```

```
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall UpcomingThread(void* pUser)
{
    Sleep(3000);

    printf("Trigger Software Once for MV_GrabStrategy_UpcomingImage\n");
    MV_CC_SetCommandValue(pUser, "TriggerSoftware");

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char * pData = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                if(MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize) !=
MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValueByString(handle, "TriggerMode", "On");
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }
        nRet = MV_CC_SetEnumValueByString(handle, "TriggerSource", "Software");
        if (MV_OK != nRet)
        {
            printf("Set Trigger Source fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nImageNodeNum = 5;
        nRet = MV_CC_SetImageNodeNum(handle, nImageNodeNum);
        if (MV_OK != nRet)
        {
            printf("Set number of image node fail! nRet [0x%x]\n", nRet);
            break;
        }


printf("\n*****************************************************************
****\n");
        printf("* 0.MV_GrabStrategy_OneByOne;
1.MV_GrabStrategy_LatestImagesOnly;  *\n");
        printf("* 2.MV_GrabStrategy_LatestImages;
3.MV_GrabStrategy_UpcomingImage;     *\n");

printf("*******************************************************************
**\n");

        printf("Please Input Grab Strategy:");
        unsigned int nGrabStrategy = 0;
        scanf_s("%d", &nGrabStrategy);

        if (nGrabStrategy == MV_GrabStrategy_UpcomingImage && MV_USB_DEVICE ==
stDeviceList.pDeviceInfo[nIndex]->nTLayerType)
        {
            printf("U3V device not support UpcomingImage\n");
            break;
        }

        switch(nGrabStrategy)
        {
        case MV_GrabStrategy_OneByOne:
            {
                printf("Grab using the MV_GrabStrategy_OneByOne default strategy
\n");
                nRet = MV_CC_SetGrabStrategy(handle, MV_GrabStrategy_OneByOne);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_LatestImagesOnly:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImagesOnly
\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImagesOnly);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_LatestImages:
            {
                printf("Grab using strategy MV_GrabStrategy_LatestImages\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_LatestImages);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                nRet = MV_CC_SetOutputQueueSize(handle, 2);
                if (MV_OK != nRet)
                {
                    printf("Set Output Queue Size fail! nRet [0x%x]\n", nRet);
                    break;
                }
            }
            break;
        case MV_GrabStrategy_UpcomingImage:
            {
                printf("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
                nRet = MV_CC_SetGrabStrategy(handle,
MV_GrabStrategy_UpcomingImage);
                if (MV_OK != nRet)
                {
                    printf("Set Grab Strategy fail! nRet [0x%x]\n", nRet);
                    break;
                }

                unsigned int nThreadID = 0;
                void* hThreadHandle = (void*) _beginthreadex( NULL , 0 ,
UpcomingThread , handle, 0 , &nThreadID );
                if (NULL == hThreadHandle)
                {
                    break;
                }
            }
            break;
        default:
            printf("Input error!Use default strategy:MV_GrabStrategy_OneByOne
\n");
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        for (unsigned int i = 0;i < nImageNodeNum;i++)
        {
            nRet = MV_CC_SetCommandValue(handle, "TriggerSoftware");
            if (MV_OK != nRet)
            {
                printf("Send Trigger Software command fail! nRet [0x%x]\n",
nRet);
                break;
            }
            Sleep(500);
        }

        MV_FRAME_OUT stOutFrame = {0};
        if (nGrabStrategy != MV_GrabStrategy_UpcomingImage)
        {
            while(true)
            {
                nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 0);
                if (nRet == MV_OK)
                {
                    printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d]
\n",
                        stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);
                }
                else
                {
                    printf("No data[0x%x]\n", nRet);
                    break;
                }

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
        }
        else
        {
            nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 5000);
            if (nRet == MV_OK)
            {
                printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d]\n",
                    stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
            else
            {
                printf("No data[0x%x]\n", nRet);
            }
        }

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

B.13 Lossless Decoding

Decode lossless compression stream from the camera into raw data.

HighBandwidthDecode.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include "MvCameraControl.h"

#define IMAGE_NAME_LEN 256
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char * pData = NULL;
    unsigned char * pDstBuf = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        MVCC_INTVALUE stParam;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        unsigned int nPayloadSize = stParam.nCurValue;

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        MV_FRAME_OUT_INFO_EX stImageInfo = {0};
        memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));

        pData = (unsigned char *)malloc(sizeof(unsigned char) * (nPayloadSize));
        if (NULL == pData)
        {
            printf("malloc pData fail !\n");
            break;
        }

        pDstBuf = (unsigned char *)malloc(sizeof(unsigned char) *
(nPayloadSize));
        if (NULL == pDstBuf)
        {
            printf("malloc pDstData fail !\n");
            break;
        }

        unsigned int nImageNum = 10;
        char chImageName[IMAGE_NAME_LEN] = {0};
        MV_CC_HB_DECODE_PARAM stDecodeParam = {0};

        for(unsigned int i = 0;i < nImageNum; i++)
        {
            nRet = MV_CC_GetOneFrameTimeout(handle, pData, nPayloadSize,
&stImageInfo, 1000);
            if (nRet == MV_OK)
            {
                printf("Get One Frame: Width[%d], Height[%d], FrameNum[%d],
PixelFormat[0x%x]\n",
                    stImageInfo.nWidth, stImageInfo.nHeight,
stImageInfo.nFrameNum, stImageInfo.enPixelType);

                stDecodeParam.pSrcBuf = pData;
                stDecodeParam.nSrcLen = stImageInfo.nFrameLen;
                stDecodeParam.pDstBuf = pDstBuf;
                stDecodeParam.nDstBufSize = nPayloadSize;
                nRet = MV_CC_HB_Decode(handle, &stDecodeParam);
                if (nRet != MV_OK)
                {
                    printf("Decode fail![0x%x]\n", nRet);
                    break;
                }

                FILE* fp = NULL;
                sprintf_s(chImageName, IMAGE_NAME_LEN, "Image_w%d_h%d_fn
%03d.raw", stDecodeParam.nWidth, stDecodeParam.nHeight, stImageInfo.nFrameNum);
                errno_t err = fopen_s(&fp, chImageName, "wb");
                if (0 != err || NULL == fp)
                {
                    printf("Open file failed\n");
                    nRet = MV_E_RESOURCE;
                    break;
                }
                fwrite(stDecodeParam.pDstBuf, 1, stDecodeParam.nDstBufLen, fp);
                fclose(fp);
                printf("Decode succeed\n");
            }
            else
            {
                printf("No data[0x%x]\n", nRet);
            }
        }

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if(pDstBuf)
    {
        free(pDstBuf);
        pDstBuf = NULL;
    }

    if (pData)
    {
        free(pData);
        pData = NULL;
    }

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.14 Correct Lens Shading

The sample code shows how to correct lens shading.

LensShadingCorrection.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <conio.h>
#include <io.h>
#include "MvCameraControl.h"

// Waiting for key input.
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        // Print camera information such as IP and user ID.
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

unsigned char * g_pDstData = NULL;
unsigned int g_nDstDataSize = 0;

unsigned char * g_pCalibBuf = NULL;
unsigned int g_nCalibBufSize = 0;

bool g_IsNeedCalib = true;

void __stdcall ImageCallBackEx(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser)
{
    printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n", pFrameInfo-
>nWidth, pFrameInfo->nHeight, pFrameInfo->nFrameNum);

    int nRet = MV_OK;
    // Judge whether the camera needs calibration.
    if (true == g_IsNeedCalib)
    {
        // LSC calibration
        MV_CC_LSC_CALIB_PARAM stLSCCalib = {0};
        stLSCCalib.nWidth = pFrameInfo->nWidth;
        stLSCCalib.nHeight = pFrameInfo->nHeight;
        stLSCCalib.enPixelType = pFrameInfo->enPixelType;
        stLSCCalib.pSrcBuf = pData;
        stLSCCalib.nSrcBufLen = pFrameInfo->nFrameLen;

        if (g_pCalibBuf == NULL || g_nCalibBufSize < (pFrameInfo-
>nWidth*pFrameInfo->nHeight*sizeof(unsigned short)))
        {
            if (g_pCalibBuf)
            {
                free(g_pCalibBuf);
                g_pCalibBuf = NULL;
                g_nCalibBufSize = 0;
            }

            g_pCalibBuf = (unsigned char *)malloc(pFrameInfo->nWidth*pFrameInfo-
>nHeight*sizeof(unsigned short));
            if (g_pCalibBuf == NULL)
            {
                printf("malloc pCalibBuf fail !\n");
                return;
            }
            g_nCalibBufSize = pFrameInfo->nWidth*pFrameInfo-
>nHeight*sizeof(unsigned short);
        }

        stLSCCalib.pCalibBuf = g_pCalibBuf;
        stLSCCalib.nCalibBufSize = g_nCalibBufSize;

        stLSCCalib.nSecNumW = 689;
        stLSCCalib.nSecNumH = 249;
        stLSCCalib.nPadCoef = 2;
        stLSCCalib.nCalibMethod = 2;
        stLSCCalib.nTargetGray = 100;
        nRet = MV_CC_LSCCalib(pUser, &stLSCCalib);
        if (MV_OK != nRet)
        {
            printf("LSC Calib fail! nRet [0x%x]\n", nRet);
            return;
        }

        // Save calibration data file.
        FILE* fp_out = fopen("./LSCCalib.bin", "wb");
        if (NULL == fp_out)
        {
            return ;
        }
        fwrite(stLSCCalib.pCalibBuf, 1, stLSCCalib.nCalibBufLen, fp_out);
        fclose(fp_out);

        g_IsNeedCalib = false;
    }

    // LSC correction
    if (g_pDstData == NULL || g_nDstDataSize < pFrameInfo->nFrameLen)
    {
        if (g_pDstData)
        {
            free(g_pDstData);
            g_pDstData = NULL;
            g_nDstDataSize = 0;
        }

        g_pDstData = (unsigned char *)malloc(pFrameInfo->nFrameLen);
        if (g_pDstData == NULL)
        {
            printf("malloc pDstData fail !\n");
            return;
        }
        g_nDstDataSize = pFrameInfo->nFrameLen;
    }

    MV_CC_LSC_CORRECT_PARAM stLSCCorrectParam = {0};

    stLSCCorrectParam.nWidth = pFrameInfo->nWidth;
    stLSCCorrectParam.nHeight = pFrameInfo->nHeight;
    stLSCCorrectParam.enPixelType = pFrameInfo->enPixelType;
    stLSCCorrectParam.pSrcBuf = pData;
    stLSCCorrectParam.nSrcBufLen = pFrameInfo->nFrameLen;

    stLSCCorrectParam.pDstBuf = g_pDstData;
    stLSCCorrectParam.nDstBufSize = g_nDstDataSize;

    stLSCCorrectParam.pCalibBuf = g_pCalibBuf;
    stLSCCorrectParam.nCalibBufLen = g_nCalibBufSize;
    nRet = MV_CC_LSCCorrect(pUser, &stLSCCorrectParam);
    if (MV_OK != nRet)
    {
        printf("LSC Correct fail! nRet [0x%x]\n", nRet);
        return;
    }

    if (pFrameInfo->nFrameNum < 10)
    {
        // Save image to file.
        MV_SAVE_IMG_TO_FILE_PARAM stSaveFileParam;
        memset(&stSaveFileParam, 0, sizeof(MV_SAVE_IMG_TO_FILE_PARAM));

        stSaveFileParam.enImageType = MV_Image_Bmp;
        stSaveFileParam.enPixelType = pFrameInfo->enPixelType;
        stSaveFileParam.nWidth      = pFrameInfo->nWidth;
        stSaveFileParam.nHeight     = pFrameInfo->nHeight;
        stSaveFileParam.nDataLen    = pFrameInfo->nFrameLen;
        stSaveFileParam.pData       = pData;
        sprintf_s(stSaveFileParam.pImagePath, 256, "BeforeImage_w%d_h%d_fn
%03d.bmp", stSaveFileParam.nWidth, stSaveFileParam.nHeight, pFrameInfo-
>nFrameNum);
        nRet = MV_CC_SaveImageToFile(pUser, &stSaveFileParam);
        if (MV_OK != nRet)
        {
            printf("SaveImageToFile failed[%x]!\n", nRet);
            return;
        }

        stSaveFileParam.pData       = g_pDstData;
        sprintf_s(stSaveFileParam.pImagePath, 256, "AfterImage_w%d_h%d_fn
%03d.bmp", stSaveFileParam.nWidth, stSaveFileParam.nHeight, pFrameInfo-
>nFrameNum);
        nRet = MV_CC_SaveImageToFile(pUser, &stSaveFileParam);
        if (MV_OK != nRet)
        {
            printf("SaveImageToFile failed[%x]!\n", nRet);
            return;
        }
    }
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        // Enumerate devices.
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        // Create a handle for the device you specify.
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open the device.
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Get the optimal package size (GigE camera only).
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet = 
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        // Set trigger mode to Off.
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Judge whether it can be imported locally.
        FILE* fp = fopen("./LSCCalib.bin", "rb+");
        if (fp)
        {
            int nFileLen = filelength(fileno(fp));
            if (g_pCalibBuf == NULL || g_nCalibBufSize < nFileLen)
            {
                if (g_pCalibBuf)
                {
                    free(g_pCalibBuf);
                    g_pCalibBuf = NULL;
                    g_nCalibBufSize = 0;
                }

                g_pCalibBuf = (unsigned char *)malloc(nFileLen);
                if (g_pCalibBuf == NULL)
                {
                    printf("malloc pCalibBuf fail !\n");
                    break;
                }
                g_nCalibBufSize = nFileLen;
            }
            fread(g_pCalibBuf, 1, g_nCalibBufSize, fp);
            fclose(fp);

            g_IsNeedCalib = false;
        }

        // Register image callback.
        nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBackEx, handle);
        if (MV_OK != nRet)
        {
            printf("Register Image CallBack fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Start image acquisition.
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        Sleep(1000);

        // Stop image acquisition.
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Close the device.
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Close Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Destroy device handle.
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (g_pCalibBuf)
    {
        free(g_pCalibBuf);
        g_pCalibBuf = NULL;
        g_nCalibBufSize = 0;
    }

    if (g_pDstData)
    {
        free(g_pDstData);
        g_pDstData = NULL;
        g_nDstDataSize = 0;
    }

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.15 Set Multicast Mode

Set the transport mode to multicast mode.

MultiCast.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;

    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) *
(g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;

    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }

    free(pData);

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }


        printf("Start multicast sample in (c)ontrol or in (m)onitor mode? (c/m)
\n");

        char key;

        do
        {
            scanf("%c", &key);
        }
        while ( (key != 'c') && (key != 'm') && (key != 'C') && (key != 'M'));

        bool monitorMode = (key == 'm') || (key == 'M');

        if (monitorMode)
        {
            nRet = MV_CC_OpenDevice(handle, MV_ACCESS_Monitor);
        }
        else
        {
            nRet = MV_CC_OpenDevice(handle, MV_ACCESS_Control);
        }
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE &&
false == monitorMode)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n", 
nPacketSize);
            }
        }

        MVCC_INTVALUE stParam;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        g_nPayloadSize = stParam.nCurValue;

        char strIp[] = "239.0.1.23";
        unsigned int nIp1, nIp2, nIp3, nIp4, nIp;
        sscanf_s(strIp, "%d.%d.%d.%d", &nIp1, &nIp2, &nIp3, &nIp4);
        nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;

        MV_TRANSMISSION_TYPE stTransmissionType;
        memset(&stTransmissionType, 0, sizeof(MV_TRANSMISSION_TYPE));

        stTransmissionType.enTransmissionType = MV_GIGE_TRANSTYPE_MULTICAST;
        stTransmissionType.nDestIp = nIp;
        stTransmissionType.nDestPort = 8787;
        nRet = MV_GIGE_SetTransmissionType(handle, &stTransmissionType);
        if (MV_OK != nRet)
        {
            printf("Set Transmission Type fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , 
handle, 0 , &nThreadID );
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.16 Open GUI of Camera Property Settings

The sample code below shows how to open the Graphical User Interface (GUI) for getting or setting camera parameters.

OpenParamsGUI

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

// Wait for key press
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        // Print current IP address and the user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        // Enumerate devices
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index(0-%d):", stDeviceList.nDeviceNum-1);
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }
        // Select the device and create a handle
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Open the camera property configuration GUI
        nRet = MV_CC_OpenParamsGUI(handle);
        if (MV_OK != nRet)
        {
            printf("Open Parameters Configuration GUI fail! nRet [0x%x]\n",
nRet);
            break;
        }

        printf("Press a key to close camera.\n");
        WaitForKeyPress();

        // Close the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.17 File Access

Export the User Set or DPC (Defective Pixel Correction) file of a connected camera to the local PC as a binary file, or import a binary file from the local PC to a connected camera.

ParametrizeCamera_FileAccess.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

unsigned int g_nMode = 0;
int g_nRet = MV_OK;
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);
```

```
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall ProgressThread(void* pUser)
{
    int nRet = MV_OK;
    MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = {0};

    while(1)
    {
        nRet = MV_CC_GetFileAccessProgress(pUser, &stFileAccessProgress);
        printf("State = 0x%x,Completed = %I64d,Total = %I64d\r
\n",nRet,stFileAccessProgress.nCompleted,stFileAccessProgress.nTotal);
        if (nRet != MV_OK || (stFileAccessProgress.nCompleted != 0 &&
stFileAccessProgress.nCompleted == stFileAccessProgress.nTotal))
        {
            break;
        }

        Sleep(50);
    }

    return 0;
}

static  unsigned int __stdcall FileAccessThread(void* pUser)
{
    MV_CC_FILE_ACCESS stFileAccess = {0};

    stFileAccess.pUserFileName = "UserSet1.bin";
    stFileAccess.pDevFileName = "UserSet1";
    if (1 == g_nMode)
    {
        g_nRet = MV_CC_FileAccessRead(pUser, &stFileAccess);
        if (MV_OK != g_nRet)
        {
            printf("File Access Read fail! nRet [0x%x]\n", g_nRet);
        }
    }
    else if (2 == g_nMode)
    {
        g_nRet = MV_CC_FileAccessWrite(pUser, &stFileAccess);
        if (MV_OK != g_nRet)
        {
            printf("File Access Write fail! nRet [0x%x]\n", g_nRet);
        }
    }

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }
        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        g_nMode = 1;
        printf("Read to file.\n");

        unsigned int nThreadID = 0;
        void* hReadHandle = (void*) _beginthreadex( NULL , 0 ,
FileAccessThread , handle, 0 , &nThreadID );
        if (NULL == hReadHandle)
        {
            break;
        }

        Sleep(5);

        nThreadID = 0;
        void* hReadProgressHandle = (void*) _beginthreadex( NULL , 0 ,
ProgressThread , handle, 0 , &nThreadID );
        if (NULL == hReadProgressHandle)
        {
            break;
        }

        WaitForMultipleObjects(1, &hReadHandle, TRUE, INFINITE);
        WaitForMultipleObjects(1, &hReadProgressHandle, TRUE, INFINITE);
        if (MV_OK == g_nRet)
        {
            printf("File Access Read Success!\n");
        }
        printf("\n");

        g_nMode = 2;
        printf("Write from file.\n");

        nThreadID = 0;
        void* hWriteHandle = (void*) _beginthreadex( NULL , 0 ,
FileAccessThread , handle, 0 , &nThreadID );
        if (NULL == hWriteHandle)
        {
            break;
        }

        Sleep(5);

        nThreadID = 0;
        void* hWriteProgressHandle = (void*) _beginthreadex( NULL , 0 ,
ProgressThread , handle, 0 , &nThreadID );
        if (NULL == hWriteProgressHandle)
        {
            break;
        }

        WaitForMultipleObjects(1, &hWriteHandle, TRUE, INFINITE);
        WaitForMultipleObjects(1, &hWriteProgressHandle, TRUE, INFINITE);
        if (MV_OK == g_nRet)
        {
            printf("File Access Write Success!\n");
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.18 Import/Export Camera Property File

Export the property configurations of the selected camera as a XML file to the local PC, and import the XML file from the local PC to the selected cameras to fast configure all its propertys without the inconvenience of configuring its property one by one.

ParametrizeCamera_LoadAndSave.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        printf("Start export the camera properties to the file\n");
        printf("Wait......\n");

        nRet = MV_CC_FeatureSave(handle, "FeatureFile.ini");
        if (MV_OK != nRet)
        {
            printf("Save Feature fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Finish export the camera properties to the file\n\n");

        printf("Start import the camera properties from the file\n");
        printf("Wait......\n");
        nRet = MV_CC_FeatureLoad(handle, "FeatureFile.ini");
        if (MV_OK != nRet)
        {
            printf("Load Feature fail! nRet [0x%x]\n", nRet);
            break;
        }
        printf("Finish import the camera properties from the file\n");

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);

    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.19 Recording

Record video files.

Recording.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
```

```
bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;

    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) *
(g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    MV_CC_INPUT_FRAME_INFO stInputFrameInfo = {0};

    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 
1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);

            stInputFrameInfo.pData = pData;
            stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
            nRet = MV_CC_InputOneFrame(pUser, &stInputFrameInfo);
            if (MV_OK != nRet)
            {
                printf("Input one frame fail! nRet [0x%x]\n", nRet);
            }

        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }

    free(pData);

    return 0;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;

    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!", 
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
            break;
        }

        MVCC_INTVALUE stParam;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "PayloadSize", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        g_nPayloadSize = stParam.nCurValue;

        MV_CC_RECORD_PARAM stRecordPar;
        memset(&stParam, 0, sizeof(MVCC_INTVALUE));
        nRet = MV_CC_GetIntValue(handle, "Width", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get Width fail! nRet [0x%x]\n", nRet);
            break;
        }
        stRecordPar.nWidth = stParam.nCurValue;

        nRet = MV_CC_GetIntValue(handle, "Height", &stParam);
        if (MV_OK != nRet)
        {
            printf("Get Height fail! nRet [0x%x]\n", nRet);
            break;
        }
        stRecordPar.nHeight = stParam.nCurValue;

        MVCC_ENUMVALUE stEnumValue;
        nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &stEnumValue);
        if (MV_OK != nRet)
        {
            printf("Get Width fail! nRet [0x%x]\n", nRet);
            break;
        }
        stRecordPar.enPixelType = MvGvspPixelType(stEnumValue.nCurValue);

        MVCC_FLOATVALUE stFloatValue;
        nRet = MV_CC_GetFloatValue(handle, "ResultingFrameRate", &stFloatValue);
        if (MV_OK != nRet)
        {
            printf("Get Float value fail! nRet [0x%x]\n", nRet);
            break;
        }
        stRecordPar.fFrameRate = stFloatValue.fCurValue;

        stRecordPar.nBitRate = 1000;
        stRecordPar.enRecordFmtType = MV_FormatType_AVI;
        stRecordPar.strFilePath= "./Recording.avi";
        nRet = MV_CC_StartRecord(handle,&stRecordPar);
        if (MV_OK != nRet)
        {
            printf("Start Record fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nThreadID = 0;
        void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread ,
handle, 0 , &nThreadID );
        if (NULL == hThreadHandle)
        {
            break;
        }

        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

        g_bExit = true;
        Sleep(1000);

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_StopRecord(handle);
        if (MV_OK != nRet)
        {
            printf("Stop record fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```

B.20 Save Images of 3D Cameras in Point Cloud Format

The sample code shows how to save images of 3D cameras in point cloud format.

SavePointCloudData_3D.cpp

```
#include <stdio.h>
#include <Windows.h>
#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}

bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp &
0x000000ff);

        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo-
>SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
        printf("Serial Number: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber);
        printf("Device Number: %d\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.nDeviceNumber);
    }
    else
    {
        printf("Not support.\n");
    }

    return true;
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    unsigned char* pDstImageBuf = NULL;
    unsigned char* pSaveImageBuf = NULL;
    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }

        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }

        printf("Please Input camera index:");
        unsigned int nIndex = 0;
        scanf_s("%d", &nIndex);

        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Input error!\n");
            break;
        }

        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        MVCC_ENUMVALUE EnumValue = {0};
        nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &EnumValue);
        if (MV_OK != nRet)
        {
            printf("Get the Camera format fail! nRet [0x%x]\n", nRet);
            break;
        }

        enum MvGvspPixelType ePixelFormat =
(MvGvspPixelType)EnumValue.nCurValue;
        switch (ePixelFormat)
        {
        case PixelType_Gvsp_Coord3D_ABC32:
        case PixelType_Gvsp_Coord3D_ABC32f:
        case PixelType_Gvsp_Coord3D_AB32:
        case PixelType_Gvsp_Coord3D_AB32f:
        case PixelType_Gvsp_Coord3D_AC32:
        case PixelType_Gvsp_Coord3D_AC32f:
        case PixelType_Gvsp_Coord3D_ABC16:
            {
                nRet = MV_OK;
                break;
            }

        default:
            {
                nRet = MV_E_SUPPORT;
                break;
            }
        }
        if (MV_OK != nRet)
        {
            printf("This is not a supported 3D format!");
            break;
        }

        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
                if(nRet != MV_OK)
                {
                    printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!",
nPacketSize);
            }
        }

        MV_XML_AccessMode pAccessMode = AM_NI;
        nRet = MV_XML_GetNodeAccessMode(handle, "TriggerMode", &pAccessMode);
        if (MV_OK != nRet)
        {
            printf("Get Access mode of trigger mode fail! nRet [0x%x]\n", nRet);
        }
        else
        {
            nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
            if (MV_OK != nRet)
            {
                printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
                break;
            }
        }

        MVCC_INTVALUE_EX stIntValue = {0};
        nRet = MV_CC_GetIntValueEx(handle, "PayloadSize", &stIntValue);
        if (MV_OK != nRet)
        {
            printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
            break;
        }
        unsigned int nPayloadSize = (unsigned int)stIntValue.nCurValue;

        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        unsigned int nImageNum = 100;
        unsigned char* pSaveImageBuf = (unsigned char*)malloc(nPayloadSize *
nImageNum);
        if (NULL == pSaveImageBuf)
        {
            printf("Malloc  Save buffer fail!\n");
            break;
        }
        unsigned int nSaveImageSize = nPayloadSize * nImageNum;

        unsigned int nSaveDataLen = 0;

        MV_FRAME_OUT stOutFrame = {0};
        for(unsigned int i = 0;i < nImageNum; i++)
        {
            nRet = MV_CC_GetImageBuffer(handle, &stOutFrame, 1000);
            if (nRet == MV_OK)
            {
                printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n",
                    stOutFrame.stFrameInfo.nWidth,
stOutFrame.stFrameInfo.nHeight, stOutFrame.stFrameInfo.nFrameNum);

                if (nSaveImageSize > (nSaveDataLen +
stOutFrame.stFrameInfo.nFrameLen))
                {
                    memcpy(pSaveImageBuf + nSaveDataLen, stOutFrame.pBufAddr,
stOutFrame.stFrameInfo.nFrameLen);
                    nSaveDataLen += stOutFrame.stFrameInfo.nFrameLen;
                }

                nRet = MV_CC_FreeImageBuffer(handle, &stOutFrame);
                if(nRet != MV_OK)
                {
                    printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
                }
            }
            else
            {
                printf("No data[0x%x]\n", nRet);
            }
        }

        MV_SAVE_POINT_CLOUD_PARAM stSavePoCloudPar = {0};

        stSavePoCloudPar.nLinePntNum = stOutFrame.stFrameInfo.nWidth;
        stSavePoCloudPar.nLineNum = stOutFrame.stFrameInfo.nHeight * nImageNum;

        unsigned char* pDstImageBuf = (unsigned
char*)malloc(stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3
+ 4) + 2048);
        if (NULL == pDstImageBuf)
        {
            printf("Malloc Dst buffer fail!\n");
            break;
        }

        unsigned int nDstImageSize = stSavePoCloudPar.nLineNum *
stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048;

        stSavePoCloudPar.enPointCloudFileType = MV_PointCloudFile_PLY;
        stSavePoCloudPar.enSrcPixelType = stOutFrame.stFrameInfo.enPixelType;
        stSavePoCloudPar.pSrcData = pSaveImageBuf;
        stSavePoCloudPar.nSrcDataLen = nSaveDataLen;
        stSavePoCloudPar.pDstBuf = pDstImageBuf;
        stSavePoCloudPar.nDstBufSize = nDstImageSize;

        nRet = MV_CC_SavePointCloudData(handle, &stSavePoCloudPar);
        if(MV_OK != nRet)
        {
            printf("Save point cloud data failed!nRet [0x%x]\n", nRet);
            break;
        }

        char pImageName[32] = "PointCloudData.ply";
        FILE* fp = fopen(pImageName, "wb+");
        if(!fp)
        {
            printf("Allocate memory fail! nRet [0x%x]\n", nRet);
            break;
        }
        fwrite(pDstImageBuf, 1, stSavePoCloudPar.nDstBufLen, fp);
        fclose(fp);
        printf("Save point cloud data succeed!\n");

        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
            break;
        }
    } while (0);


    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }

    if(pSaveImageBuf)
    {
        free(pSaveImageBuf);
    }
    if(pDstImageBuf)
    {
        free(pDstImageBuf);
    }

    printf("Press a key to exit.\n");
    WaitForKeyPress();

    return 0;
}
```