Machine Vision Camera SDK (C)

Developer Guide

Legal Information

Contents

Chapter 1 Overview

Machine vision camera SDK (MvCameraSDK) contains API definitions, examples, camera driver and so on. It is compatible with standard protocols, and currently, GigEVision and USB3Vision 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 drive program can be selected to be installed during installing Machine Vision Software (MVS).

1.2 Integrated Development Environment

The IDE (integrated development environment) configuration of MvCameraSDK is shown below.

Operating System

Item	Required
Linux Operating System	Ubuntu with version 14.04 (32-bit or 64-bit) , Ubuntu with version 16.04 (32-bit or 64-bit), Ubuntu with version 18.04 (32-bit or 64-bit), Ubuntu with version 20.04 (64-bit), Redhat7 (64-bit), Centos7 (32-bit or 64-bit), gcc/g++ with version 4.6.3 or above. Supported ARM board type: NVIDIA TX2 (Ubuntu16.04), RaspberryPiB3.0+ (NOOBS_2.8.2), ODROID-XU4 (Ubuntu16.04/ Ubuntu18.04).

Item	Required
	Note RaspberryPiB3.0+ and ODROID-XU4 only support GigE camera.

MVS Installation

Note

The MVS installation packages are different of different hardware environment (x86_64, i386, armhf, aarch64, arm-none), you should select a suitable installation package according to the actual hardware environment. For details, you can contact the technical supporter.

1. Get the system root authority ("sudo su" or "su root") before Machine Vision Software (MVS) installation.
2. Two types of installation packages are provided: xxxx.deb and xxxx.tar.gz.
   - For xxxx.deb: Enter the folder of the installation package, execute "sudo dpkg -i xxxx.deb" to install MVS directly.
     For xxxx.tar.gz: Enter the folder of the installation package, execute "tar –xzvf xxxx.tar.gz" to unzip the package, open the folder, execute the script "source ./setup.sh" to install MVS.
3. MVS is installed under the directory ***opt/MVS***.
4. Run "/opt/MVS/bin/MVS.sh" (or run "./MVS.sh" under the directory ***/opt/MVS/bin***) to check if MVS is installed properly.

Development Folder Contents

By default, Machine Vision Software (MVS) is installed in the path of ***/opt/MVS***. After installation, the folder MVS contains the folder Development, of which the contents are as below:

Content Name	Description
driver	Camera drive files
license	License
lib	lib files
logserver	Log server
doc	Programming documents
bin	Executable file
include	Header files
Samples	Sample programs

Driver Installation

The GigE camera driver will be installed when the MVS is installed. You can also install or uninstall the driver according to the method below:

- Installation: Execute "./load.sh" in the directory ***/opt/MVS/driver***.
- Uninstallation: Execute "./unload.sh" in the directory ***/opt/MVS/driver***.

Disable Firewall

If cameras cannot be enumerated in Linux operating system, you should disable the firewall. Refer to the following methods to disable firewall in different Linux systems:

- Ubuntu
  The firewall on Ubuntu is disabled by default.
- CentOS 7
  Disable temporarily: service firewalld stop or systemctl stop firewalld
  Disable permanently: chkconfig firewalld off
- RedHat7
  Disable temporarily: service firewalld stop or systemctl stop firewalld
  Disable permanently: chkconfig firewalld off

NIC Configuration

It is recommended to enable jumbo frame in Linux. Refer to the following methods (only for reference):

- Enable temporarily: ifconfig eth0 mtu 9000
- Enable permanently: write "ifconfig eth0 mtu 9000" to the last line of /opt/run_driver.sh, and it takes effect after restarting.

Note

eth0 is a NIC name, and varies in different systems; 9000 indicates the maximum size of the received packet, and the packet with maximum size may not be received by older NIC, you should set the size according to actual situation.

Script

- set_env_path.sh: Set environment variable, and add dynamic link library to PATH environment variable.
- set_rp_filter.sh: Disable rpfilter to enumerate the camera when the camera and PC are in different IP segments.
- set_sdk_version.sh: Generate SDK soft links.
- set_socket_buffer_size.sh: For GigE packet loss, you can increase socket buffer to reduce the packet loss.

- set_usb_priority.sh: Set udev rule to use USB 3.0 device when the user has no administrator permission.
- set_usbfs_memory_size.sh: Set USB buffer to connect multiple USB 3.0 devices with ultra HD resolution.

Note

- For GigE cameras, it is recommended to enable jumbo frame, set MTU to 9000 and increase socket buffer (set_socket_buffer_size.sh).
- For USB 3.0 cameras, it is recommended to set the USB buffer (set_usbfs_memory_size.sh).

1.3 Update History

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

Summary of Changes in Version 3.2.0_Aug., 2021

Version	Content
Version 3.2.0_Aug., 2021	1. The new version uses static linking library of GenICam. And the GenICam dynamic library is not provided.
	2. The GigE driver supports the ARM structure and two types of ARM boards: NVIDIA TX2 and ODROID-XU4.
	3. Provided the SDK installation package (Runtime package), which includes the script of driver installation and the script of enabling read and write caching for USB3 vision cameras. The provided installation packages vary with the system (ARM or x86).
	4. Updated the driver name to gevfilter.
	5. Added one API for clearing the streaming data buffer: ***MV_CC_ClearImageBuffer*** .
	6. Added one API for rotating images: ***MV_CC_RotateImage*** .
	7. Added one API for flipping images: ***MV_CC_FlipImage*** .
	8. Added one API for setting gamma parameters of Bayer pattern: ***MV_CC_SetBayerGammaParam*** .
	9. Added one API for decoding the lossless compression stream into raw data: ***MV_CC_HB_Decode*** .
	10. Added one API for setting the packet size of USB3 vision device: ***MV_USB_SetTransferSize*** .

Version	Content
	11. Added one API for getting the packet size of USB3 vision device: ***MV_USB_GetTransferSize*** .
	12. Added one API for setting the number of transmission channels for USB3 vision device: ***MV_USB_SetTransferWays*** .
	13. Added one API for getting the number of transmission channels for USB3 vision device: ***MV_USB_GetTransferWays*** .
	14. Added one API for getting the GVCP command timeout: ***MV_GIGE_GetGvcpTimeout*** .
	15. Added one API for setting GVCP command retransmission times: ***MV_GIGE_SetRetryGvcpTimes*** .
	16. Added one API for getting GVCP command retransmission times: ***MV_GIGE_GetRetryGvcpTimes*** .

Summary of Changes in Version 3.1.0_Sep., 2019

Version	Content
Version 3.1.0_Sep., 2019	1. Added the drive support for connecting GigE camera.
	2. Added API for sending PTP (Precision Time Protocol) command of taking photo: ***MV_GIGE_IssueActionCommand*** .
	3. Added API for checking if device is connected: ***MV_CC_IsDeviceConnected*** .
	4. Added API for clearing GenICam node cache: ***MV_CC_InvalidateNodes*** .

Summary of Changes in Version 3.0.0_Nov., 2018

New document.

1.4 Notice

The sample codes provided in this manual is only for reference. Improper sample code may fail to implement the expected function or even cause damage to the program. To ensure the normal operation of program, you should adjust the sample code according to your actual requirement and make a through test before using.

Chapter 2 Programming Guideline

2.1 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

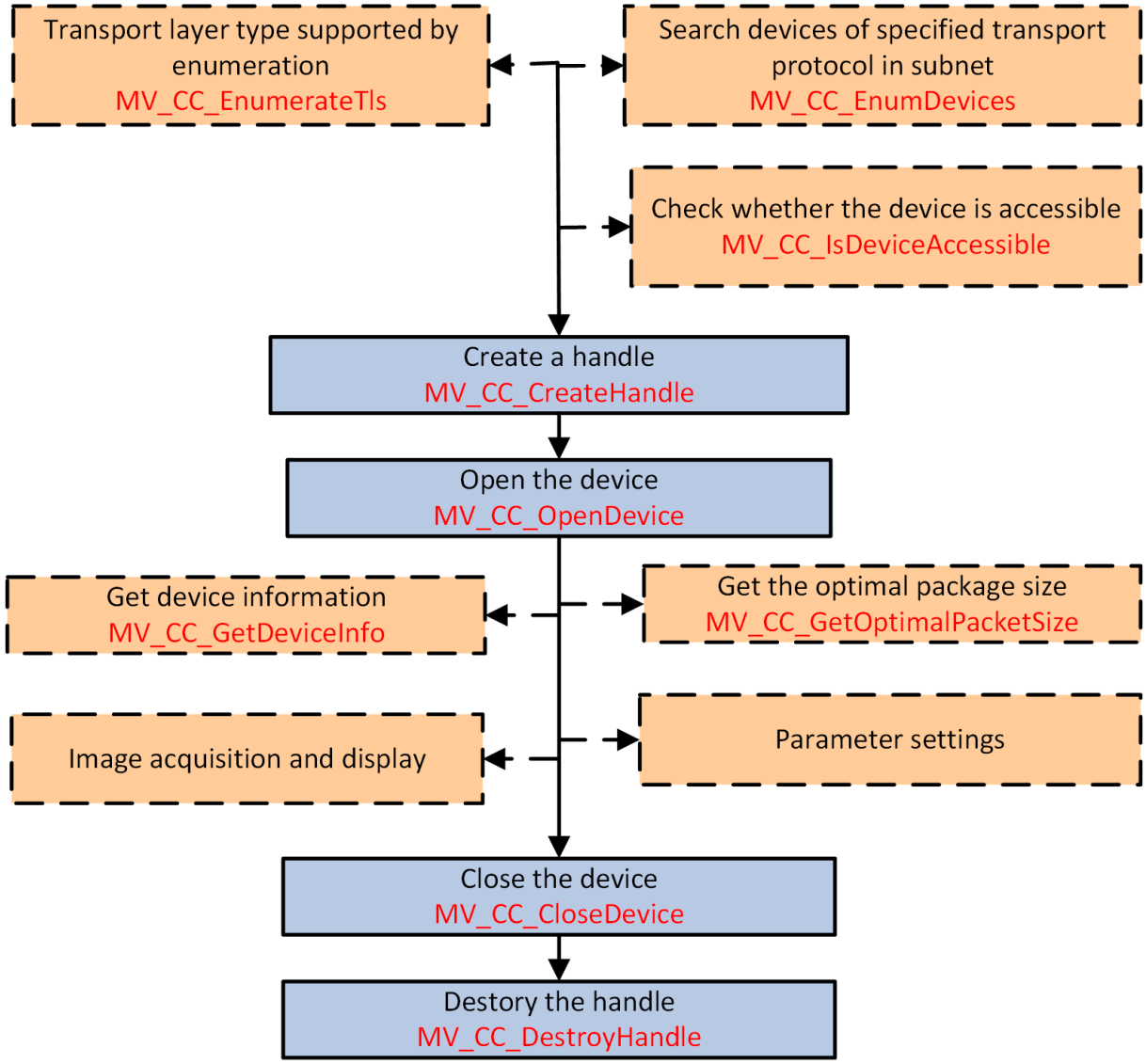


Figure 2-1 Programming Flow of Connecting Device

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.

2.2 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.

2.2.1 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

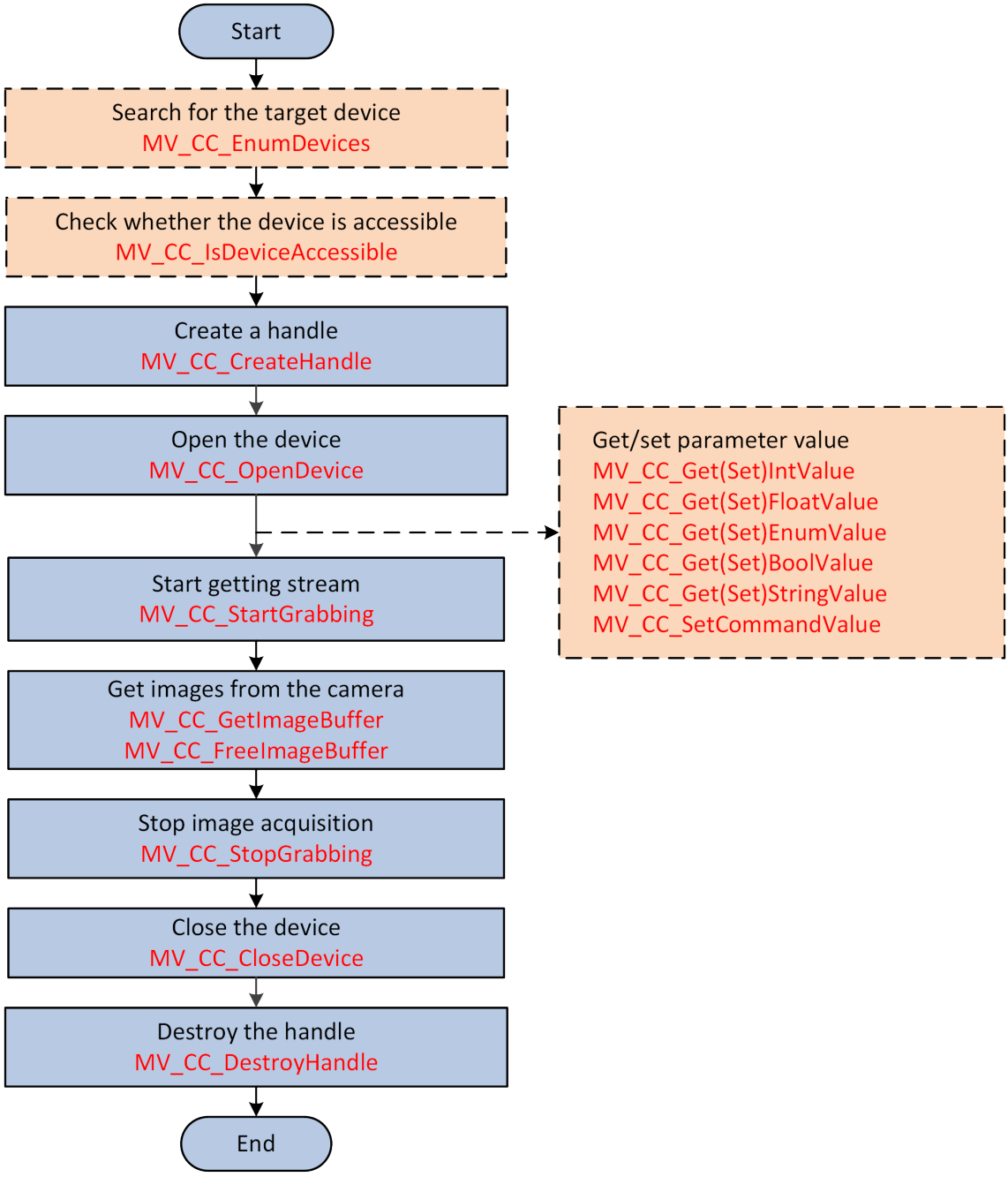


Figure 2-2 Programming Flow of Getting Image Directly

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.

2.2.2 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

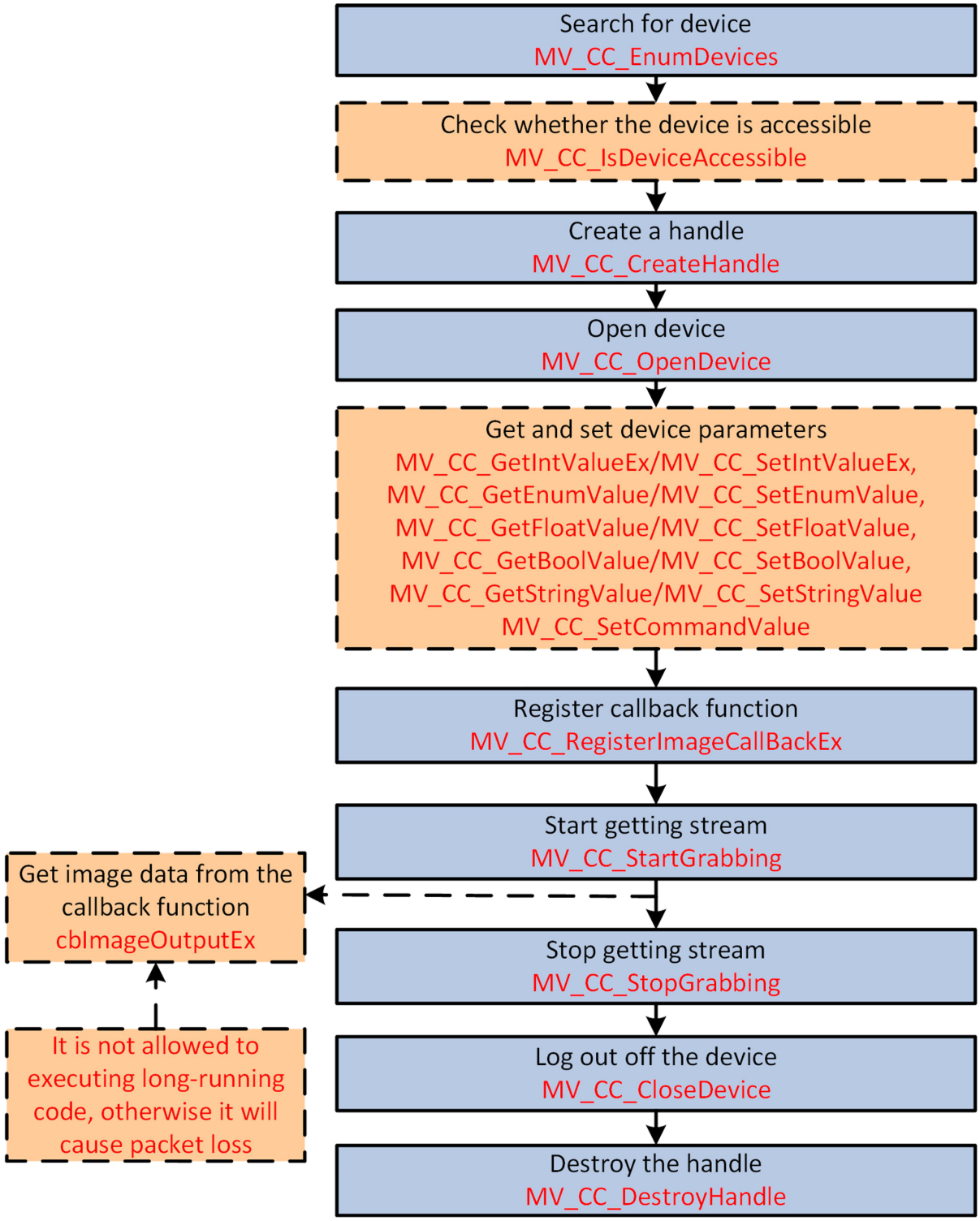


Figure 2-3 Programming Flow of Getting Image in Callback Function

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 3 API Reference

3.1 General

3.1.1 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.

3.1.2 MV_CC_EnumerateTls

Get supported transport layers.

API Definition

```
int MV_CC_EnumerateTls(
);
```

Return Value

Return supported device type, indicated by bit, supporting multiple selection, available protocol types are shown below:

Macro Definition	Value	Description
MyCamera.MV_UNKNOW_DEVICE	0x00000000	Unknown Device Type
MyCamera.MV_GIGE_DEVICE	0x00000001	GigE Device
MyCamera.MV_1394_DEVICE	0x00000002	1394-a/b Device

Macro Definition	Value	Description
MyCamera.MV_USB_DEVICE	0x00000004	USB3.0 Device
MyCamera.MV_CAMERALINK_DEVICE	0x00000008	CameraLink Device

E.g., if nTLayerType == MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE, it indicates that GigE device and USB3.0 device are both supported.

3.1.3 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

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.

See Also

MV_CC_EnumDevicesEx

3.1.4 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.

See Also

MV_CC_EnumDevices

3.1.5 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

3.1.6 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.

3.1.7 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

3.1.8 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

3.1.9 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

3.1.10 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

3.1.11 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

3.1.12 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

3.2 Parameter Settings

3.2.1 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

3.2.2 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.

3.2.3 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

3.2.4 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

3.2.5 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

3.2.6 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

3.2.7 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

3.2.8 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

3.2.9 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

3.2.10 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

3.2.11 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

3.2.12 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.

3.2.13 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

3.2.14 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

3.2.15 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

3.2.16 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

3.2.17 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**.

3.3 Functional

3.3.1 General APIs

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_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_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_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.

MV_CC_RegisterExceptionCallBack

Register exception message callback.

API Definition

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

Parameters

handle

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

fExceptionCallBack

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

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

nMsgType

[OUT] Exception message type

pUser

[OUT] 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_RegisterAllEventCallBack

Register the callback function for multiple events.

API Definition

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

Parameters

handle

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

fEventCallBack

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

```
void(__stdcall* cbEvent)(
  unsigned int    nExternalEventId,
  void            *pUser
);
```

nExternalEventId

[OUT] Output event ID, see the enumeration ***MV_EVENT_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

- 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,
  cbEvent       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

[OUT] Output event information, see enumeration ***MV_EVENT_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

- 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_FeatureSave

Save the camera feature files.

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] Input parameter

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 feature files to the local PC.

See Also

MV_CC_FeatureLoad

MV_CC_FeatureLoad

Import camera feature files.

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] Input parameter

Return Value

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

See Also

MV_CC_FeatureSave

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_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.

3.3.2 GigE APIs

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_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	Drive 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 "drive mode" by default, and supported only by GigEVision camera.

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_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_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 drive 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_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_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_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_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.

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.

3.4 Image Acquisition

3.4.1 MV_CC_RegisterImageCallBackEx

Register image data callback function, supporting getting chunk information.

API Definition

```
int MV_CC_RegisterImageCallBackEx(
  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_EX    *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_EX*** 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_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.

3.4.2 MV_CC_RegisterImageCallBackForRGB

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

API Definition

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

Parameters

handle

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

fOutputCallBack

[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

[OUT] Address of buffer that saves image data

pFrameInfo

[OUT] 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

[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_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

3.4.3 MV_CC_RegisterImageCallBackForBGR

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

API Definition

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

Parameters

handle

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

fOutputCallBack

[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

[OUT] Address of buffer that saves image data

pFrameInfo

[OUT] 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

[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_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

3.4.4 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

3.4.5 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

3.4.6 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.

3.4.7 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.

3.4.8 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.

3.4.9 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

3.4.10 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.

3.4.11 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.

3.5 Image Processing

3.5.1 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

3.5.2 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.

3.5.3 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.

3.5.4 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.

3.5.5 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.

3.5.6 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.

3.5.7 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.

3.5.8 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***).

3.5.9 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.

3.5.10 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

3.5.11 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

3.6 Camera Internal APIs

3.6.1 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

3.6.2 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

3.6.3 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**.

3.6.4 MV_XML_GetRootNode

Get the root node.

API Definition

```
int MV_XML_GetRootNode(
  void                    *handle,
  MV_XML_NODE_FEATURE     *pstNode
)
```

Parameters

handle

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

pstNode

[OUT] The root node information structure. See ***MV_XML_NODE_FEATURE*** for details.

Return Value

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

3.6.5 MV_XML_GetChildren

Get all child nodes of a specified father node.

API Definition

```
int MV_XML_GetChildren(
  void                     *handle,
  MV_XML_NODE_FEATURE      *pstNode,
  MV_XML_NODES_LIST        *pstNodesList
)
```

Parameters

handle

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

pstNode

[IN] The root node information structure. See ***MV_XML_NODE_FEATURE*** for details.

pstNodesList

[OUT] The node information list structure. See ***MV_XML_NODES_LIST*** for details.

Return Value

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

3.6.6 MV_XML_GetNodeFeature

Get the current node feature.

API Definition

```
int MV_XML_GetNodeFeature(
  void                      *handle,
  MV_XML_NODE_FEATURE       *pstNode,
  void                      *pstFeature
)
```

Parameters

handle

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

pstNode

[IN] The root node information structure. See ***MV_XML_NODE_FEATURE*** for details.

pstFeature

[OUT] The current node feature structure. See MV_XML_FEATURE_x for details.

Return Value

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

3.6.7 MV_XML_RegisterUpdateCallBack

Register the update callback.

API Definition

```
int MV_XML_RegisterUpdateCallBack(
  void              *handle,
  void__stdcall     *cbUpdate,
  void              *pUser
)
```

Parameters

handle

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

cbUpdate

[IN] Pointer to the callback function, see the details below:

```
void(__stdcall *cbUpdate)(
  enum MV_XML_InterfaceType     enType,
  void                          *pstFeature,
  MV_XML_NODES_LIST             *pstNodesList,
  void                          *pUser
```

enType

The interface type corresponding to each node, see details in ***MV_XML_InterfaceType*** .

pstFeature

Current node feature.

pstNodesList

The updated node list, see details in ***MV_XML_NODES_LIST*** .

pUser

The user data.

pUser

[IN] The user data.

Return Value

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

3.6.8 MV_XML_UpdateNodeFeature

Update the node.

API Definition

```
int MV_XML_UpdateNodeFeature(
  void                          *handle,
  enum MV_XML_InterfaceType     enType,
  void                          *pstFeature
)
```

Parameters

handle

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

enType

[IN] The node type. See ***MV_XML_InterfaceType*** for details.

pstFeature

[OUT] The current node feature structure. See MV_XML_FEATURE_x for details.

Return Value

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

3.7 U3V APIs

3.7.1 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.

3.7.2 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.

3.7.3 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.

3.7.4 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.

Chapter 4 Data Structure and Enumeration

4.1 Data Structure

4.1.1 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.

4.1.2 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.

4.1.3 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

4.1.4 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.

4.1.5 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.

4.1.6 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

4.1.7 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.

4.1.8 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

4.1.9 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

4.1.10 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

4.1.11 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

4.1.12 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.

4.1.13 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.

4.1.14 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.

4.1.15 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.

4.1.16 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.

4.1.17 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.

4.1.18 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

4.1.19 MV_CC_ROTATE_IMAGE_PARAM

Structure about Image Rotation

Member	Data Type	Description
public **enPixelType**	enum ***MvGvspPixelType***	Pixel format
nWidth	unsigned int	Image width

Member	Data Type	Description
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.

4.1.20 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

4.1.21 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

4.1.22 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.

4.1.23 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.

4.1.24 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.

4.1.25 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

4.1.26 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.

4.1.27 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

4.1.28 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.

4.1.29 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

4.1.30 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.

4.1.31 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.

4.1.32 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

4.1.33 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.

4.1.34 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.

4.2 Enumeration

4.2.1 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.

Enumeration Type	Macro Definition Value	Description
		Note This parameter is valid for color interpolation only, it is invalid for color correction.

4.2.2 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

4.2.3 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

4.2.4 MV_IMG_FLIP_TYPE

Enumeration about Flip Types

Member	Marco Definition Value	Description
MV_FLIP_VERTICAL	1	Vertical
MV_FLIP_HORIZONTAL	2	Horizontal

4.2.5 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°

4.2.6 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

4.2.7 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

4.2.8 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

4.2.9 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 5 FAQ (Frequently Asked Questions)

Here are some frequently asked questions in programming process. We provide the corresponding answers to help the users to solve the problems.

How to shoot the troubles?

- For the program exception during SDK development, run the MVS client first to check the corresponding functions.
- For MVS normally running but program exception during SDK development, mainly shoot the program trouble of secondary development.
- For MVS client exception, refer to the following FAQ for solving the problems.
- If the problem still cannot be solved by the above methods, provide the exception description and pictures, MVS client version No. (see it in the Help of MVS), MvCameraControl.dll, MVGigEVisionSDK.dll, and MvUsb3vTL.dll to our technical supports for help.

5.1 GigE Vision Camera

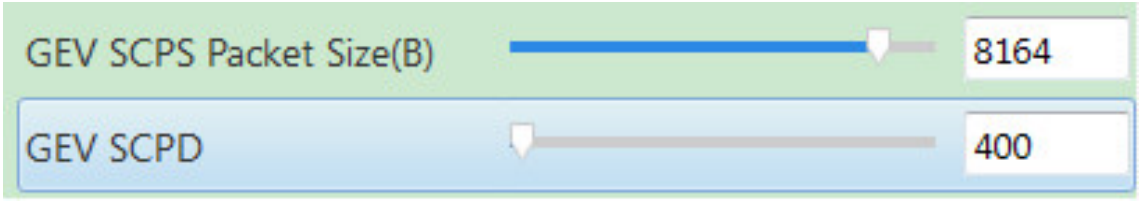
5.1.1 Why is there packet loss?

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.



5.1.2 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.

5.1.3 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.

5.1.4 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.

5.1.5 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.

5.1.6 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.

5.1.7 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:

5.1.8 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.

5.1.9 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.

5.1.10 How to fix error "ld:-lMvCameraControl cannot find libMvCameraControl library" when compiling?

Cause

The environment variables of current console do not take effect after installation.

Solution

1. Check if all SDK environment variables take effect by executing:
   echo $MVCAM_COMMON_RUNENV
   If the path to relative SDK is not printed, you should execute:
   cd installation directory
   source set_env_path.sh
2. If there is no problem with the environment variables, check if the library exists by executing:
   cd /opt/MVS/lib/64
   ls -lh

5.1.11 How to fix error "ld:-lMvCameraControl not compatable symbol" when compiling?

Cause

SDK version and hardware environment mismatched, or compiler toolchain does not support.

Solution

1. Check if the SDK version and hardware environment are matched:
   uname -a
   cd /opt/MVS/lib/64
   readelf -h libMvCameraControl.so
2. Check if the gcc version and corresponding SDK version are matched, the gcc version of different SDK is shown below:
   x86_64: gcc-4.4.7
   i386: gcc-4.4.7
   armhf: gcc-4.8.2
   aarch64: gcc-4.9.4
   If used gcc version is lower than the SDK required version, you can use gcc with higher version. For details, please contact our technical supporter.

5.1.12 Why can't I enumerate the GigE cameras?

Cause

The computer IP address is not static, and the LAN IP address cannot be assigned on Linux system, so the GigE camera cannot be enumerated.

Solution

Set the computer IP address to static.

5.2 USB3 Vision Camera

5.2.1 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 Get The Chunk Information

The sample code below shows how to enable the ChunkData function, configure ChunkData parameters and get the ChunkData information.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void* WorkThread(void* pUser)
{
    int nRet = MV_OK;
    // Get the payload size
    MVCC_INTVALUE stParam;
    memset(&stParam, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(pUser, "PayloadSize", &stParam);
    if (MV_OK != nRet)
    {
        printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
        return NULL;
    }
    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) *
stParam.nCurValue);
    if (NULL == pData)
    {
        return NULL;
    }
    unsigned int nDataSize = stParam.nCurValue;
    while(1)
    {
        if(g_bExit)
        {
            break;
        }

        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            //Print the parsed timestamp information of one frame
            printf("Get One Frame: Chunk_ExposureTime[%f],
Chunk_SecondCount[%d], Chunk_CycleCount[%d], Chunk_CycleOffset[%d],
nFrameNum[%d]\n",
                stImageInfo.fExposureTime, stImageInfo.nSecondCount,
stImageInfo.nCycleCount, stImageInfo.nCycleOffset, stImageInfo.nFrameNum);
        }
```

```
        else{
            printf("No data[%x]\n", nRet);
        }
    }
    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));
        // Enumerate devices
        // enum device
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("MV_CC_EnumDevices fail! nRet [%x]\n", nRet);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
```

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

        // Open Chunk mode
        nRet = MV_CC_SetBoolValue(handle, "ChunkModeActive", true);
        if (MV_OK != nRet)
        {
            printf("Set Chunk Mode fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Set the  Chunk Selector to Exposure
        nRet = MV_CC_SetEnumValueByString(handle, "ChunkSelector", "Exposure");
        if (MV_OK != nRet)
        {
            printf("Set Exposure Chunk fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Enable Chunk
        nRet = MV_CC_SetBoolValue(handle, "ChunkEnable", true);
        if (MV_OK != nRet)
        {
            printf("Set Chunk Enable fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Set the Chunk Selector to Timestamp
        nRet = MV_CC_SetEnumValueByString(handle, "ChunkSelector", "Timestamp");
        if (MV_OK != nRet)
        {
            printf("Set Timestamp Chunk fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Enable Chunk
        nRet = MV_CC_SetBoolValue(handle, "ChunkEnable", true);
        if (MV_OK != nRet)
        {
            printf("Set Chunk Enable fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
nPacketSize);
            }
        }

        // Set the trigger mode to off
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
            break;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        // Shut down device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
```

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

B.2 Connect to Cameras via IP Address

Connect to cameras via its IP address and the related NIC's IP address.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
static void* 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 (NULL == pData)
```

```
    {
        return NULL;
    }
    unsigned int nDataSize = g_nPayloadSize;
    while(1)
    {
        if(g_bExit)
        {
            break;
        }
        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);
            break;
        }
    }
    free(pData);
    return 0;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    MV_CC_DEVICE_INFO stDevInfo = {0};
    MV_GIGE_DEVICE_INFO stGigEDev = {0};
    // IP address of the camera to be connected
    printf("Please input Current Camera Ip : ");
    char nCurrentIp[128];
    scanf("%s", &nCurrentIp);
    // The NIC IP address corresponding to the camera to be connected
    printf("Please input Net Export Ip : ");
    char nNetExport[128];
    scanf("%s", &nNetExport);
    unsigned int nIp1, nIp2, nIp3, nIp4, nIp;
    sscanf(nCurrentIp, "%d.%d.%d.%d", &nIp1, &nIp2, &nIp3, &nIp4);
    nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;
    stGigEDev.nCurrentIp = nIp;
    sscanf(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;// It is valid for GigE cameras only
    stDevInfo.SpecialInfo.stGigEInfo = stGigEDev;
    do
    {
        // Create a handle for the selected device
```

```
        nRet = MV_CC_CreateHandle(&handle, &stDevInfo);
        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;
        }
        // Detect the optimal packet size (it is valid for GigE cameras only)
        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);
        }

        // Set the 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;
        }
        // Get the payload size
        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;
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
```

```
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        printf("Press a key to stop grabbing.\n");
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Shut down 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("exit.\n");
    return 0;
}
```

B.3 Get Camera Events

The sample code below show how to configure camera events, register the event callback function and handle events in callback function.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
void __stdcall EventCallBack(MV_EVENT_OUT_INFO * pEventInfo, void* pUser)
```

```
{
    if (pEventInfo)
    {
        int64_t nBlockId = pEventInfo->nBlockIdHigh;
        nBlockId = (nBlockId << 32) + pEventInfo->nBlockIdLow;
        int64_t nTimestamp = pEventInfo->nTimestampHigh;
        nTimestamp = (nTimestamp << 32) + pEventInfo->nTimestampLow;
        printf("EventName[%s], EventID[%d], BlockId[%lld], Timestamp[%lld]\n",
            pEventInfo->EventName, pEventInfo->nEventID, nBlockId, nTimestamp);
    }
}
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 Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
```

```
        // Create a handle for the selected device
        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 packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
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;
        }
        // Enable Event
        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;
        }
        // Register the event callback
        nRet = MV_CC_RegisterEventCallBackEx(handle, "ExposureEnd",
EventCallBack, handle);
        if (MV_OK != nRet)
        {
            printf("Register Event CallBack 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;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Shut down 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 (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit.\n");
    return 0;
}
```

B.4 Set Static IP Address of The Camera

The sample code below shows how to set the camera's static IP address by entering the IP address, subnet mask (NetMask), and default gateway (DefaultWay).

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
bool ConvertToHexIp(unsigned int *nHexIP, unsigned int *nDecIP, char c)
{
    if ( nDecIP[0] < 0 || nDecIP[0] > 255
        || nDecIP[1] < 0 || nDecIP[1] > 255
        || nDecIP[2] < 0 || nDecIP[2] > 255
        || nDecIP[3] < 0 || nDecIP[3] > 255
        || c != '\n')
    {
        return false;
    }
    *nHexIP = (nDecIP[0] << 24) + (nDecIP[1] << 16) + (nDecIP[2] << 8) +
nDecIP[3];
    return true;
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
```

```
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        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;
    unsigned int nIP[4] = {0};
    char c = '\0';
    unsigned int nIpAddr = 0, nNetWorkMask = 0, nDefaultGateway = 0;
    do
    {
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        // Enumerate devices
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("MV_CC_EnumDevices fail! nRet [%x]\n", nRet);
            break;
        }
        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");
```

```
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Enter the IP address, subnet mask, and default gateway
        printf("Please input ip, example: 192.168.1.100\n");
        int ch;
        if ( 5 != scanf("%d.%d.%d.%d%c", &nIP[0], &nIP[1], &nIP[2], &nIP[3],
&c) )
        {
            printf("input count error\n");
            MV_CC_DestroyHandle(handle);
            break;
        }
        if (!ConvertToHexIp(&nIpAddr, nIP, c))
        {
            printf("input IpAddr format is not correct\n");
            MV_CC_DestroyHandle(handle);
            break;
        }
        printf("Please input NetMask, example: 255.255.255.0\n");
        if ( 5 != scanf("%d.%d.%d.%d%c", &nIP[0], &nIP[1], &nIP[2], &nIP[3],
&c) )
        {
            printf("input count error\n");
            MV_CC_DestroyHandle(handle);
            break;
        }
        if (!ConvertToHexIp(&nNetWorkMask, nIP, c))
        {
            printf("input NetMask format is not correct\n");
            MV_CC_DestroyHandle(handle);
            break;
        }
        printf("Please input DefaultWay, example: 192.168.1.1\n");
        if ( 5 != scanf("%d.%d.%d.%d%c", &nIP[0], &nIP[1], &nIP[2], &nIP[3],
&c) )
        {
            printf("input count error\n");
```

```
            MV_CC_DestroyHandle(handle);
            break;
        }
        if (!ConvertToHexIp(&nDefaultGateway, nIP, c))
        {
            printf("input DefaultWay format is not correct\n");
            MV_CC_DestroyHandle(handle);
            break;
        }
        // Set the ForceIP
        nRet = MV_GIGE_ForceIpEx(handle, nIpAddr, nNetWorkMask,
nDefaultGateway);
        if (MV_OK != nRet)
        {
            printf("MV_GIGE_ForceIpEx fail! nRet [%x]\n", nRet);
            break;
        }
        printf("set IP succeed\n");
        PressEnterToExit();
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.5 Get Images in Callback Function

The sample code below shows how to get images by registering the image callback function.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
```

```
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
void __stdcall ImageCallBackEx(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        printf("GetOneFrame, 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));
        // 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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
```

```
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }

        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
nPacketSize);
            }
        }

        // Set the trigger mode to off
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
            break;
        }
        // Register the callback for grabbing images
        nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBackEx, handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_RegisterImageCallBackEx fail! nRet [%x]\n", nRet);
            break;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
```

```
            break;
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.6 Get Images Directly

The sample code below shows how to get images directly.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bExit = false;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
```

```
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void* WorkThread(void* pUser)
{
    int nRet = MV_OK;
    // Get the payload size
    MVCC_INTVALUE stParam;
    memset(&stParam, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(pUser, "PayloadSize", &stParam);
    if (MV_OK != nRet)
    {
        printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
        return NULL;
    }
    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) *
```

```
stParam.nCurValue);
    if (NULL == pData)
    {
        return NULL;
    }
    unsigned int nDataSize = stParam.nCurValue;
    while(1)
    {
        if(g_bExit)
        {
            break;
        }
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("GetOneFrame, Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else{
            printf("No data[%x]\n", nRet);
        }
    }
    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));
        // 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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }

        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
nPacketSize);
            }
        }

        // Set the trigger mode to off
```

```
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
            break;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
```

```
    return 0;
}
```

B.7 Get Images Directly with High Performance

The sample code shows how to get images directly with high performance.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
// Wait for the key press
void WaitForKeyPress(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        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  void* 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
    {
        // 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 Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        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 packet size (it is valid for GigE cameras 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 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 payload size
        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;
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
```

```
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Shut down 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.8 Grab Images of Multiple Cameras

The sample code below shows how to grab images of multiple cameras.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
#define CAMERA_NUM             2
bool g_bExit = false;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
```

```
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void* WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MVCC_STRINGVALUE stStringValue = {0};
    char camSerialNumber[256] = {0};
    nRet = MV_CC_GetStringValue(pUser, "DeviceSerialNumber", &stStringValue);
    if (MV_OK == nRet)
    {
        memcpy(camSerialNumber, stStringValue.chCurValue,
sizeof(stStringValue.chCurValue));
    }
```

```
    else
    {
        printf("Get DeviceUserID Failed! nRet = [%x]\n", nRet);
    }
    // Get the payload size
    MVCC_INTVALUE stParam;
    memset(&stParam, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(pUser, "PayloadSize", &stParam);
    if (MV_OK != nRet)
    {
        printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
        return NULL;
    }
    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) *
stParam.nCurValue);
    if (NULL == pData)
    {
        return NULL;
    }
    unsigned int nDataSize = stParam.nCurValue;
    while(1)
    {
        if(g_bExit)
        {
            break;
        }

        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("Cam Serial Number[%s]:GetOneFrame, Width[%d], Height[%d],
nFrameNum[%d]\n",
                camSerialNumber, stImageInfo.nWidth, stImageInfo.nHeight,
stImageInfo.nFrameNum);
        }
        else
        {
            printf("cam[%s]:Get One Frame failed![%x]\n", camSerialNumber,
nRet);
        }
    }
    return 0;
}
int main()
{
    int nRet = MV_OK;
    void* handle[CAMERA_NUM] = {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;
    }
    if(stDeviceList.nDeviceNum < CAMERA_NUM)
    {
        printf("only have %d camera\n", stDeviceList.nDeviceNum);
        return -1;
    }

    // Tips for testing multiple cameras
    printf("Start %d camera Grabbing Image test\n", CAMERA_NUM);
    for(int i = 0; i < CAMERA_NUM; i++)
    {
        printf("Please Input Camera Index: ");
        scanf("%d", &nIndex);
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle[i], stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            MV_CC_DestroyHandle(handle[i]);
            return -1;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle[i]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            MV_CC_DestroyHandle(handle[i]);
```

```
            return -1;
        }

        // Detect the optimal packet size (it is valid for GigE cameras only)
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(handle[i]);
            if (nPacketSize > 0)
            {
                nRet =
MV_CC_SetIntValue(handle[i],"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);
            }
        }
    }
    for(int i = 0; i < CAMERA_NUM; i++)
    {
        // Set the trigger mode to off
        nRet = MV_CC_SetEnumValue(handle[i], "TriggerMode",
MV_TRIGGER_MODE_OFF);
        if (MV_OK != nRet)
        {
            printf("Cam[%d]: MV_CC_SetTriggerMode fail! nRet [%x]\n", i, nRet);
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle[i]);
        if (MV_OK != nRet)
        {
            printf("Cam[%d]: MV_CC_StartGrabbing fail! nRet [%x]\n",i, nRet);
            return -1;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle[i]);
        if (nRet != 0)
        {
            printf("Cam[%d]: thread create failed.ret = %d\n",i, nRet);
            return -1;
        }
    }
    PressEnterToExit();
    for(int i = 0; i < CAMERA_NUM; i++)
    {
        // Stop grabbing images
```

```
        nRet = MV_CC_StopGrabbing(handle[i]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            return -1;
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle[i]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            return -1;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle[i]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            return -1;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.9 Process The Image

The sample code below shows how to convert the image pixel format and save the image.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        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;
    unsigned char * pData = NULL;
    unsigned char *pDataForRGB = NULL;
    unsigned char *pDataForSaveImage = NULL;
    do
    {
        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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }

        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
```

```
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
nPacketSize);
            }
        }

        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 0);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
            break;
        }
        // Get the payload size
        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;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%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) *
stParam.nCurValue);
        if (NULL == pData)
        {
            break;
        }
        unsigned int nDataSize = stParam.nCurValue;
        nRet = MV_CC_GetOneFrameTimeout(handle, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("Now you GetOneFrame, Width[%d], Height[%d], nFrameNum[%d]\n
\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
            // Process images
            printf("input 0 to do nothing, 1 to convert RGB, 2 to save as BMP
\n");
            int nInput = 0;
            scanf("%d", &nInput);
            switch (nInput)
            {
                // Continue without operation
```

```
            case 0:
                {
                    break;
                }
                // Convert the image format to RGB. The user can convert the
image format to other type according to the requirement
            case 1:
                {
                    pDataForRGB = (unsigned char*)malloc(stImageInfo.nWidth *
stImageInfo.nHeight * 4 + 2048);
                    if (NULL == pDataForRGB)
                    {
                        break;
                    }
                    // Convert pixel format
                    MV_CC_PIXEL_CONVERT_PARAM stConvertParam = {0};
                    // Image width, image height, input data buffer, input data
size, source pixel format
                    // target pixel format, output data buffer, provided output
buffer size
                    stConvertParam.nWidth = stImageInfo.nWidth;
                    stConvertParam.nHeight = stImageInfo.nHeight;
                    stConvertParam.pSrcData = pData;
                    stConvertParam.nSrcDataLen = stImageInfo.nFrameLen;
                    stConvertParam.enSrcPixelType = stImageInfo.enPixelType;
                    stConvertParam.enDstPixelType = PixelType_Gvsp_RGB8_Packed;
                    stConvertParam.pDstBuffer = pDataForRGB;
                    stConvertParam.nDstBufferSize = stImageInfo.nWidth *
stImageInfo.nHeight *  4 + 2048;
                    nRet = MV_CC_ConvertPixelType(handle, &stConvertParam);
                    if (MV_OK != nRet)
                    {
                        printf("MV_CC_ConvertPixelType fail! nRet [%x]\n",
nRet);
                        break;
                    }
                    FILE* fp = fopen("AfterConvert_RGB.raw", "wb");
                    if (NULL == fp)
                    {
                        printf("fopen failed\n");
                        break;
                    }
                    fwrite(pDataForRGB, 1, stConvertParam.nDstLen, fp);
                    fclose(fp);
                    printf("convert succeed\n");
                    break;
                }
            case 2:
                {
                    pDataForSaveImage = (unsigned
char*)malloc(stImageInfo.nWidth * stImageInfo.nHeight * 4 + 2048);
                    if (NULL == pDataForSaveImage)
```

```
                    {
                        break;
                    }
                    // Set image saving parameters
                    MV_SAVE_IMAGE_PARAM_EX stSaveParam;
                    memset(&stSaveParam, 0, sizeof(MV_SAVE_IMAGE_PARAM_EX));
                    // Output image format, input data pixel format, provided
output buffer size, image width
                    // image height, input data buffer, output image buffer,
JPG encoding quality
                    stSaveParam.enImageType = MV_Image_Bmp;
                    stSaveParam.enPixelType = stImageInfo.enPixelType;
                    stSaveParam.nBufferSize = stImageInfo.nWidth *
stImageInfo.nHeight * 4 + 2048;
                    stSaveParam.nWidth      = stImageInfo.nWidth;
                    stSaveParam.nHeight     = stImageInfo.nHeight;
                    stSaveParam.pData       = pData;
                    stSaveParam.nDataLen    = stImageInfo.nFrameLen;
                    stSaveParam.pImageBuffer = pDataForSaveImage;
                    stSaveParam.nJpgQuality = 80;
                    nRet = MV_CC_SaveImageEx2(handle, &stSaveParam);
                    if(MV_OK != nRet)
                    {
                        printf("failed in MV_CC_SaveImage,nRet[%x]\n", nRet);
                        break;
                    }
                    FILE* fp = fopen("image.bmp", "wb");
                    if (NULL == fp)
                    {
                        printf("fopen failed\n");
                        break;
                    }
                    fwrite(pDataForSaveImage, 1, stSaveParam.nImageLen, fp);
                    fclose(fp);
                    printf("save image succeed\n");
                    break;
                }
            default:
                break;
            }
        }
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
```

```
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    if (pData)
    {
        free(pData);
        pData = NULL;
    }
    if (pDataForRGB)
    {
        free(pDataForRGB);
        pDataForRGB = NULL;
    }
    if (pDataForSaveImage)
    {
        free(pDataForSaveImage);
        pDataForSaveImage = NULL;
    }
    PressEnterToExit();
    printf("exit\n");
    return 0;
}
```

B.10 Set The Multicast Mode

Set the transport mode to multicast mode.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
```

```
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void*  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
    {
        // 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 Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("Create Handle fail! nRet [0x%x]\n", nRet);
            break;
        }

        int c;
        while ( (c = getchar()) != '\n' && c != EOF );

        char key;

        // Ask the user if to launch the multicast controlling application or
multicast monitoring application
        printf("Start multicast sample in (c)ontrol or in (m)onitor mode? (c/m)
\n");
        scanf("%c", &key);

        if((key != 'c') && (key != 'm') && (key != 'C') && (key != 'M'))
        {
            printf("Input error\n");
            break;
        }

        // Query for the used mode
        bool monitorMode = (key == 'm') || (key == 'M');
        // Open the device
        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;
        }
```

```
        // Detect the optimal packet size (it is valid for GigE cameras only)
        if (MV_GIGE_DEVICE == stDeviceList.pDeviceInfo[nIndex]->nTLayerType &&
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);
            }
        }

        // Get the payload size
        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;
        // Specify the multicast IP address
        char strIp[] = "239.192.1.1";
        unsigned int nIp1, nIp2, nIp3, nIp4, nIp;
        sscanf(strIp, "%d.%d.%d.%d", &nIp1, &nIp2, &nIp3, &nIp4);
        nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;
        // Specify the multicast port
        MV_TRANSMISSION_TYPE stTransmissionType;
        memset(&stTransmissionType, 0, sizeof(MV_TRANSMISSION_TYPE));
        stTransmissionType.enTransmissionType = MV_GIGE_TRANSTYPE_MULTICAST;
        stTransmissionType.nDestIp = nIp;
        stTransmissionType.nDestPort = 1042;
        nRet = MV_GIGE_SetTransmissionType(handle, &stTransmissionType);
        if (MV_OK != nRet)
        {
            printf("Set Transmission Type 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;
        }
        pthread_t nThreadID = 0;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            break;
        }
        printf("Press a key to stop grabbing.\n");
        PressEnterToExit();
        g_bExit = true;
        sleep(1);
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        // Shut down 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("exit.\n");
    return 0;
}
```

B.11 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.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
unsigned int g_nMode = 0;
int g_nRet = MV_OK;
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void*  ProgressThread(void* pUser)
```

```
{
    int nRet = MV_OK;
    MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = {0};
    while(1)
    {
        // Get the file access progress
        nRet = MV_CC_GetFileAccessProgress(pUser, &stFileAccessProgress);
        printf("State = 0x%x,Completed = %ld,Total = %ld\r
\n",nRet,stFileAccessProgress.nCompleted,stFileAccessProgress.nTotal);
        if (nRet != MV_OK || (stFileAccessProgress.nCompleted != 0 &&
stFileAccessProgress.nCompleted == stFileAccessProgress.nTotal))
        {
            break;
        }
        usleep(50000);
    }
    return 0;
}
static void*  FileAccessThread(void* pUser)
{
    MV_CC_FILE_ACCESS stFileAccess = {0};
    stFileAccess.pUserFileName = "UserSet1.bin";
    stFileAccess.pDevFileName = "UserSet1";
    if (1 == g_nMode)
    {
        // In read mode
        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)
    {
        // In write mode
        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
    {
        // 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 Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        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;
        }
        //In read mode
        g_nMode = 1;
        printf("Read to file.\n");
        pthread_t nReadHandle;
        nRet = pthread_create(&nReadHandle, NULL ,FileAccessThread , handle);
        if (nRet != 0)
```

```
        {
            break;
        }
        usleep(5000);
        pthread_t nReadProcessHandle;
        nRet = pthread_create(&nReadProcessHandle, NULL ,ProgressThread ,
handle);
        if (nRet != 0)
        {
            break;
        }
        void *statusRead;
        void *statusReadProcess;
        pthread_join(nReadHandle, &statusRead);
        pthread_join(nReadProcessHandle, &statusReadProcess);
        if (MV_OK == g_nRet)
        {
            printf("File Access Read Success!\n");
        }
        printf("\n");
        // In write mode
        g_nMode = 2;
        printf("Write from file.\n");
        pthread_t nWriteHandle;
        nRet = pthread_create(&nWriteHandle, NULL ,FileAccessThread , handle);
        if (nRet != 0)
        {
            break;
        }
        usleep(5000);
        pthread_t nWriteProgressHandle;
        nRet = pthread_create(&nWriteProgressHandle, NULL ,ProgressThread ,
handle);
        if (nRet != 0)
        {
            break;
        }
        void *statusWrite;
        void *statusWriteProcess;
        pthread_join(nWriteHandle, &statusWrite);
        pthread_join(nWriteProgressHandle, &statusWriteProcess);
        if (MV_OK == g_nRet)
        {
            printf("File Access Write Success!\n");
        }
        // Shut down 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("exit.\n");
    return 0;
}
```

B.12 Import/Export The Camera Feature File

Export the feature configurations of the selected camera as a MFS file to the local PC, and import the MFS file from the local PC to the selected cameras to fast configure all its features without the inconvenience of configuring its features one by one.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
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 the IP address and user defined name of the current camera
```

```
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        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 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 Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        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;
        }
        printf("Start export the camera properties to the file\n");
        printf("Wait......\n");
        // Export the camera properties to the file
        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");
        // Import the camera properties from the file
        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");
        // Shut down 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("exit.\n");
    return 0;
}
```

B.13 Camera Reconnection

The sample code below shows how to reconnect to the camera when it turns offline.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
void* g_hHandle = NULL;
bool  g_bConnect = false;
char  g_strSerialNumber[64] = {0};
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
void __stdcall cbException(unsigned int nMsgType, void* pUser)
{
    printf("Device disconnect!\n");
    g_bConnect = false;
}
static void* WorkThread(void* pUser)
{
    int nRet = MV_OK;
    // Get the payload size
    MVCC_INTVALUE stParam;
    memset(&stParam, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(g_hHandle, "PayloadSize", &stParam);
    if (MV_OK != nRet)
    {
        printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
        return NULL;
    }
    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) *
stParam.nCurValue);
    if (NULL == pData)
    {
        return NULL;
    }
    unsigned int nDataSize = stParam.nCurValue;
    while(1)
```

```
    {
        if(!g_bConnect)
        {
            break;
        }
        nRet = MV_CC_GetOneFrameTimeout(g_hHandle, pData, nDataSize,
&stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("GetOneFrame, Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else
        {
            printf("no data[%x]\n", nRet);
        }
    }
    free(pData);
    return 0;
}
static void* ReconnectProcess(void* pUser)
{
    int nRet = MV_OK;
    MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
    while(1)
    {
        if (true == g_bConnect)
        {
            sleep(1);
            continue;
        }
        nRet = MV_CC_StopGrabbing(g_hHandle);
        nRet = MV_CC_CloseDevice(g_hHandle);
        nRet = MV_CC_DestroyHandle(g_hHandle);
        g_hHandle = NULL;
        printf("connecting...\n");
        // 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);
            continue;
        }
        // Select the device according to the serial number
        unsigned int nIndex = -1;
        if (stDeviceList.nDeviceNum > 0)
        {
            for (int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
```

```
                    continue;
                }
                if (pDeviceInfo->nTLayerType == MV_GIGE_DEVICE)
                {
                    if (!strcmp((char*)(pDeviceInfo-
>SpecialInfo.stGigEInfo.chSerialNumber), g_strSerialNumber))
                    {
                        nIndex = i;
                        break;
                    }
                }
                else if (pDeviceInfo->nTLayerType == MV_USB_DEVICE)
                {
                    if (!strcmp((char*)(pDeviceInfo-
>SpecialInfo.stUsb3VInfo.chSerialNumber), g_strSerialNumber))
                    {
                        nIndex = i;
                        break;
                    }
                }
            }
        }
        else
        {
            continue;
        }
        if (-1 == nIndex)
        {
            continue;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&g_hHandle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            continue;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(g_hHandle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            continue;
        }
        g_bConnect = true;

        // Detect the optimal packet size (it is valid for GigE cameras only)
        if (stDeviceList.pDeviceInfo[nIndex]->nTLayerType == MV_GIGE_DEVICE)
        {
            int nPacketSize = MV_CC_GetOptimalPacketSize(g_hHandle);
            if (nPacketSize > 0)
            {
```

```
                nRet =
MV_CC_SetIntValue(g_hHandle,"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);
            }
        }

        // Register the exception callback
        nRet = MV_CC_RegisterExceptionCallBack(g_hHandle, cbException, NULL);
        if (MV_OK != nRet)
        {
            printf("MV_CC_RegisterExceptionCallBack fail! nRet [%x]\n", nRet);
            continue;
        }
        printf("connect succeed\n");
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(g_hHandle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            continue;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , NULL);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            continue;
        }
    }
    return 0;
}
int main()
{
    int nRet = MV_OK;
    MV_CC_DEVICE_INFO_LIST stDeviceList = {0};
    unsigned int nSelectNum = 0;
    // 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;
    }
```

```
    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;
    }
    printf("Please Intput camera index: ");
    scanf("%d", &nSelectNum);
    if (nSelectNum >= stDeviceList.nDeviceNum)
    {
        printf("Intput error!\n");
        return -1;
    }
    if (stDeviceList.pDeviceInfo[nSelectNum]->nTLayerType == MV_GIGE_DEVICE)
    {
        memcpy(g_strSerialNumber, stDeviceList.pDeviceInfo[nSelectNum]-
>SpecialInfo.stGigEInfo.chSerialNumber,
            sizeof(stDeviceList.pDeviceInfo[nSelectNum]-
>SpecialInfo.stGigEInfo.chSerialNumber));
    }
    else if (stDeviceList.pDeviceInfo[nSelectNum]->nTLayerType == MV_USB_DEVICE)
    {
        memcpy(g_strSerialNumber, stDeviceList.pDeviceInfo[nSelectNum]-
>SpecialInfo.stUsb3VInfo.chSerialNumber,
            sizeof(stDeviceList.pDeviceInfo[nSelectNum]-
>SpecialInfo.stUsb3VInfo.chSerialNumber));
    }
    pthread_t nThreadID;
    nRet = pthread_create(&nThreadID, NULL, ReconnectProcess, NULL);
    if (nRet != 0)
    {
        printf("thread create failed nRet = %d\n",nRet);
        return -1;
    }
    PressEnterToExit();
    g_bConnect = false;
    // Shut down the device
    nRet = MV_CC_CloseDevice(g_hHandle);
    // Destroy the handle
    nRet = MV_CC_DestroyHandle(g_hHandle);
```

```
    printf("exit\n");
    return 0;
}
```

B.14 Set The Camera IO Status

The sample code below shows how to get or set the camera IO status.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
```

```
        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
    {
        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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
```

```
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Get the LineSelector
        MVCC_ENUMVALUE stLineSelector = {0};
        nRet = MV_CC_GetEnumValue(handle, "LineSelector", &stLineSelector);
        if (MV_OK == nRet)
        {
            printf("stLineSelector current value:%d\n",
stLineSelector.nCurValue);
            printf("supported stLineSelector number:%d\n",
stLineSelector.nSupportedNum);
            for (unsigned int i = 0; i < stLineSelector.nSupportedNum; ++i)
            {
                printf("supported stLineSelector [%d]:%d\n", i,
stLineSelector.nSupportValue[i]);
            }
            printf("\n");
        }
        else
        {
            printf("get stLineSelector failed! nRet [%x]\n\n", nRet);
        }
        // Set the LineSelector
        unsigned int nLineSelector = 0;
        printf("please input the LineSelector to set: ");
        scanf("%d", &nLineSelector);
        nRet = MV_CC_SetEnumValue(handle, "LineSelector", nLineSelector);
        if (MV_OK == nRet)
        {
            printf("set LineSelector OK!\n\n");
        }
        else
        {
            printf("set LineSelector failed! nRet [%x]\n\n", nRet);
        }
        // Get the LineMode
        MVCC_ENUMVALUE stLineMode = {0};
        nRet = MV_CC_GetEnumValue(handle, "LineMode", &stLineMode);
        if (MV_OK == nRet)
        {
            printf("stLineMode current value:%d\n", stLineMode.nCurValue);
            printf("supported stLineSelector number:%d\n",
```

```
stLineMode.nSupportedNum);
            for (unsigned int i = 0; i < stLineMode.nSupportedNum; ++i)
            {
                printf("supported stLineSelector [%d]:%d\n", i,
stLineMode.nSupportValue[i]);
            }
            printf("\n");
        }
        else
        {
            printf("get stLineMode failed! nRet [%x]\n\n", nRet);
        }
        // Set the LineMode
        unsigned int nLineMode = 0;
        printf("please input the LineMode to set:");
        scanf("%d", &nLineMode);
        nRet = MV_CC_SetEnumValue(handle, "LineMode", nLineMode);
        if (MV_OK == nRet)
        {
            printf("set LineMode OK!\n\n");
        }
        else
        {
            printf("set LineMode failed! nRet [%x]\n\n", nRet);
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.15 Set Camera Parameters

The sample code below shows how to set the camera parameters, including types of int, float, enum, bool, and so on.

```
#include <stdio.h>
#include <string.h>
#include "MvCameraControl.h"
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        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
    {
        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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
```

```
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Get the int type variable
        MVCC_INTVALUE stHeight = {0};
        nRet = MV_CC_GetIntValue(handle, "Height", &stHeight);
        if (MV_OK == nRet)
        {
            printf("height current value:%d\n", stHeight.nCurValue);
            printf("height max value:%d\n", stHeight.nMax);
            printf("height min value:%d\n", stHeight.nMin);
            printf("height increment value:%d\n\n", stHeight.nInc);
        }
        else
        {
            printf("get height failed! nRet [%x]\n\n", nRet);
        }
        // Set the int type variable
        unsigned int nHeightValue = 0;
        printf("please input the height to set:");
        scanf("%d", &nHeightValue);
        // Step (16) should be considered when setting width and height, that
is the width and height should be a multiple of 16
        nRet = MV_CC_SetIntValue(handle, "Height", nHeightValue);
        if (MV_OK == nRet)
        {
            printf("set height OK!\n\n");
        }
        else
        {
            printf("set height failed! nRet [%x]\n\n", nRet);
        }
        // Get the float type variable
        MVCC_FLOATVALUE stExposureTime = {0};
        nRet = MV_CC_GetFloatValue(handle, "ExposureTime", &stExposureTime);
        if (MV_OK == nRet)
        {
            printf("exposure time current value:%f\n",
stExposureTime.fCurValue);
            printf("exposure time max value:%f\n", stExposureTime.fMax);
            printf("exposure time min value:%f\n\n", stExposureTime.fMin);
        }
        else
        {
            printf("get exposure time failed! nRet [%x]\n\n", nRet);
        }
        // Set the float type variable
        // set IFloat variable
        float fExposureTime = 0.0f;
        printf("please input the exposure time to set: ");
```

```
        scanf("%f", &fExposureTime);
        nRet = MV_CC_SetFloatValue(handle, "ExposureTime", fExposureTime);
        if (MV_OK == nRet)
        {
            printf("set exposure time OK!\n\n");
        }
        else
        {
            printf("set exposure time failed! nRet [%x]\n\n", nRet);
        }
        // Get the enum type variable
        MVCC_ENUMVALUE stTriggerMode = {0};
        nRet = MV_CC_GetEnumValue(handle, "TriggerMode", &stTriggerMode);
        if (MV_OK == nRet)
        {
            printf("TriggerMode current value:%d\n", stTriggerMode.nCurValue);
            printf("supported TriggerMode number:%d\n",
stTriggerMode.nSupportedNum);
            for (unsigned int i = 0; i < stTriggerMode.nSupportedNum; ++i)
            {
                printf("supported TriggerMode [%d]:%d\n", i,
stTriggerMode.nSupportValue[i]);
            }
            printf("\n");
        }
        else
        {
            printf("get TriggerMode failed! nRet [%x]\n\n", nRet);
        }
        // Set the enum type variable
        unsigned int nTriggerMode = 0;
        printf("please input the TriggerMode to set:");
        scanf("%d", &nTriggerMode);
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", nTriggerMode);
        if (MV_OK == nRet)
        {
            printf("set TriggerMode OK!\n\n");
        }
        else
        {
            printf("set TriggerMode failed! nRet [%x]\n\n", nRet);
        }
        // Get the bool type variable
        bool bGetBoolValue = false;
        nRet = MV_CC_GetBoolValue(handle, "ReverseX", &bGetBoolValue);
        if (MV_OK == nRet)
        {
            if (0 != bGetBoolValue)
            {
                printf("ReverseX current is true\n\n");
            }
            else
```

```
        {
            printf("ReverseX current is false\n\n");
        }
    }
    // Set the bool type variable
    int nSetBoolValue;
    bool bSetBoolValue;
    printf("please input the ReverseX to set(bool): ");
    scanf("%d", &nSetBoolValue);
    if (0 != nSetBoolValue)
    {
        bSetBoolValue = true;
    }
    else
    {
        bSetBoolValue = false;
    }
    nRet = MV_CC_SetBoolValue(handle, "ReverseX", bSetBoolValue);
    if (MV_OK == nRet)
    {
        printf("Set ReverseX OK!\n\n");
    }
    else
    {
        printf("Set ReverseX Failed! nRet = [%x]\n\n", nRet);
    }
    // Get the string type variable
    MVCC_STRINGVALUE stStringValue = {0};
    nRet = MV_CC_GetStringValue(handle, "DeviceUserID", &stStringValue);
    if (MV_OK == nRet)
    {
        printf("Get DeviceUserID [%s]\n\n", stStringValue.chCurValue);
    }
    else
    {
        printf("Get DeviceUserID Failed! nRet = [%x]\n\n", nRet);
    }
    // Set the string type variable
    unsigned char strValue[256];
    printf("please input the DeviceUserID to set(string):");
    scanf("%s", strValue);
    nRet = MV_CC_SetStringValue(handle, "DeviceUserID", (char*)strValue);
    if (MV_OK == nRet)
    {
        printf("Set DeviceUserID OK!\n\n");
    }
    else
    {
        printf("Set DeviceUserID Failed! nRet = [%x]\n\n", nRet);
    }
    // Shut down the device
    nRet = MV_CC_CloseDevice(handle);
```

```
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.16 Get Images Directly in Triggering Mode

The sample code below shows how to get the image directly in triggering mode.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bExit = false;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
static void* WorkThread(void* pUser)
{
    int nRet = MV_OK;
    // Get the payload size
    MVCC_INTVALUE stParam;
    memset(&stParam, 0, sizeof(MVCC_INTVALUE));
    nRet = MV_CC_GetIntValue(pUser, "PayloadSize", &stParam);
    if (MV_OK != nRet)
    {
        printf("Get PayloadSize fail! nRet [0x%x]\n", nRet);
        return NULL;
    }
    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) *
stParam.nCurValue);
    if (NULL == pData)
    {
        return NULL;
```

```
    }
    unsigned int nDataSize = stParam.nCurValue;
    while(1)
    {
        nRet = MV_CC_SetCommandValue(pUser, "TriggerSoftware");
        if(MV_OK != nRet)
        {
            printf("failed in TriggerSoftware[%x]\n", nRet);
        }
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo,
1000);
        if (nRet == MV_OK)
        {
            printf("GetOneFrame, Width[%d], Height[%d], nFrameNum[%d]\n",
                stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else
        {
            printf("Get One Frame failed![%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    if (pData)
    {
        free(pData);
        pData = NULL;
    }
    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));
        // 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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
```

```
nPacketSize);
            }
        }

        nRet = MV_CC_SetBoolValue(handle, "AcquisitionFrameRateEnable", false);
        if (MV_OK != nRet)
        {
            printf("set AcquisitionFrameRateEnable fail! nRet [%x]\n", nRet);
            break;
        }

        // Set the trigger mode to on
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 1);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
            break;
        }
        // Set the trigger source
        nRet = MV_CC_SetEnumValue(handle, "TriggerSource",
MV_TRIGGER_SOURCE_SOFTWARE);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerSource fail! nRet [%x]\n", nRet);
            break;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
```

```
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```

B.17 Get Images via Callback in Triggering Mode

The sample code below shows how to get images via the callback in triggering mode.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include "MvCameraControl.h"
bool g_bIsGetImage = true;
bool g_bExit = false;
// Wait for the user to press Enter to stop grabbing or end the program
void PressEnterToExit(void)
{
    int c;
    while ( (c = getchar()) != '\n' && c != EOF );
    fprintf( stderr, "\nPress enter to exit.\n");
    while( getchar() != '\n');
    g_bExit = true;
    sleep(1);
}
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 the IP address and user defined name of the current camera
        printf("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stGigEInfo.chModelName);
        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("Device Model Name: %s\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chModelName);
        printf("UserDefinedName: %s\n\n", pstMVDevInfo-
>SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
void __stdcall ImageCallBackEx(unsigned char * pData, MV_FRAME_OUT_INFO_EX*
pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        printf("GetOneFrame, Width[%d], Height[%d], nFrameNum[%d]\n",
            pFrameInfo->nWidth, pFrameInfo->nHeight, pFrameInfo->nFrameNum);
        g_bIsGetImage = true;
    }
}
static void* WorkThread(void* pUser)
{
    while(1)
    {
        if(g_bExit)
        {
            break;
        }
        if (true == g_bIsGetImage)
        {
```

```
            int nRet = MV_CC_SetCommandValue(pUser, "TriggerSoftware");
            if(MV_OK != nRet)
            {
                printf("failed in TriggerSoftware[%x]\n", nRet);
            }
            else
            {
                g_bIsGetImage = false;
            }
        }
        else
        {
            continue;
        }
    }
}
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));
        // 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);
            break;
        }
        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");
            break;
        }
        printf("Please Intput camera index: ");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
```

```
        if (nIndex >= stDeviceList.nDeviceNum)
        {
            printf("Intput error!\n");
            break;
        }
        // Create a handle for the selected device
        nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CreateHandle fail! nRet [%x]\n", nRet);
            break;
        }
        // Open the device
        nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_OpenDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Detect the optimal packet size (it is valid for GigE cameras 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]!\n",
nRet);
                }
            }
            else
            {
                printf("Warning: Get Packet Size fail nRet [0x%x]!\n",
nPacketSize);
            }
        }

        nRet = MV_CC_SetBoolValue(handle, "AcquisitionFrameRateEnable", false);
        if (MV_OK != nRet)
        {
            printf("set AcquisitionFrameRateEnable fail! nRet [%x]\n", nRet);
            break;
        }

        // Set the trigger mode to on
        nRet = MV_CC_SetEnumValue(handle, "TriggerMode", 1);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerMode fail! nRet [%x]\n", nRet);
```

```
            break;
        }
        // Set the trigger source
        nRet = MV_CC_SetEnumValue(handle, "TriggerSource",
MV_TRIGGER_SOURCE_SOFTWARE);
        if (MV_OK != nRet)
        {
            printf("MV_CC_SetTriggerSource fail! nRet [%x]\n", nRet);
            break;
        }
        // Register the image callback function
        nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBackEx, handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_RegisterImageCallBackEx fail! nRet [%x]\n", nRet);
            break;
        }
        // Start grabbing images
        nRet = MV_CC_StartGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StartGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        pthread_t nThreadID;
        nRet = pthread_create(&nThreadID, NULL ,WorkThread , handle);
        if (nRet != 0)
        {
            printf("thread create failed.ret = %d\n",nRet);
            break;
        }
        PressEnterToExit();
        // Stop grabbing images
        nRet = MV_CC_StopGrabbing(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_StopGrabbing fail! nRet [%x]\n", nRet);
            break;
        }
        // Shut down the device
        nRet = MV_CC_CloseDevice(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_CloseDevice fail! nRet [%x]\n", nRet);
            break;
        }
        // Destroy the handle
        nRet = MV_CC_DestroyHandle(handle);
        if (MV_OK != nRet)
        {
            printf("MV_CC_DestroyHandle fail! nRet [%x]\n", nRet);
            break;
```

```
        }
    } while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("exit\n");
    return 0;
}
```