

Machine Vision Camera SDK (C)

Developer Guide

Legal Information

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

Contents

Chapter 1 Overview 1

1.1 Introduction 1

1.2 Integrated Development Environment 1

1.3 Update History 3

1.4 Notice 4

Chapter 2 Programming Guideline 5

2.1 Connect Device 5

2.2 Image Acquisition and Display 7

2.2.1 Get Image Directly 7

2.2.2 Get Image in Callback Function 10

Chapter 3 API Reference 14

3.1 General 14

3.1.1 MV_CC_GetSDKVersion 14

3.1.2 MV_CC_EnumerateTls 14

3.1.3 MV_CC_EnumDevices 15

3.1.4 MV_CC_EnumDevicesEx 16

3.1.5 MV_CC_IsDeviceAccessible 17

3.1.6 MV_CC_SetSDKLogPath 20

3.1.7 MV_CC_CreateHandle 22

3.1.8 MV_CC_CreateHandleWithoutLog 23

3.1.9 MV_CC_DestroyHandle 25

3.1.10 MV_CC_OpenDevice 27

3.1.11 MV_CC_CloseDevice 30

3.1.12 MV_CC_GetDeviceInfo 31

3.2 Configuration and Maintenance 34

3.2.1 MV_CC_GetIntValueEx 34

3.2.2 MV_CC_SetIntValueEx 36
3.2.3 MV_CC_GetEnumValue 38
3.2.4 MV_CC_SetEnumValue 41
3.2.5 MV_CC_SetEnumValueByString 43
3.2.6 MV_CC_GetFloatValue 47
3.2.7 MV_CC_SetFloatValue 50
3.2.8 MV_CC_GetBoolValue 52
3.2.9 MV_CC_SetBoolValue 54
3.2.10 MV_CC_GetStringValue 57
3.2.11 MV_CC_SetStringValue 59
3.2.12 MV_CC_SetCommandValue 61
3.2.13 MV_CC_ReadMemory 64
3.2.14 MV_CC_WriteMemory 66
3.2.15 MV_CC_LocalUpgrade 69
3.2.16 MV_CC_GetUpgradeProcess 71
3.2.17 MV_XML_GetGenICamXML 73
3.3 Functional 76
3.3.1 MV_CC_IsDeviceConnected 76
3.3.2 MV_CC_SetImageNodeNum 78
3.3.3 MV_CC_GetAllMatchInfo 81
3.3.4 MV_CC_InvalidateNodes 84
3.3.5 MV_CC_RegisterExceptionCallBack 86
3.3.6 MV_CC_RegisterAllEventCallBack 89
3.3.7 MV_CC_RegisterEventCallBackEx 91
3.3.8 MV_CC_FeatureSave 94
3.3.9 MV_CC_FeatureLoad 97
3.3.10 MV_CC_FileAccessRead 101
3.3.11 MV_CC_FileAccessWrite 103

3.3.12 MV_CC_GetFileAccessProgress 105
3.3.13 MV_GIGE_ForceIpEx 110
3.3.14 MV_GIGE_SetIpConfig 112
3.3.15 MV_GIGE_SetNetTransMode 114
3.3.16 MV_GIGE_GetNetTransInfo 116
3.3.17 MV_CC_GetOptimalPacketSize 120
3.3.18 MV_GIGE_SetResend 122
3.3.19 MV_GIGE_GetResendMaxRetryTimes 124
3.3.20 MV_GIGE_SetResendMaxRetryTimes 125
3.3.21 MV_GIGE_GetResendTimeInterval 125
3.3.22 MV_GIGE_SetResendTimeInterval 126
3.3.23 MV_GIGE_GetGvspTimeout 126
3.3.24 MV_GIGE_SetGvspTimeout 127
3.3.25 MV_GIGE_SetTransmissionType 127
3.3.26 MV_GIGE_IssueActionCommand 132
3.4 Image Acquisition 138
3.4.1 MV_CC_RegisterImageCallBackEx 138
3.4.2 MV_CC_RegisterImageCallBackForRGB 141
3.4.3 MV_CC_RegisterImageCallBackForBGR 145
3.4.4 MV_CC_StartGrabbing 148
3.4.5 MV_CC_StopGrabbing 151
3.4.6 MV_CC_GetImageForRGB 153
3.4.7 MV_CC_GetImageForBGR 157
3.4.8 MV_CC_GetImageBuffer 160
3.4.9 MV_CC_FreeImageBuffer 163
3.4.10 MV_CC_GetOneFrameTimeout 166
3.5 Image Processing 169
3.5.1 MV_CC_DisplayOneFrame 169

3.5.2 MV_CC_SaveImageEx2 172
3.5.3 MV_CC_ConvertPixelType 176
3.5.4 MV_CC_SetBayerCvtQuality 180
3.5.5 MV_CC_InputOneFrame 184
3.5.6 MV_CC_StartRecord 189
3.5.7 MV_CC_StopRecord 194
3.6 Advanced Settings 199
3.6.1 MV_GIGE_SetGvcpTimeout 199
Chapter 4 Data Structure and Enumeration 202
4.1 Data Structure 202
4.1.1 MVCC_ENUMVALUE 202
4.1.2 MVCC_FLOATVALUE 202
4.1.3 MVCC_INTVALUE_EX 203
4.1.4 MVCC_STRINGVALUE 204
4.1.5 MV_ACTION_CMD_INFO 204
4.1.6 MV_ACTION_CMD_RESULT 205
4.1.7 MV_ACTION_CMD_RESULT_LIST 205
4.1.8 MV_ALL_MATCH_INFO 206
4.1.9 MV_CamL_DEV_INFO 207
4.1.10 MV_CC_DEVICE_INFO 208
4.1.11 MV_CC_DEVICE_INFO_LIST 209
4.1.12 MV_CC_FILE_ACCESS 210
4.1.13 MV_CC_FILE_ACCESS_PROGRESS 210
4.1.14 MV_CC_PIXEL_CONVERT_PARAM 211
4.1.15 MV_DISPLAY_FRAME_INFO 213
4.1.16 MV_EVENT_OUT_INFO 213
4.1.17 MV_FRAME_OUT 215
4.1.18 MV_FRAME_OUT_INFO 215

4.1.19 MV_FRAME_OUT_INFO_EX 216
4.1.20 MV_GIGE_DEVICE_INFO 218
4.1.21 MV_IMAGE_BASIC_INFO 220
4.1.22 MV_MATCH_INFO_NET_DETECT 221
4.1.23 MV_MATCH_INFO_USB_DETECT 222
4.1.24 MV_NETTRANS_INFO 222
4.1.25 MV_SAVE_IMAGE_PARAM_EX 223
4.1.26 MV_TRANSMISSION_TYPE 224
4.1.27 MV_USB3_DEVICE_INFO 225
4.2 Enumeration 226
4.2.1 MV_GIGE_EVENT 226
4.2.2 MV_GIGE_TRANSMISSION_TYPE 227
4.2.3 MV_SAVE_IAMGE_TYPE 228
4.2.4 MvGvspPixelType 229
Chapter 5 FAQ (Frequently Asked Questions) 231
5.1 GigE Camera 231
5.1.1 Why is there packet loss? 231
5.1.2 Why does link error occur in the normal compiled Demo? 231
5.1.3 Why can't I set the static IP under DHCP? 232
5.1.4 Why do I failed to perform the software trigger command when calling SDK? 232
5.1.5 Why does the camera often be offline? 232
5.1.6 Why is no permission returned when calling API MV_CC_OpenDevice 232
5.1.7 Why is there error code returned during debug process? 233
5.1.8 Why is no data error returned when calling API MV_CC_GetOneFrameTimeout? .. 233
5.1.9 Why is there always no data when calling MV_CC_GetOneFrameTimeout? 233
5.1.10 How to fix error "ld:-lMvCameraControl cannot find libMvCameraControl library" when compiling? 234
5.1.11 How to fix error "ld:-lMvCameraControl not compatable symbol" when compiling? 234

5.1.12 Why can't I enumerate the GigE cameras? 234

5.2 USB3Vision Camera 235

5.2.1 Why can't the MVS get the data or why is the frame rate far smaller the actual frame rare? 235

Appendix A. Appendixes 236

A.1 Error Code 236

A.2 MvCameraNode 238

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 16.04 (32-bit or 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). **Note** RaspberryPiB3.0+ and ODROID-XU4 only support GigE camera.

MVS Installation

1. Get the system root authority before Machine Vision Software (MVS) installation.
2. Unzip the MVS installation package by "tar-xzvf".

Note

The MVS installation packages are different of different hardware environment (x86_64, i386, armhf, aarch64), you should select suitable installation package according to actual hardware environment, for details contact the technical supporter.

3. Execute the script "source ./setup.sh" to install MVS.
4. Run ./MVS.sh under the directory "/opt/MVS/bin/MVSPlayer" to check if MVS is installed properly.

Development Folder Contents

By default, the Machine Vision Software (MVS) is installed in the path of /opt/MVS. After installation, 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

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 camera, it is recommended to enable jumbo frame, set MTU to 9000 and increase socket buffer (set_socket_buffer_size.sh).
- For USB 3.0 camera, 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 3.1.0_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*** .

Version	Content
	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

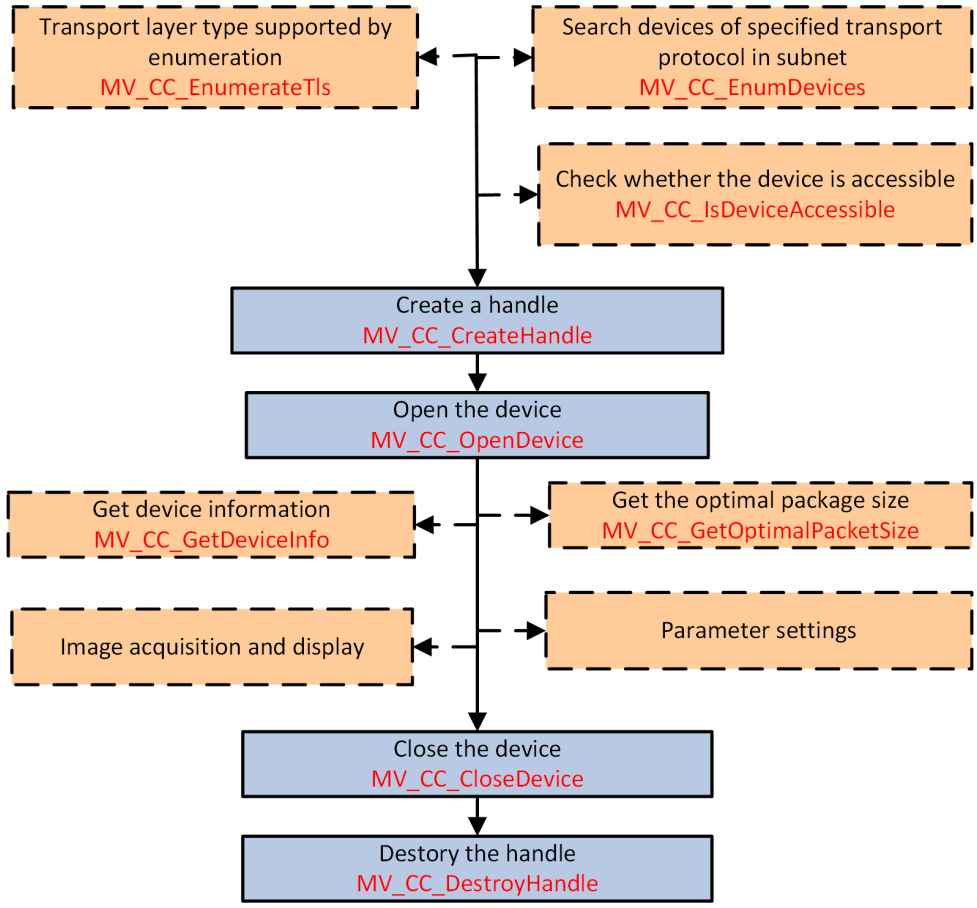


Figure 2-1 Programming Flow of Connecting Device

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

Example

Sample Code of Connecting Device

```
#include "MvCameraControl.h"
void main()
{
    int nRet = -1;
    void*  m_handle = NULL;

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

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

    //Select the searched first online device, create a device handle
    int nDeviceIndex = 0;

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect the device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
```

```
    unsigned short nSwitchoverKey = 0;
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }

    //...Other processing method

    //Close the device, release the resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy the handle, release the resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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*** .

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

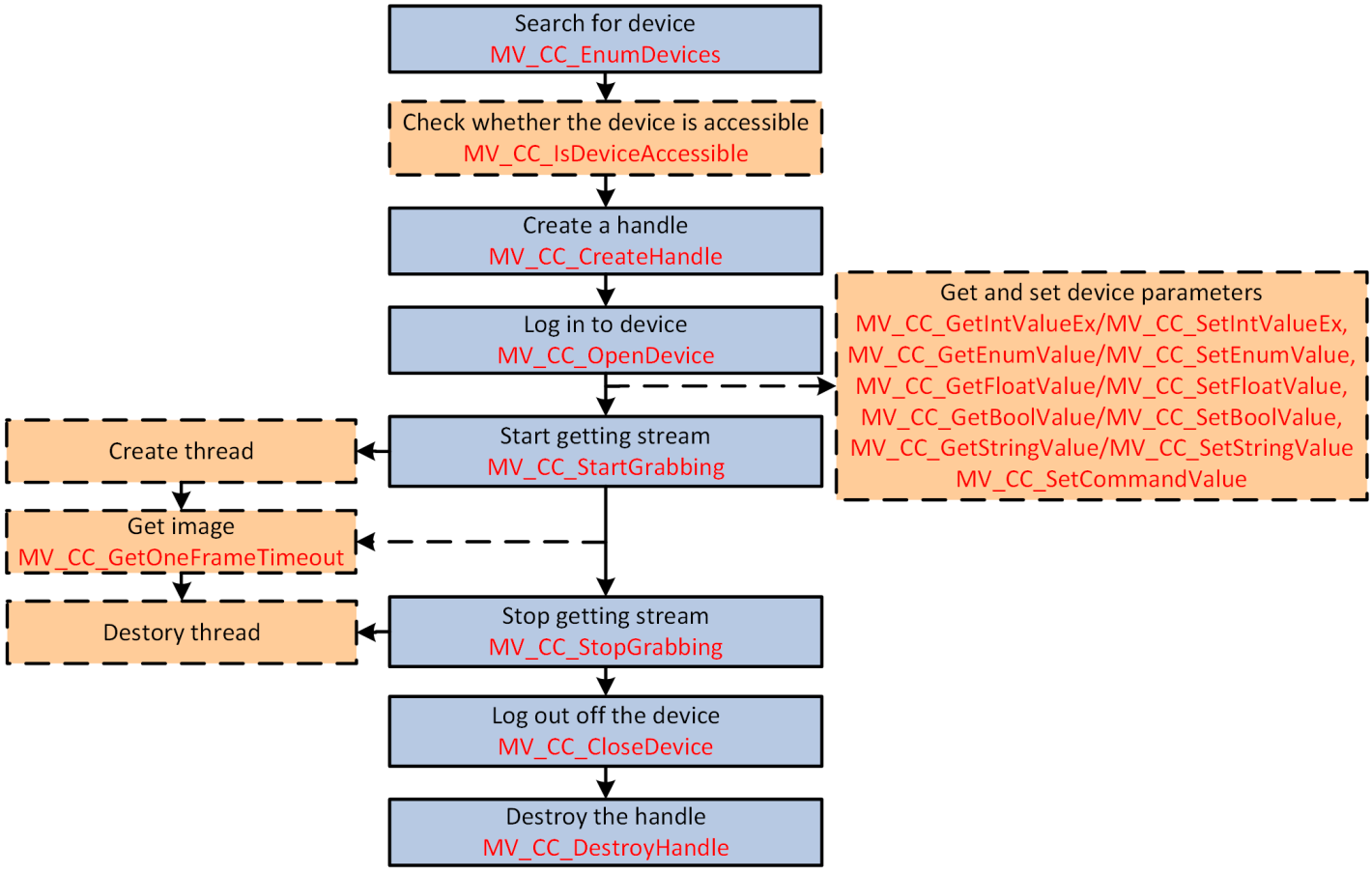


Figure 2-2 Programming Flow of Getting Image Directly

Note

All the open properties of the camera can be referred to the node sheet " ***MvCameraNode*** .xlsx" under the installation directory. The name, definition, data type, value range, and access mode of each node are described in the sheet.

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_StartGrabbing*** to start getting streams.

Note

- Before starting the acquisition, you can call ***MV_CC_SetImageNodeNum*** to set the number of image cache 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.

2) Call ***MV_CC_GetOneFrameTimeout*** repeatedly in the application layer to get the frame data with specified pixel format.

Note

When getting the frame data, the application program should control the frequency of calling this API according to the frame rate.

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.

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

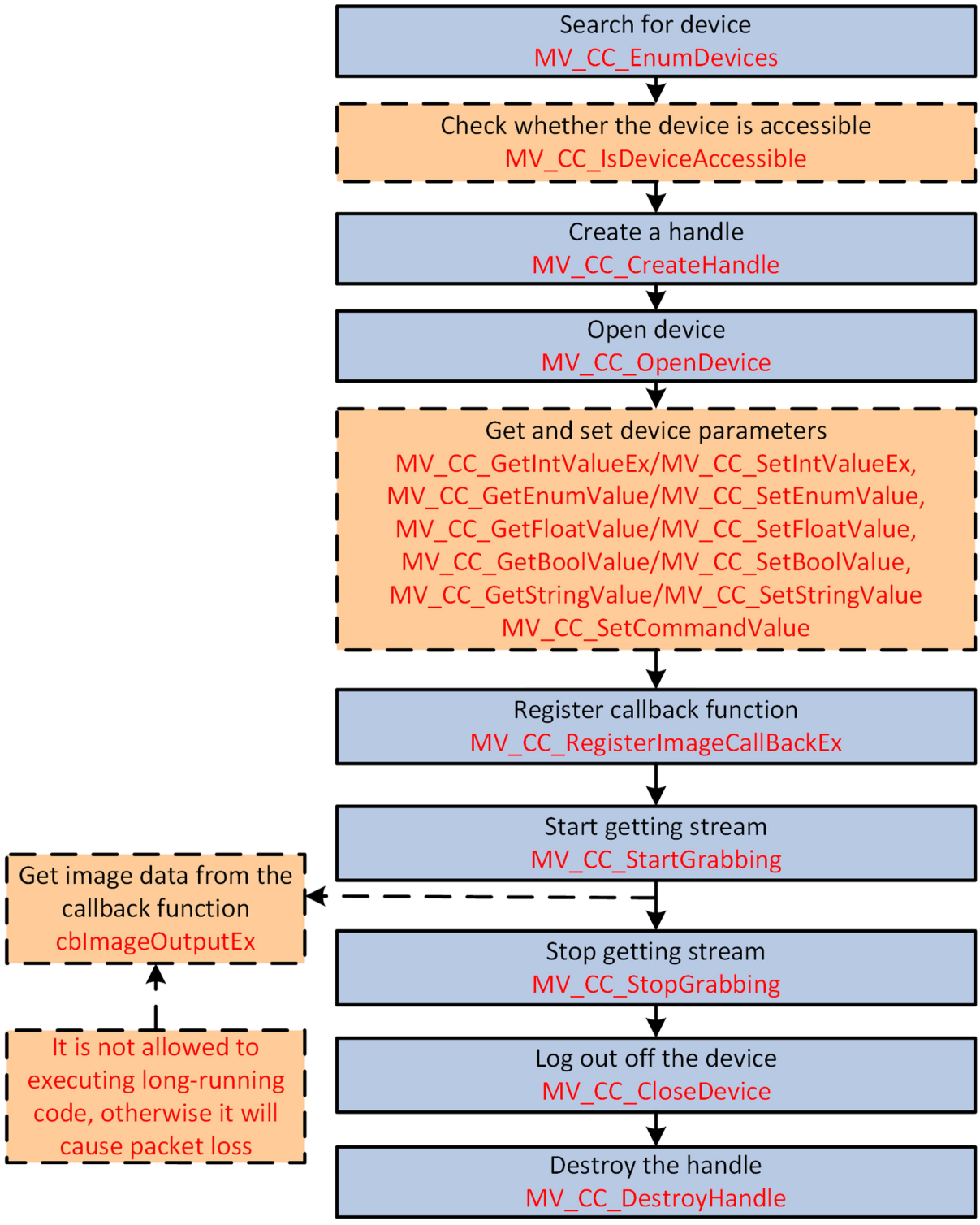


Figure 2-3 Programming Flow of Getting Image in Callback Function

Note

All the open properties of the camera can be referred to the node sheet " ***MvCameraNode*** .xlsx" under the installation directory. The name, definition, data type, value range, and access mode of each node are described in the sheet.

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 cache 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
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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
   unsigned int nTransLayers = MV_CC_EnumerateTls();
   if (nTransLayers & MV_GIGE_DEVICE)
   {
      OutputString(_T("MV_GIGE_DEVICE"));
   }
}
```

3.1.3 MV_CC_EnumDevices

Enumerate all devices corresponding to specified transport protocol within 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;

    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
    if (MV_OK != nRet)
    {
        printf("error: EnumDevices fail [%x]\n", nRet);
    }
}
```

3.1.4 MV_CC_EnumDevicesEx

Enumerate all the devices of specified transport protocol and manufacturer in 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
   unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;

   MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
   int nRet = MV_CC_EnumDevicesEx(nTLayerType, &m_stDevList, "abc");
   if (MV_OK != nRet)
   {
      printf("error: EnumDevicesEx fail [%x]\n", nRet);
   }
}
```

3.1.5 MV_CC_IsDeviceAccessible

Whether 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Determine whether specified device is accessible
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    nRet = MV_CC_IsDeviceAccessible(&m_stDevInfo, nAccessMode);
    if (MV_OK != nRet)
```

```
    {
        printf("error: IsDeviceAccessible fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
{
    int nRet = -1;
    void* m_handle = NULL;
    //Enumerate all devices of specific transport protocol in the subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
```

```
MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}
int i = 0;
if (m_stDevList.nDeviceNum == 0)
{
   printf("no camera found!\n");
   return;
}
string strPath = "D:/SDK";
nRet = MV_CC_SetSDKLogPath(strPath.c_str());
if (MV_OK != nRet)
{
   printf("error: Set SDK Log Path fail [%x]\n", nRet);
   return;
}
//Select the first found online device and create handle
int nDeviceIndex = 0;
MV_CC_DEVICE_INFO m_stDevInfo = {0};
memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], sizeof(MV_CC_DEVICE_INFO));
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
if (MV_OK != nRet)
{
    printf("error: CreateHandle fail [%x]\n", nRet);
    return;
}
//Connect to device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;
nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
}
//Other process...

//Close camera
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
     printf("error: CloseDevice fail [%x]\n", nRet);
     return;
}
//Destroy handle and release resources
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
{
```

```
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //...other processing

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandleWithoutLog(&m_handle, &m_stDevInfo);

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

    //...other processing

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    else
    {
        //...other processing
    }
```

```
    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.1.10 MV_CC_OpenDevice

Open the device (connect 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

Macro Definition	Value	Description
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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

    int i = 0;
```

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

    //Connect device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.1.11 MV_CC_CloseDevice

Shut down 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
```

```
    printf("no camera found!\n");
    return;
  }

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

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

  nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

  //Shut device and release resource
  nRet = MV_CC_CloseDevice(m_handle);
  if (MV_OK != nRet)
  {
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
  }

  //Destroy handle and release resource
  nRet = MV_CC_DestroyHandle(m_handle);
  if (MV_OK != nRet)
  {
    printf("error: DestroyHandle fail [%x]\n", nRet);
    return;
  }
}
```

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 camera.
- The API is not supported by CameraLink device.

See Also

MV_CC_OpenDevice

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

   int i = 0;
   if (m_stDevList.nDeviceNum == 0)
   {
```

```
    printf("no camera found!\n");
    return;
}

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

//Get device information
MV_CC_DEVICE_INFO struDeviceInfo;
memset(&deviceInfo, 0, sizeof(MV_CC_DEVICE_INFO));
nRet = MV_CC_GetDeviceInfo(m_handle, &struDeviceInfo);
if (MV_OK != nRet)
{
    printf("error: GetDeviceInfo[%x]\n", nRet);
    return;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
{
```

```
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.2 Configuration and Maintenance

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, refer to ***MvCameraNode*** for details

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. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

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

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect to device
     unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

     //Get int parameters
     MVCC_INTVALUE_EX struIntValue = {0};

    nRet = MV_CC_GetIntValueEx(m_handle, "Width", &struIntValue);
```

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

    //...other processing

     //Shut device and release resource
     nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resources
     nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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)* on success, and return ***Error Code*** on failure.

Remarks

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

   //Connect to device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
```

```
    unsigned short nSwitchoverKey = 0;

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

     //Set int type parameters
     int64_t nValue = 752;

    nRet = MV_CC_SetIntValueEx(m_handle, "GevMACAddress", nValue);
    if (MV_OK != nRet)
    {
        printf("error: SetIntValueEx fail [%x]\n", nRet);
        return;
    }

    //...other processing

     //Shut device and release resource
     nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resources
     nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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, refer to ***MvCameraNode*** for details

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. For value of **strKey**, refer to ***MvCameraNode*** . The node values of IEnumeration can be obtained through this API, **strKey** value corresponds to the Name column.

See Also

MV_CC_SetEnumValue

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Get Enum parameters
MVCC_ENUMVALUE struEnumValue = {0};

nRet = MV_CC_GetEnumValue(m_handle, "TriggerSource", &struEnumValue);
if (MV_OK != nRet)
{
   printf("error: GetEnumValue fail [%x]\n", nRet);
   return;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
   return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
{
   printf("error: DestroyHandle fail [%x]\n", nRet);
   return;
```

```
    }
}
```

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, refer to ***MvCameraNode*** for details

nValue

[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 specified Enum node after connecting the device. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Set Enum parameters
unsigned int enMode = MV_TRIGGER_SOURCE_SOFTWARE; //Set trigger as soft trigger

nRet = MV_CC_SetEnumValue(m_handle, "TriggerSource", enMode);
if (MV_OK != nRet)
{
   printf("error: SetEnumValue fail [%x]\n", nRet);
   return;
}
```

```
    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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, refer to ***MvCameraNode*** for details

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. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

// Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
        if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x000000ff);

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

```
    }
    return true;
}

void __stdcall EventCallBack(MV_EVENT_OUT_INFO * pEventInfo, void* pUser)
{
    if (pEventInfo)
    {
        __int64 nBlockId = pEventInfo->nBlockIdHigh;
        nBlockId = (nBlockId << 32) + pEventInfo->nBlockIdLow;
        __int64 nTimestamp = pEventInfo->nTimestampHigh;
        nTimestamp = (nTimestamp << 32) + pEventInfo->nTimestampLow;
        printf("EventName[%s], EventID[%u], BlockId[%I64d], Timestamp[%I64d]\n";
        pEventInfo->EventName, pEventInfo->nEventID,nBlockId,nTimestamp);
     }
}

int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {
        //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;
}
//Select device and create handle
nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
if (MV_OK != nRet)
{
    printf("Create Handle fail! nRet [0x%x]\n", nRet);
    break;
}
//Open camera
nRet = MV_CC_OpenDevice(handle);
if (MV_OK != nRet)
{
    printf("Open Device fail! nRet [0x%x]\n", nRet);
    break;
}
nRet = MV_CC_SetEnumValue(handle, "TriggerMode", MV_TRIGGER_MODE_OFF);
if (MV_OK != nRet)
{
    printf("Set Trigger Mode fail! nRet [0x%x]\n", nRet);
    break
}
//Set Event of ExposureEnd On
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 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 image acquisition
nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
    printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
    break;
```

```
        }
        printf("Press a key to stop grabbing.\n");
        WaitForKeyPress();

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

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

        if (nRet != MV_OK)
        {
            if (handle != NULL)
            {
                MV_CC_DestroyHandle(handle);
                handle = NULL;
            }
        }
        printf("Press a key to exit.\n");
        WaitForKeyPress();
        return 0;
    }
}
```

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, refer to ***MvCameraNode*** for details

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. For value of **strKey**, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

    int i = 0;
    if (m_stDevList.nDeviceNum == 0)
    {
```

```
    printf("no camera found!\n");
    return;
}

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Get float parameters
MVCC_FLOATVALUE struFloatValue = {0};

nRet = MV_CC_GetFloatValue(m_handle, "TriggerDelay", &struFloatValue);
if (MV_OK != nRet)
{
    printf("error: GetFloatValue fail [%x]\n", nRet);
    return;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
```

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

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, refer to ***MvCameraNode*** for details.

fValue

[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 specified float node after connecting the device. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void main()
```

```
{
    int nRet = -1;
    void* m_handle = NULL;

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

    //Set float parameters
    float fValue = 0.0;

    nRet = MV_CC_SetFloatValue(m_handle, "TriggerDelay", fValue);
    if (MV_OK != nRet)
```

```
    {
        printf("error: SetFloatValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.2.8 MV_CC_GetBoolValue

Get the value of camera bool type node.

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)* on success, and return ***Error Code*** on failure.

Remarks

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

See Also

MV_CC_SetBoolValue

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

```
    }

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

    //Get bool parameters
    bool bValue = 0;

    nRet = MV_CC_GetBoolValue(m_handle, "ReverseX", &bValue);
    if (MV_OK != nRet)
    {
        printf("error: GetBoolValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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. For **strKey** value, refer to ***MvCameraNode*** . All the node values of "IBoolean" can be set via this API. **strKey** corresponds to the Name column.

See Also

MV_CC_GetBoolValue

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

```
}

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Set bool parameters
bool nValue = TRUE;

nRet = MV_CC_SetBoolValue(m_handle, "ReverseX", nValue);
if (MV_OK != nRet)
{
   printf("error: SetBoolValue fail [%x]\n", nRet);
   return;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
```

```
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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, refer to ***MvCameraNode*** for details

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. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

```
    //Get string parameters
    MVCC_STRINGVALUE struStringValue = {0};

    nRet = MV_CC_GetStringValue(m_handle, "DeviceUserID", &struStringValue);
    if (MV_OK != nRet)
    {
        printf("error: GetStringValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.2.11 MV_CC_SetStringValue

Set the value of camera string type node.

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, refer to ***MvCameraNode*** for details

sValue

[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 specified string type node after connecting the device. For **strKey** value, refer to ***MvCameraNode*** . 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
```

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

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

    //Set string parameters
    nRet = MV_CC_SetStringValue(m_handle, "DeviceUserID", "Camera");
    if (MV_OK != nRet)
    {
        printf("error: SetStringValue fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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, refer to ***MvCameraNode*** for details

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. For **strKey** value, refer to ***MvCameraNode*** . All the node values of "ICommand" in the list can be set via this API. **strKey** corresponds to the Name column.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Set Command node
nRet = MV_CC_SetCommandValue(m_handle, "TriggerSoftware");
if (MV_OK != nRet)
{
   printf("error: SetCommandValue fail [%x]\n", nRet);
   return;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
   return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
{
   printf("error: DestroyHandle fail [%x]\n", nRet);
   return;
```

```
    }
}
```

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

Example

The following sample code is for reference only.

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

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    else
    {
        unsigned int nAccessMode = MV_ACCESS_Exclusive;
        unsigned short nSwitchoverKey = 0;

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

    //Read data from device register
```

```
    unsigned int nWidth = 0;
    nRet = MV_CC_ReadMemory(m_handle, &nWidth, 0x00030360L, 4L);
    if (MV_OK != nRet)
    {
        printf("error: ReadMemory [%x]\n", nRet);
        break;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
```

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

else
{
   unsigned int nAccessMode = MV_ACCESS_Exclusive;
   unsigned short nSwitchoverKey = 0;

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

//Read data from device register
unsigned int nWidth = 0;
nRet = MV_CC_ReadMemory(m_handle, &nWidth, 0x00030360L, 4L);
if (MV_OK != nRet)
{
   printf("error: ReadMemory [%x]\n", nRet);
   break;
}

// Write data into device register
unsigned int nWidth = 320;
nRet = MV_CC_WriteMemory(m_handle, &nWidth, 0x00030360L, 4L);
if (MV_OK != nRet)
{
   printf("error: WriteMemory [%x]\n", nRet);
   break;
}

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
   return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
```

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

3.2.15 MV_CC_LocalUpgrade

Device local upgrade

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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


//Device local upgrade
nRet = MV_CC_LocalUpgrade(m_handle, "C://digicap.dav");
if (MV_OK != nRet)
{
   printf("error: LocalUpgrade fail [%x]\n", nRet);
   return;
}
```

```
    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

    //Device local upgrade
    nRet = MV_CC_LocalUpgrade(m_handle, "C://digicap.dav");
    if (MV_OK != nRet)
```

```
    {
        printf("error: LocalUpgrade fail [%x]\n", nRet);
        return;
    }

    //Get upgrade progress
    unsigned int iProcess = 0;
    while(1)
    {
        nRet = MV_CC_GetUpgradeProcess(m_handle, &iProcess);
        if(iProcess == 100)
        {
            printf("Upgrade finished!\n");
            break;
        }
        Sleep(1000);
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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**.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

#define MAX_XML_FILE_SIZE      (1024*1024*3)

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

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

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

```
    return;
}

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

unsigned char* pXMLBuf = (unsigned char*)malloc(MAX_XML_FILE_SIZE);
if (NULL == pXMLBuf)
{
    printf("error: malloc xml buffer failed!\n");
    break;
}

unsigned int nXMLDataLen = 0;
nRet = MV_XML_GetGenICamXML(m_handle, pXMLBuf, MAX_XML_FILE_SIZE, &nXMLDataLen);
if (MV_OK != nRet || nXMLDataLen > MAX_XML_FILE_SIZE)
{
    printf("error: GetGenICamXML failed! [%x]\n", nRet);
    break;
}

//Other process...

//Shut down device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}
```

```
    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3 Functional

3.3.1 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.

Example

The following sample code is for reference only.

```
#include <stdio.h>

#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

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

```
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {
        //Enumerate device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

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

        //Open device
          nRet = MV_CC_OpenDevice(handle);
        if (MV_OK != nRet)
        {
            printf("Open Device fail! nRet [0x%x]\n", nRet);
            break;
        }
        bool bIsConnect = MV_CC_IsDeviceConnected(handle);
        if (!bIsConnect)
        {
            printf("Device is connected!\n");
            break;
        }
        else
        {
            printf("Device is unconnected!\n");
            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;
}
```

3.3.2 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO* pFrameInfo, void* pUser)
```

```
{
    if (pFrameInfo)
    {
        // Add system time at output
        char   szInfo[128] = {0};
        SYSTEMTIME sys;
        GetLocalTime( &sys );
        sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] : GetOneFrame succeed, width[%d], height[%d]",
sys.wYear, sys.wMonth,
            sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds, pFrameInfo->nWidth, pFrameInfo-
>nHeight);

        printf("%s\n", szInfo);
    }
}

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

```
//Connect device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

//Set number of SDK internal image buffer nodes
unsigned int nNum = 15;
nRet = MV_CC_SetImageNodeNum(handle, nNum);
if (MV_OK != nRet)
{
   printf("error: SetImageNodeNum fail [%x]\n", nRet);
   return;
}

//Register data callback function, get acquired image data in the callback function
nRet = MV_CC_RegisterImageCallBack(handle, ImageCallBack, NULL);
if (MV_OK != nRet)
{
   printf("error: RegisterImageCallBack fail [%x]\n", nRet);
   return;
}

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

//...other processing

//Stop image acquiring
nRet = MV_CC_StopGrabbing(m_handle);
if (MV_OK != nRet)
{
   OutputString(_T("Operation: Stop Grabbing.....failed!"));
   return;
}

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
```

```
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.3 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    else
    {
        unsigned int nAccessMode = MV_ACCESS_Exclusive;
        unsigned short nSwitchoverKey = 0;

        //Connect device
        nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
        if (MV_OK != nRet)
        {
            printf("error: OpenDevice fail [%x]\n", nRet);
            return;
```

```
    }
}

//...other processing

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

Sleep(5000);

//Get camera network data flow and packet loss information
MV_ALL_MATCH_INFO struMatchInfo = {0};
MV_MATCH_INFO_NET_DETECT stMatchInfoNetDetect;
struMatchInfo.pInfo = &stMatchInfoNetDetect;

struMatchInfo.nType = MV_MATCH_TYPE_NET_DETECT;
memset(struMatchInfo.pInfo, 0, sizeof(MV_MATCH_INFO_NET_DETECT));
struMatchInfo.nInfoSize = sizeof(MV_MATCH_INFO_NET_DETECT);

nRet = MV_CC_GetAllMatchInfo(m_handle, &struMatchInfo);
if (MV_OK != nRet)
{
    printf("error: GetAllMatchInfo fail [%x]\n", nRet);
    return;
}

//...other processing

//Stop acquiring images
nRet = MV_CC_StopGrabbing(m_handle);
if (MV_OK != nRet)
{
    OutputString(_T("Operation: Stop Grabbing.....failed!"));
    return;
}

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
```

```
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.4 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

    int i = 0;
```

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect to device
 unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

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

 //Set int parameters
 int64_t nValue = 752;

nRet = MV_CC_SetIntValueEx(m_handle, "Width", nValue);
if (MV_OK != nRet)
{
    printf("error: SetIntValueEx fail [%x]\n", nRet);
    return;
}

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

//...other processing

 //Shut device and release resource
```

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

    //Destroy handle and release resources
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.5 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

//Exception callback function
void __stdcall ExceptionCB(unsigned int nMsgType, void* pUser)
{
   printf("Exception message is: 0x%x", nMsgType);
}

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

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

   int i = 0;
```

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

//...other processing

nRet = MV_CC_RegisterExceptionCallBack(m_handle, ExceptionCB, NULL);

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

//...other processing

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
```

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

3.3.6 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall EvCallback(MV_EVENT_OUT_INFO *pEventInfo, void* pUser)
{
   printf("EventName:+%s",pEventInfo->EventName);
}

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

   //Enumerate all devices of specific transport protocol in the subnet
   unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
   MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
   nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
   if (MV_OK != nRet)
   {
      printf("error: EnumDevices fail [%x]\n", nRet);
      return;
   }
   int i = 0;
   if (m_stDevList.nDeviceNum == 0)
   {
      printf("no camera found!\n");
      return;
   }

   //Select the first found online device and create handle
   int nDeviceIndex = 0;
   MV_CC_DEVICE_INFO m_stDevInfo = {0};
   memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], sizeof(MV_CC_DEVICE_INFO));
   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
   if (MV_OK != nRet)
   {
      printf("error: CreateHandle fail [%x]\n", nRet);
      return;
   }
```

```
    //Connect to device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    nRet = MV_CC_RegisterAllEventCallBack(m_handle, EvCallback, m_handle);
    if (MV_OK != nRet)
    {
        printf("Register Event CallBack fail[%x]\n", nRet);
        return;
    }
    //Other process...

    //Close camera
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.7 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall EvCallback(MV_EVENT_OUT_INFO *pEventInfo, void* pUser)
{
   printf("EventName:+%s",pEventInfo->EventName);
}

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

   //Enumerate all devices of specific transport protocol in the subnet
```

```
unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}
int i = 0;
if (m_stDevList.nDeviceNum == 0)
{
   printf("no camera found!\n");
   return;
}

//Select the first found online device and create handle
int nDeviceIndex = 0;
MV_CC_DEVICE_INFO m_stDevInfo = {0};
memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], sizeof(MV_CC_DEVICE_INFO));
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
if (MV_OK != nRet)
{
   printf("error: CreateHandle fail [%x]\n", nRet);
   return;
}

//Register data callback function
 nRet = MV_CC_RegisterEventCallBackEx(m_handle,"ExposureEnd", EvCallBack, NULL);
if (MV_OK != nRet)
{
   printf("error: RegisterImageCallBack fail [%x]\n", nRet);
   return;
}

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

//Other process...

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

```
    }
    //Other process...

     //Stop image acquisition
     nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close camera
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.8 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

//Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
        printf("The Pointer of pstMVDevInfo is NULL!\n");
        return false;
    }
    if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
    {
        int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0xff000000) >> 24);
        int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x00ff0000) >> 16);
        int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x0000ff00) >> 8);
        int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x000000ff);

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

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

        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
    }
    else
    {
        printf("Find No Devices!\n");
        break;
    }
    printf("Please Intput camera index:");
    unsigned int nIndex = 0;
    scanf("%d", &nIndex);
    if (nIndex >= stDeviceList.nDeviceNum)
    {
        printf("Intput error!\n");
        break;
    }
    //Select device and create handle
    nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
    if (MV_OK != nRet)
    {
        printf("Create Handle fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Open camera
    nRet = MV_CC_OpenDevice(handle);
    if (MV_OK != nRet)
```

```
    {
        printf("Open Device fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Export features from MVC as a file
    nRet = MV_CC_FeatureSave(handle, "FeatureFile.ini");
    if (MV_OK != nRet)
    {
        printf("Save Feature fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Import features to the MVC
    nRet = MV_CC_FeatureLoad(handle, "FeatureFile.ini");
    if (MV_OK != nRet)
    {
        printf("Load Feature fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Close device
    nRet = MV_CC_CloseDevice(handle);
    if (MV_OK != nRet)
    {
        printf("ClosDevice fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Destroy handle
    nRet = MV_CC_DestroyHandle(handle);
    if (MV_OK != nRet)
    {
        printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
        break;
    }
    while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

3.3.9 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

//Wait for key input
void WaitForKeyPress(void)
{
   while(!_kbhit())
   {
      Sleep(10);
   }
   _getch();
}
bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
   if (NULL == pstMVDevInfo)
   {
      printf("The Pointer of pstMVDevInfo is NULL!\n");
      return false;
   }
   if (pstMVDevInfo->nTLayerType == MV_GIGE_DEVICE)
   {
      int nIp1 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0xff000000) >> 24);
      int nIp2 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x00ff0000) >> 16);
      int nIp3 = ((pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x0000ff00) >> 8);
      int nIp4 = (pstMVDevInfo->SpecialInfo.stGigEInfo.nCurrentIp & 0x000000ff);
```

```
        //Print the current camera's IP address and user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo->SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n\n", pstMVDevInfo->SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {

        //Enumerate device
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (stDeviceList.nDeviceNum > 0)
        {
            for (unsigned int i = 0; i < stDeviceList.nDeviceNum; i++)
            {
                printf("[device %d]:\n", i);
                MV_CC_DEVICE_INFO* pDeviceInfo = stDeviceList.pDeviceInfo[i];
                if (NULL == pDeviceInfo)
                {
                    break;
                }
                PrintDeviceInfo(pDeviceInfo);
            }
        }
        else
        {
            printf("Find No Devices!\n");
            break;
        }
        printf("Please Intput camera index:");
        unsigned int nIndex = 0;
        scanf("%d", &nIndex);
        if (nIndex >= stDeviceList.nDeviceNum)
```

```
{
   printf("Intput error!\n");
   break;
}
//Select device and create handle
nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[nIndex]);
if (MV_OK != nRet)
{
   printf("Create Handle fail! nRet [0x%x]\n", nRet);
   break;
}
//Open camera
nRet = MV_CC_OpenDevice(handle);
if (MV_OK != nRet)
{
   printf("Open Device fail! nRet [0x%x]\n", nRet);
   break;
}
//Export features from MVC as a file
nRet = MV_CC_FeatureSave(handle, "FeatureFile.ini");
if (MV_OK != nRet)
{
    printf("Save Feature fail! nRet [0x%x]\n", nRet);
    break;
}
//Import features to the MVC
nRet = MV_CC_FeatureLoad(handle, "FeatureFile.ini");
if (MV_OK != nRet)
{
    printf("Load Feature fail! nRet [0x%x]\n", nRet);
    break;
}
//Close device
nRet = MV_CC_CloseDevice(handle);
if (MV_OK != nRet)
{
     printf("ClosDevice fail! nRet [0x%x]\n", nRet);
     break;
}
//Destroy handle
nRet = MV_CC_DestroyHandle(handle);
if (MV_OK != nRet)
{
     printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
     break;
}
while (0);
if (nRet != MV_OK)
{
     if (handle != NULL)
     {
         MV_CC_DestroyHandle(handle);
```

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

3.3.10 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

```
//Enumerate all devices of specific transport protocol in the subnet
unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}
int i = 0;
if (m_stDevList.nDeviceNum == 0)
{
   printf("no camera found!\n");
   return;
}

//Select the first found online device and create handle
int nDeviceIndex = 0;
MV_CC_DEVICE_INFO m_stDevInfo = {0};
memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], sizeof(MV_CC_DEVICE_INFO));
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
if (MV_OK != nRet)
{
   printf("error: CreateHandle fail [%x]\n", nRet);
   return;
}
//Connect to device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;
nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
   printf("error: OpenDevice fail [%x]\n", nRet);
   return;
}
MV_CC_FILE_ACCESS stMcFileAccess;
stMcFileAccess.pUserFileName =  "UserSet1.txt";
stMcFileAccess.pDevFileName =  "UserSet1";
nRet = MV_CC_FileAccessRead( m_handle, &stMcFileAccess);
if (MV_OK != nRet)
{
   printf("error: Reading from camera failed. [%x]\n", nRet);
   return;
}
//Other process...

//Close camera

nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
```

```
        return;
    }
    //Destroy handle and release resources
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.11 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

```
//Enumerate all devices of specific transport protocol in the subnet
unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}
int i = 0;
if (m_stDevList.nDeviceNum == 0)
{
   printf("no camera found!\n");
   return;
}

//Select the first found online device and create handle
int nDeviceIndex = 0;
MV_CC_DEVICE_INFO m_stDevInfo = {0};
memcpy(&m_stDevInfo, m_stDevList.pDeviceInfo[nDeviceIndex], sizeof(MV_CC_DEVICE_INFO));
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect to device
unsigned int nAccessMode = MV_ACCESS_Exclusive;
unsigned short nSwitchoverKey = 0;

nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
   printf("error: OpenDevice fail [%x]\n", nRet);
   return;
}
MV_CC_FILE_ACCESS stMcFileAccess;
stMcFileAccess.pUserFileName =  "UserSet1.txt";
stMcFileAccess.pDevFileName =  "UserSet1";
nRet = MV_CC_FileAccessWrite( m_handle, &stMcFileAccess);
if (MV_OK != nRet)
{
   printf("error: Writing to camera failed. [%x]\n", nRet);
   return;
}
//Other process...

//Close camera
```

```
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }
    //Destroy handle and release resources
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.12 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

unsigned int g_nMode = 0;
int g_nRet = MV_OK;
// Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
```

```
    {
        Sleep(10);
    }
    _getch();
}

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

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

    return true;
}

static  unsigned int __stdcall ProgressThread(void* pUser)
{
    int nRet = MV_OK;
    MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = {0};

    while(1)
    {
        //Get progress of file access
        nRet = MV_CC_GetFileAccessProgress(pUser, &stFileAccessProgress);
        printf("State = 0x%x,Completed = %I64d,Total = %I64d\r
\n",nRet,stFileAccessProgress.nCompleted,stFileAccessProgress.nTotal);
        if (nRet != MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted ==
stFileAccessProgress.nTotal))
        {
```

```
            break;
        }

        Sleep(50);
    }

    return 0;
}

static  unsigned int __stdcall FileAccessThread(void* pUser)
{
    MV_CC_FILE_ACCESS stFileAccess = {0};

    stFileAccess.pUserFileName = "UserSet1.bin";
    stFileAccess.pDevFileName = "UserSet1";
    if (1 == g_nMode)
    {
        //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)
    {
        //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 device
        MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
```

```
}

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

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

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

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

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

//Read mode
g_nMode = 1;
printf("Read to file.\n");

unsigned int nThreadID = 0;
```

```
void* hReadHandle = (void*) _beginthreadex( NULL , 0 , FileAccessThread , handle, 0 , &nThreadID );
if (NULL == hReadHandle)
{
   break;
}

Sleep(5);

nThreadID = 0;
void* hReadProgressHandle = (void*) _beginthreadex( NULL , 0 , ProgressThread , handle, 0 , &nThreadID );
if (NULL == hReadProgressHandle)
{
   break;
}

WaitForMultipleObjects(1, &hReadHandle, TRUE, INFINITE);
WaitForMultipleObjects(1, &hReadProgressHandle, TRUE, INFINITE);
if (MV_OK == g_nRet)
{
   printf("File Access Read Success!\n");
}
printf("\n");

//Write mode
g_nMode = 2;
printf("Write from file.\n");

nThreadID = 0;
void* hWriteHandle = (void*) _beginthreadex( NULL , 0 , FileAccessThread , handle, 0 , &nThreadID );
if (NULL == hWriteHandle)
{
   break;
}

Sleep(5);

nThreadID = 0;
void* hWriteProgressHandle = (void*) _beginthreadex( NULL , 0 , ProgressThread , handle, 0 , &nThreadID );
if (NULL == hWriteProgressHandle)
{
   break;
}

WaitForMultipleObjects(1, &hWriteHandle, TRUE, INFINITE);
WaitForMultipleObjects(1, &hWriteProgressHandle, TRUE, INFINITE);
if (MV_OK == g_nRet)
{
   printf("File Access Write Success!\n");
}

//Close device
nRet = MV_CC_CloseDevice(handle);
```

```
        if (MV_OK != nRet)
        {
            printf("ClosDevice fail! nRet [0x%x]\n", nRet);
            break;
        }

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

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

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

    return 0;
}
```

3.3.13 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

   //Selected the first online device in the search result, and create device handle.
   int nDeviceIndex = 0;

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

```
    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Set device IP address
    unsigned int nIP = 0x0a0f0636; //10.15.6.54
    unsigned int nSubNetMask = 0xffffff00; //255.255.255.0
    unsigned int nDefaultGateWay = 0x0a0f06fe; //10.15.6.254
    nRet = MV_GIGE_ForceIpEx(m_handle, nIP, nSubNetMask, nDefaultGateWay);
    if (MV_OK != nRet)
    {
        printf("error: ForceIpEx fail [%x]\n", nRet);
        return;
    }

    //...Other processes.

    //Destroy handle, release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.14 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

```
    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Set device IP address
    unsigned int nType = MV_IP_CFG_STATIC; //Static IP address mode
    nRet = MV_GIGE_SetIpConfig(m_handle, nType);
    if (MV_OK != nRet)
    {
        printf("error: SetIpConfig fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.15 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

```
    return;
  }

  //Connect device
  unsigned int nAccessMode = MV_ACCESS_Exclusive;
  unsigned short nSwitchoverKey = 0;

  nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
  if (MV_OK != nRet)
  {
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
  }

  //Set priority network mode
  unsigned int nType = MV_NET_TRANS_SOCKET; //Socket mode
  nRet = MV_GIGE_SetNetTransMode(m_handle, nType);
  if (MV_OK != nRet)
  {
    printf("error: SetNetTransMode fail [%x]\n", nRet);
    return;
  }

  //...other processing

  //Destroy handle and release resource
  nRet = MV_CC_DestroyHandle(m_handle);
  if (MV_OK != nRet)
  {
    printf("error: DestroyHandle fail [%x]\n", nRet);
    return;
  }
}
```

3.3.16 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.

Example

The following sample code is for reference only.

```
using namespace MvCamCtrl;

#define MAX_BUF_SIZE    (1920*1080*3)

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

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

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

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

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

   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
```

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

    else
    {
        unsigned int nAccessMode = MV_ACCESS_Exclusive;
        unsigned short nSwitchoverKey = 0;

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

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

    //Get size of one frame
     MVCC_INTVALUE stIntvalue = {0};
    nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
    if (nRet != MV_OK)
    {
        printf("Get PayloadSize failed! nRet [%x]\n", nRet);
        return;
    }
    int nBufSize = stIntvalue.nCurValue; //One frame data size

    unsigned int    nTestFrameSize = 0;
    unsigned char*  pFrameBuf = NULL;
    pFrameBuf = (unsigned char*)malloc(nBufSize);

    MV_FRAME_OUT_INFO stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO));

    MV_NETTRANS_INFO stNetTransInfo = {0};

    //The frequency of calling this interface should be controlled by upper layer application according to frame rate.
    //The codes are for reference only. In practical application, it is recommended to create a new thread for image
acquisition and processing.
    while(1)
```

```
{
    if (nTestFrameSize > 99)
    {
        break;
    }
    nRet = MV_CC_GetOneFrame(m_handle, g_pFrameBuf, MAX_BUF_SIZE, &stInfo);
    if (MV_OK != nRet)
    {
        Sleep(10);
    }
    else
    {
        //...Image data processing

        // Get network transmission information
        if ((nTestFrameSize % 10) == 0)
        {
            nRet = MV_GIGE_GetNetTransInfo(m_handle, &stNetTransInfo);
            if(MV_OK == nRet)
            {
                printf("GetDataSize [%I64d]\n", stNetTransInfo.nReviceDataSize);
            }
        }

        nTestFrameSize++;
    }
}

//...other processing

//Stop acquiring image
nRet = MV_CC_StopGrabbing(m_handle);
if (MV_OK != nRet)
{
    printf("error: StopGrabbing fail [%x]\n", nRet);
    return;
}

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
    printf("error: CloseDevice fail [%x]\n", nRet);
    return;
}

//Destroy handle and release resource
nRet = MV_CC_DestroyHandle(m_handle);
if (MV_OK != nRet)
{
    printf("error: DestroyHandle fail [%x]\n", nRet);
    return;
```

```
    }
}
```

3.3.17 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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

    //Enumerate all devices corresponding to specified transport protocol within subnet
    unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
    MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
    int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
```

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

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

//Get optimized packet size
int iPacketSize = MV_CC_GetOptimalPacketSize(m_handle);

//Detect network optimal package size (It only works for the GigE camera)
if (stDevInfo.nTLayerType == MV_GIGE_DEVICE)
{
   int nPacketSize = MV_CC_GetOptimalPacketSize(handle);
   if (nPacketSize > 0)
   {
      nRet = MV_CC_SetIntValue(handle,"GevSCPSPacketSize",nPacketSize);
      if(nRet != MV_OK)
      {
         printf("Warning: Set Packet Size fail nRet [0x%x]!", nRet);
      }
   }
```

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

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.18 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

```
    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

    //Set resend packet properties
    nRet = MV_GIGE_SetResend(m_handle, 1, 1, 100);
    if (MV_OK != nRet)
    {
        printf("error: MV_GIGE_SetResend fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.3.19 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.

3.3.20 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.

Return Value

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

3.3.21 MV_GIGE_GetResendTimeInterval

Get the packet resending interval.

API Definition

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

Parameters

handle

[IN] Device handle

pnMililsec

[OUT] Packet resending interval, range value: [0, 0xffffffff], unit: millisecond

Return Value

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

3.3.22 MV_GIGE_SetResendTimeInterval

Set the packet resending interval.

API Definition

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

Parameters

handle

[IN] Device handle

nMililsec

[IN] Packet resending interval, range value: [0, 0xffffffff], unit: millisecond

Return Value

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

3.3.23 MV_GIGE_GetGvspTimeout

Get packet resending timeout

API Definition

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

Parameters

handle

[IN] Device handle

pnMililsec

[OUT] Packet resending timeout, which is 100 ms by default, range value: [0, 0xffffffff], unit: millisecond

Return Value

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

3.3.24 MV_GIGE_SetGvspTimeout

Set packet resending timeout

API Definition

```
public native static int MV_GIGE_SetGvspTimeout(
  void              *handle,
  unsigned int          nMililsec
);
```

Parameters

handle

[IN] Device handle

nMililsec

[IN] Packet resending timeout, which is 100 ms by default, range value: [0, 0xffffffff], unit: millisecond

Return Value

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

3.3.25 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
//Wait for key input
void WaitForKeyPress(void)
{
   while(!_kbhit())
   {
      Sleep(10);
   }
   _getch();
}

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

      //Print the camera's IP address and user defined name
      printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
      printf("UserDefinedName: %s\n\n" , pstMVDevInfo->SpecialInfo.stGigEInfo.chUserDefinedName);
```

```
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n\n", pstMVDevInfo->SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}

static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * g_nPayloadSize);
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n"
             stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    free(pData);
    return 0;
}

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

//Ask for enabling multicast control or monitoring App
printf("Start multicast sample in (c)ontrol or in (m)onitor mode? (c/m)\n");
char key;
do
{
    scanf("%c", &key);
}
while ( (key != 'c') && (key != 'm') && (key != 'C') && (key != 'M'));

//Search for the adopted mode
bool monitorMode = (key == 'm') || (key == 'M');
```

```
//Open camera
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;
}

//Get data packet 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 multicast IP address
char strIp[] = "239.0.1.23";
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 multicast port number
MV_CC_TRANSMISSION_TYPE stTransmissionType;
memset(&stTransmissionType, 0, sizeof(MV_CC_TRANSMISSION_TYPE));
stTransmissionType.enTransmissionType = MV_GIGE_TRANSTYPE_MULTICAST;
stTransmissionType.nDestIp = nIp;
stTransmissionType.nDestPort = 8787;
nRet = MV_GIGE_SetTransmissionType(handle, &stTransmissionType);
if (MV_OK != nRet)
{
   printf("Set Transmission Type fail! nRet [0x%x]\n", nRet);
   break;
}

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

```
    }
    unsigned int nThreadID = 0;
    void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , handle, 0 , &nThreadID );
    if (NULL == hThreadHandle)
    {
        break;
    }
    printf("Press a key to stop grabbing.\n");
    WaitForKeyPress();
    g_bExit = true;
    Sleep(1000);

    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(handle);
    if (MV_OK != nRet)
    {
        printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
        break;
    }
    //Close camera
    nRet = MV_CC_CloseDevice(handle);
    if (MV_OK != nRet)
    {
        printf("ClosDevice fail! nRet [0x%x]\n", nRet);
        break;
    }
     //Destroy handle
    nRet = MV_CC_DestroyHandle(handle);
    if (MV_OK != nRet)
    {
        printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
        break;
    }
    while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
            MV_CC_DestroyHandle(handle);
            handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

3.3.26 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.

Example

The following sample code is for reference only.

```
#include <stdio.h>

#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"
bool g_bExit = false;
bool g_bReceive = false;
unsigned int g_nPayloadSize = 0;
unsigned int g_DeviceKey = 1;
unsigned int g_GroupKey = 1;
unsigned int g_GroupMask= 1;

//Wait for key input
void WaitForKeyPress(void)
{
   while(!_kbhit())
   {
      Sleep(10);
   }
   _getch();
}
static  unsigned int __stdcall ActionCommandWorkThread(void* pUser)
{
   int nRet = MV_OK;
   MV_ACTION_CMD_INFO stActionCmdInfo = {0};
```

```
    MV_ACTION_CMD_RESULT_LIST stActionCmdResults = {0};
    stActionCmdInfo.nDeviceKey = g_DeviceKey;
    stActionCmdInfo.nGroupKey = g_GroupKey;
    stActionCmdInfo.nGroupMask = g_GroupMask;
    stActionCmdInfo.pBroadcastAddress = "255.255.255.255";
    stActionCmdInfo.nTimeOut = 100;
    stActionCmdInfo.bActionTimeEnable = 0;
    while(!g_bExit)
    {
        nRet = MV_GIGE_IssueActionCommand(&stActionCmdInfo,&stActionCmdResults);
        if (MV_OK != nRet)
        {
            printf("Issue Action Command fail! nRet [0x%x]\n", nRet);
        }
        printf("NumResults = %d\r\n",stActionCmdResults.nNumResults);
        MV_ACTION_CMD_RESULT* pResults = stActionCmdResults.pResults;
        for (unsigned int i = 0;i < stActionCmdResults.nNumResults;i++)
        {
            printf("Ip == %s, Status == 0x%x\r\n",pResults->strDeviceAddress,pResults->nStatus);
            pResults++;
        }
    }
    return 0;
}

static  unsigned int __stdcall ReceiveImageWorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * (g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            g_bExit = true;
            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 device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

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

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

        //Get optimal network packet size (It works only for GigE camera)
        if (stDeviceList.pDeviceInfo[0]->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 Action Device Key
nRet = MV_CC_SetIntValue(handle, "ActionDeviceKey", g_DeviceKey);
if (MV_OK != nRet)
{
    printf("Set Action Device Key fail! nRet [0x%x]\n", nRet);
    break;
}

//Set Action Group Key
nRet = MV_CC_SetIntValue(handle, "ActionGroupKey", g_GroupKey);
if (MV_OK != nRet)
{
    printf("Set Action Group Key fail! nRet [0x%x]\n", nRet);
    break;
}

//Set Action Group Mask
nRet = MV_CC_SetIntValue(handle, "ActionGroupMask", g_GroupMask);
if (MV_OK != nRet)
{
    printf("Set Action Group Mask fail! nRet [0x%x]\n", nRet);
    break;
}

//Get packet 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 acquisition
  nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
    printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
    break;
}
unsigned int nReceiveImageThreadID = 0;
```

```
    void* hReceiveImageThreadHandle = (void*) _beginthreadex( NULL , 0 , ReceiveImageWorkThread , handle, 0 ,
&nReceiveImageThreadID );
    if (NULL == hReceiveImageThreadHandle)
    {
      break;
    }
    unsigned int nActionCommandThreadID = 0;
    void* hActionCommandThreadHandle = (void*) _beginthreadex( NULL , 0 , ActionCommandWorkThread , NULL,
0 , &nActionCommandThreadID );
    if (NULL == hActionCommandThreadHandle)
    {
      return 0;
    }
    printf("Press a key to stop grabbing.\n");
    WaitForKeyPress();
    g_bExit = true;
    Sleep(1000);

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

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

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

  if (nRet != MV_OK)
  {
    if (handle != NULL)
    {
      MV_CC_DestroyHandle(handle);
      handle = NULL;
    }
  }
  printf("Press a key to exit.\n");
```

```
    WaitForKeyPress();
    return 0;
}
```

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 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.

See Also

MV_CC_StartGrabbing

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO_EX* pFrameInfo, void* pUser)
{
   if (pFrameInfo)
   {
      // Add current system time at output
      char   szInfo[128] = {0};
      SYSTEMTIME sys;
      GetLocalTime( &sys );
      sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] : GetOneFrame succeed, width[%d], height[%d]",
sys.wYear, sys.wMonth,
         sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds, pFrameInfo->nWidth, pFrameInfo-
>nHeight);

      printf("%s\n", szInfo);
   }
}

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

   //Enumerate all devices corresponding to specified transport protocol within subnet
   unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
   MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
```

```
int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Register data callback function
nRet = MV_CC_RegisterImageCallBackEx(handle, ImageCallBack, NULL);
if (MV_OK != nRet)
{
   printf("error: RegisterImageCallBack fail [%x]\n", nRet);
   return;
}

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

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

```
    //...other processing


    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO_EX* pFrameInfo, void* pUser)
{
   if (pFrameInfo)
```

```
    {
        // Add the system time when output.
        char   szInfo[128] = {0};
        SYSTEMTIME sys;
        GetLocalTime( &sys );
        sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] : GetOneFrame succeed, width[%d], height[%d]",
sys.wYear, sys.wMonth,
            sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds, pFrameInfo->nWidth, pFrameInfo-
>nHeight);

        printf("%s\n", szInfo);
    }
}

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

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

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

    //Selected the first online device in the search result, and create device handle.
    int nDeviceIndex = 0;

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Register RGB data callback function
    nRet = MV_CC_RegisterImageCallBackForRGB(handle, ImageCallBack, NULL);
```

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

    //Connect to device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...Other processes.

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

    //...Other processes.


    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close device, release resource.
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle, release resource.
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO_EX* pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        // Add the system time when output.
        char   szInfo[128] = {0};
        SYSTEMTIME sys;
        GetLocalTime( &sys );
        sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] : GetOneFrame succeed, width[%d], height[%d]",
sys.wYear, sys.wMonth,
            sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds, pFrameInfo->nWidth, pFrameInfo-
>nHeight);

        printf("%s\n", szInfo);
    }
}

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

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

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

//Selected the first online device in the search result, and create device handle.
int nDeviceIndex = 0;

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Register BGR data callback function
nRet = MV_CC_RegisterImageCallBackForBGR(handle, ImageCallBack, NULL);
if (MV_OK != nRet)
{
    printf("error: RegisterImageCallBackForBGR fail [%x]\n", nRet);
    return;
}

//Connect to device
nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
}
//...Other processes.

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

//...Other processes.


//Stop image acquisition
nRet = MV_CC_StopGrabbing(m_handle);
if (MV_OK != nRet)
```

```
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close device, release resource.
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle, release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

void __stdcall ImageCallBack(unsigned char * pData, MV_FRAME_OUT_INFO* pFrameInfo, void* pUser)
{
    if (pFrameInfo)
    {
        // Add system time at output
        char   szInfo[128] = {0};
        SYSTEMTIME sys;
        GetLocalTime( &sys );
        sprintf_s(szInfo, 128, "[%d-%02d-%02d %02d:%02d:%02d:%04d] : GetOneFrame succeed, width[%d], height[%d]",
sys.wYear, sys.wMonth,
            sys.wDay, sys.wHour, sys.wMinute, sys.wSecond, sys.wMilliseconds, pFrameInfo->nWidth, pFrameInfo-
>nHeight);

        printf("%s\n", szInfo);
    }
}

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

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

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

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

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);
```

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

else
{
   unsigned int nAccessMode = MV_ACCESS_Exclusive;
   unsigned short nSwitchoverKey = 0;

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

//Register data callback function, get acquired image data in the callback function
nRet = MV_CC_RegisterImageCallBack(handle, ImageCallBack, NULL);
if (MV_OK != nRet)
{
   printf("error: RegisterImageCallBack fail [%x]\n", nRet);
   return;
}

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

//...other processing

//Stop acquiring image
nRet = MV_CC_StopGrabbing(m_handle);
if (MV_OK != nRet)
{
   OutputString(_T("Operation: Stop Grabbing.....failed!"));
   return;
}

//Shut device and release resource
nRet = MV_CC_CloseDevice(m_handle);
if (MV_OK != nRet)
{
   printf("error: CloseDevice fail [%x]\n", nRet);
```

```
    return;
  }

  //Destroy handle and release resource
  nRet = MV_CC_DestroyHandle(m_handle);
  if (MV_OK != nRet)
  {
    printf("error: DestroyHandle fail [%x]\n", nRet);
    return;
  }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

  //Enumerate all devices corresponding to specified transport protocol within subnet
  unsigned int nTLayerType = MV_GIGE_DEVICE | MV_USB_DEVICE;
```

```
MV_CC_DEVICE_INFO_LIST m_stDevList = {0};
int nRet = MV_CC_EnumDevices(nTLayerType, &m_stDevList);
if (MV_OK != nRet)
{
   printf("error: EnumDevices fail [%x]\n", nRet);
   return;
}

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

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

else
{
   unsigned int nAccessMode = MV_ACCESS_Exclusive;
   unsigned short nSwitchoverKey = 0;

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

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

```
    //...other processing

    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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 cache 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

```
    return;
}

//Selected the first online device in the search result, and create device handle.
int nDeviceIndex = 0;

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect to device
nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
}
//...Other processes

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

//Get the size of one frame data
MVCC_INTVALUE stIntvalue = {0};
nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
if (nRet != MV_OK)
{
    printf("Get PayloadSize failed! nRet [%x]\n", nRet);
    return;
}
int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size + reserved bytes (handled in SDK)

unsigned int    nTestFrameSize = 0;
unsigned char*  pFrameBuf = NULL;
pFrameBuf = (unsigned char*)malloc(nBufSize);

MV_FRAME_OUT_INFO_EX stInfo;
memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));

//The upper layer program should control the frequency of calling  this API according to the frame rate.
```

```
    //The codes is for reference only, please establish another thread to execute image acquisition and handling in
actual application.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetImageForRGB(m_handle,pFrameBuf, nBufSize, &stInfo, 1000);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //...Processing image data
            nTestFrameSize++;
        }
    }

    //...Other processes

    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close device, release resource.
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle, release resource.
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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 cache 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

    //Selected the first online device in the search result, and create device handle.
    int nDeviceIndex = 0;

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect to device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...Other processes

    //Start image acquisition
    nRet = MV_CC_StartGrabbing(m_handle);
```

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

    //Get the size of one frame data
    MVCC_INTVALUE stIntvalue = {0};
    nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
    if (nRet != MV_OK)
    {
        printf("Get PayloadSize failed! nRet [%x]\n", nRet);
        return;
    }
    int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size + reserved bytes (handled in SDK)

    unsigned int    nTestFrameSize = 0;
    unsigned char*  pFrameBuf = NULL;
    pFrameBuf = (unsigned char*)malloc(nBufSize);

    MV_FRAME_OUT_INFO_EX stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));

    //The upper layer program should control the frequency of calling  this API according to the frame rate.
    //The codes is for reference only, please establish another thread to execute image acquisition and handling in
actual application.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetImageForBGR(m_handle,pFrameBuf, nBufSize, &stInfo, 1000);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //...Processing image data
            nTestFrameSize++;
        }
    }

    //...Other processes

    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
```

```
    }

    //Close device, release resource.
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle, release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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 cache 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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

```
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

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

MV_FRAME_OUT stOutFrame;
memset(&stOutFrame, 0, sizeof(MV_FRAME_OUT));

//The upper layer should control the frequency of calling this API according to frame rate.
 //The codes are for reference only, it is recommended to create a new thread for image acquisition and processing
 while(1)
{
   if (nTestFrameSize > 99)
   {
      break;
   }
   nRet = MV_CC_GetImageBuffer(m_handle, &stOutFrame, 1000);
   if (MV_OK != nRet)
   {
      Sleep(10);
   }
   else
   {
      //...Process image data
      nTestFrameSize++;
   }
   nRet = MV_CC_FreeImageBuffer(m_handle, &stOutFrame);
   if(nRet != MV_OK)       {          printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
   }
}

//...other processing
```

```
    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: StopGrabbing fail [%x]\n", nRet);
      return;
   }

   //Close device, and release resource
    nRet = MV_CC_CloseDevice(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: CloseDevice fail [%x]\n", nRet);
      return;
   }

   //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: DestroyHandle fail [%x]\n", nRet);
      return;
   }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

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

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

   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

   if (MV_OK != nRet)
```

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

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

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

MV_FRAME_OUT stOutFrame;
memset(&stOutFrame, 0, sizeof(MV_FRAME_OUT));

//The upper layer should control the frequency of calling this API according to frame rate.
 //The codes are for reference only, it is recommended to create a new thread for image acquisition and processing
 while(1)
{
    if (nTestFrameSize > 99)
    {
        break;
    }
    nRet = MV_CC_GetImageBuffer(m_handle, &stOutFrame, 1000);
    if (MV_OK != nRet)
    {
        Sleep(10);
    }
    else
    {
        //...Process image data
        nTestFrameSize++;
    }
    nRet = MV_CC_FreeImageBuffer(m_handle, &stOutFrame);
    if(nRet != MV_OK)       {           printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
    }
}

 //...other processing

 //Stop image acquisition
 nRet = MV_CC_StopGrabbing(m_handle);
```

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

    //Close device, and release resource
     nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
     nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

    //Selected the first online device in the search result, and create device handle.
    int nDeviceIndex = 0;

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

```
nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect to device
nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
}
//...Other processes

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

//Get the size of one frame data
MVCC_INTVALUE stIntvalue = {0};
nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
if (nRet != MV_OK)
{
    printf("Get PayloadSize failed! nRet [%x]\n", nRet);
    return;
}
int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size + reserved bytes (handled in SDK)

unsigned int    nTestFrameSize = 0;
unsigned char*  pFrameBuf = NULL;
pFrameBuf = (unsigned char*)malloc(nBufSize);

MV_FRAME_OUT_INFO_EX stInfo;
memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));

//The upper layer program should control the frequency of calling  this API according to the frame rate.
//The codes is for reference only, please establish another thread to execute image acquisition and handling in
actual application.
while(1)
{
    if (nTestFrameSize > 99)
    {
        break;
    }
```

```
        nRet = MV_CC_GetOneFrameTimeout(m_handle, pFrameBuf, nBufSize, &stInfo, 1000);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //...Processing image data
            nTestFrameSize++;
        }
    }

    //...Other processes

    //Stop image acquisition
    nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close device, release resource.
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle, release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

   //Select the first found online device and create device handle
```

```
int nDeviceIndex = 0;

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

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

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

MV_FRAME_OUT stOutFrame;
memset(&stOutFrame, 0, sizeof(MV_FRAME_OUT));
MV_DISPLAY_FRAME_INFO stDisplayOneFrame;
memset(&stDisplayOneFrame, 0, sizeof(MV_DISPLAY_FRAME_INFO));

//The frequency of calling this API should be controlled by upper layer application according to frame rate
 //The codes are for reference only. In practical application, it is recommended to create a new thread for image
acquisition and processing.
 while(1)
{
   if (nTestFrameSize > 99)
   {
      break;
   }
   nRet = MV_CC_GetImageBuffer(m_handle, &stOutFrame, 1000);
   if (MV_OK != nRet)
   {
      Sleep(10);
   }
   else
   {
      //...Process image data
```

```
        nTestFrameSize++;
      }

      stDisplayOneFrame.hWnd = hWnd;      stDisplayOneFrame.pData = stFrameOut.pData;
stDisplayOneFrame.nDataLen = stFrameOut.stFrameInfo.nFrameLen;      stDisplayOneFrame.nWidth =
stFrameOut.stFrameInfo.nWidth;      stDisplayOneFrame.nHeight = stFrameOut.stFrameInfo.nHeight;
stDisplayOneFrame.enPixelType = stFrameOut.stFrameInfo.enPixelType;
       nRet = MV_CC_DisplayOneFrame(m_handle, &stDisplayOneFrame);
      if(nRet != MV_OK)      {          printf("Display one frame fail! nRet [0x%x]\n", nRet);
      }

      nRet = MV_CC_FreeImageBuffer(m_handle, &stOutFrame);
      if(nRet != MV_OK)      {          printf("Free Image Buffer fail! nRet [0x%x]\n", nRet);
      }
   }

   //...other processing

    //Stop acquiring image
    nRet = MV_CC_StopGrabbing(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: StopGrabbing fail [%x]\n", nRet);
      return;
   }

   //Shut down device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: CloseDevice fail [%x]\n", nRet);
      return;
   }

   //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
   if (MV_OK != nRet)
   {
      printf("error: DestroyHandle fail [%x]\n", nRet);
      return;
   }
}
```

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.

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"
void main()
{
   int nRet = -1;
   void* m_handle = NULL;

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

   int i = 0;
   if (m_stDevList.nDeviceNum == 0)
```

```
{
    printf("no camera found!\n");
    return;
}

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

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

nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

//Connect to device
 nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
if (MV_OK != nRet)
{
    printf("error: OpenDevice fail [%x]\n", nRet);
    return;
}
//Other process...

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

//Get the size of one data frame
 MVCC_INTVALUE stIntvalue = {0};
nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
if (nRet != MV_OK)
{
    printf("Get PayloadSize failed! nRet [%x]\n", nRet);
    return;
}
int nBufSize = stIntvalue.nCurValue + 2048; //One data frame size + reserved bytes

unsigned int    nTestFrameSize = 0;
unsigned char*  pFrameBuf = NULL;
pFrameBuf = (unsigned char*)malloc(nBufSize);

MV_FRAME_OUT_INFO_EX stInfo;
memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
```

```
  //The upper-layer program should control the frequency of calling this API according to the frame rate
   //For actual application, you'd better create another thread for collecting and processing picture frame
   while(1)
  {
     if (nTestFrameSize > 99)
     {
        break;
     }
     nRet = MV_CC_GetOneFrameTimeout(m_handle,pFrameBuf, nBufSize, &stInfo, 1000);
     if (MV_OK != nRet)
     {
        Sleep(10);
     }
     else
     {
        //Input and output parameters of picture data
        MV_SAVE_IMAGE_PARAM_EX stParam;

        //Source data
        stParam.pData        = pFrameBuf;              //Original image data
        stParam.nDataLen     = stFrameInfo.nFrameLen;    //Size of original image data
        stParam.enPixelType   = stFrameInfo.enPixelType;  //Pixel format of original image data
        stParam.nWidth       = stFrameInfo.nWidth;      //Image width
        stParam.nHeight      = stFrameInfo.nHeight;     //Image height
        stParam.nJpgQuality   = 70;                     //JPEG encoding quality

        //Target data
        stParam.enImageType   = MV_Image_Jpeg;          //Image type to be saved, convert to JPEG format
        stParam.nBufferSize   = nBufSize;               //Storage node size
        unsigned char* pImage = (unsigned char*)malloc(nBufSize);
       stParam.pImageBuffer  = pImage;                //Buffer for output data, this buffer is also used for storing
converted picture data

        nRet = MV_CC_SaveImageEx2(m_handle, &stParam);
        if(MV_OK != nRet)
        {
           break;
        }

        //Save the converted picture data as file
         FILE* fp = fopen("image", "wb");
        fwrite(pImage, 1, stParam.nImageLen, fp);
        fclose(fp);
        free(pImage);

        //Process other image data...
        nTestFrameSize++;
     }
  }
  free(pFrameBuf);
```

```
    //Other process...

    //Stop image acquisition
     nRet = MV_CC_StopGrabbing(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: StopGrabbing fail [%x]\n", nRet);
        return;
    }

    //Close camera
     nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resources
     nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

3.5.3 MV_CC_ConvertPixelType

Convert pixel format.

API Definition

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

Parameters

handle

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

pstCvtParam

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

Return Value

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

Remarks

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

   //Selected the first online device in the search result, and create device handle.
   int nDeviceIndex = 0;

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

   nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

```
    //Connect to device
    nRet = MV_CC_OpenDevice(m_handle, nAccessMode, nSwitchoverKey);
    if (MV_OK != nRet)
    {
        printf("error: OpenDevice fail [%x]\n", nRet);
        return;
    }
    //...Other processes

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

    //Get the size of a frame data
    MVCC_INTVALUE stIntvalue = {0};
    nRet = MV_CC_GetIntValue(m_handle, "PayloadSize", &stIntvalue);
    if (nRet != MV_OK)
    {
        printf("Get PayloadSize failed! nRet [%x]\n", nRet);
        return;
    }
    int nBufSize = stIntvalue.nCurValue + 2048; //One frame data size + reserved bytes (handled in SDK)

    unsigned int    nTestFrameSize = 0;
    unsigned char*  pFrameBuf = NULL;
    pFrameBuf = (unsigned char*)malloc(nBufSize);

    MV_FRAME_OUT_INFO stInfo;
    memset(&stInfo, 0, sizeof(MV_FRAME_OUT_INFO));

    //The upper layer program should control the frequency of calling  this API according to the frame rate.
    //The codes is for reference only, please establish another thread to execute image acquisition and handling in
actual application.
    while(1)
    {
        if (nTestFrameSize > 99)
        {
            break;
        }
        nRet = MV_CC_GetOneFrameTimeout(m_handle, pFrameBuf, nBufSize, &stInfo, 1000);
        if (MV_OK != nRet)
        {
            Sleep(10);
        }
        else
        {
            //Transform input and output parameter to pixel format.
            MV_CC_PIXEL_CONVERT_PARAM stParam;
```

```
        memset(&stParam, 0, sizeof(MV_CC_PIXEL_CONVERT_PARAM));

        //Source data
        stParam.pSrcData      = pFrameBuf;            //Original image data
        stParam.nSrcDataLen   = stInfo.nFrameLen;     //Length of original image data
        stParam.enSrcPixelType = stInfo.enPixelType;  //Pixel format of original image
        stParam.nWidth        = stInfo.nWidth;        //Image width
        stParam.nHeight       = stInfo.nHeight;       //Image height

        //Target data
        stParam.enDstPixelType = PixelType_Gvsp_Mono8;    //Pixel format type needs to be saved, it will transform to
MONO8 format
        stParam.nDstBufferSize = nBufSize;          //Size of storage node
        unsigned char* pImage  = (unsigned char*)malloc(nBufSize);
        stParam.pDstBuffer     = pImage;            //Buffer for the output data,used to save the transformed data.

        nRet = MV_CC_ConvertPixelType(m_handle, &stParam);
        if(MV_OK != nRet)
        {
           break;
        }

        //...Processing other image data
        nTestFrameSize++;
        free(pImage);
     }
  }
  free(pFrameBuf);

  //...Other processes

  //Stop image acquisition
  nRet = MV_CC_StopGrabbing(m_handle);
  if (MV_OK != nRet)
  {
     printf("error: StopGrabbing fail [%x]\n", nRet);
     return;
  }

  //Close device, and release resource
  nRet = MV_CC_CloseDevice(m_handle);
  if (MV_OK != nRet)
  {
     printf("error: CloseDevice fail [%x]\n", nRet);
     return;
  }

  //Destroy handle and release resource
  nRet = MV_CC_DestroyHandle(m_handle);
  if (MV_OK != nRet)
  {
     printf("error: DestroyHandle fail [%x]\n", nRet);
```

```
        return;
    }
}
```

3.5.4 MV_CC_SetBayerCvtQuality

Set the interpolation method of Bayer format.

API Definition

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

Parameters

handle

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

nBayerCvtQuality

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

Return Value

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

Remarks

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

//Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
bool PrintDeviceInfo(MV_CC_DEVICE_INFO* pstMVDevInfo)
{
    if (NULL == pstMVDevInfo)
    {
```

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

        //Print the camera's IP address and user defined name
        printf("CurrentIp: %d.%d.%d.%d\n" , nIp1, nIp2, nIp3, nIp4);
        printf("UserDefinedName: %s\n\n" , pstMVDevInfo->SpecialInfo.stGigEInfo.chUserDefinedName);
    }
    else if (pstMVDevInfo->nTLayerType == MV_USB_DEVICE)
    {
        printf("UserDefinedName: %s\n\n", pstMVDevInfo->SpecialInfo.stUsb3VInfo.chUserDefinedName);
    }
    else
    {
        printf("Not support.\n");
    }
    return true;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {

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

//Open device
nRet = MV_CC_OpenDevice(handle);
if (MV_OK != nRet)
{
    printf("Open Device fail! nRet [0x%x]\n", nRet);
    break;
}
nRet = MV_CC_SetBayerCvtQuality(handle,0);
if (MV_OK != nRet)
{
    printf("Set BayerCvt Quality fail! nRet [0x%x]\n", nRet);
    break;
}
//Set the triggering mode as 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 data packet 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;
}
```

```
unsigned int nPayloadSize = stParam.nCurValue;
MV_FRAME_OUT_INFO_EX stImageInfo = {0};
memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
MV_BUFFER_INFO stRevBufInfo = {0};
stRevBufInfo.pBuffer = (unsigned char *)malloc(sizeof(unsigned char) * nPayloadSize);
if (stRevBufInfo.pBuffer == NULL)
{
   printf("malloc register buffer memory fail !\n");
   break;
}
stRevBufInfo.nBufSize = nPayloadSize;

//Start image acquisition
nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
   printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
   break;
}
int nCount = 0;
while (nCount++ != 100)
{
  nRet = MV_CC_GetOneFrameTimeout(handle, stRevBufInfo.pBuffer, nPayloadSize, &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);
  }
}
//Stop image acquisition
nRet = MV_CC_StopGrabbing(handle);
if (MV_OK != nRet)
{
   printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
   break;
}
 //Close device
 nRet = MV_CC_CloseDevice(handle);
 if (MV_OK != nRet)
 {
    printf("ClosDevice fail! nRet [0x%x]\n", nRet);
    break;
 }
 //Destroy handle
 nRet = MV_CC_DestroyHandle(handle);
 if (MV_OK != nRet)
 {
    printf("Destroy Handle fail! nRet [0x%x]\n", nRet);
```

```
        break;
    }
    while (0);
    if (nRet != MV_OK)
    {
        if (handle != NULL)
        {
          MV_CC_DestroyHandle(handle);
           handle = NULL;
        }
    }
    printf("Press a key to exit.\n");
    WaitForKeyPress();
    return 0;
}
```

3.5.5 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.

Example

The following sample code is for reference only.

```
#include <stdio.h>

#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;
```

```
//Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * (g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    MV_CC_INPUT_FRAME_INFO stInputFrameInfo = {0};
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n"
            stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
            stInputFrameInfo.pData = pData;
            stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
            nRet = MV_CC_InputOneFrame(pUser, &stInputFrameInfo);
            if (MV_OK != nRet)
            {
                printf("Input one frame fail! nRet [0x%x]\n", nRet);
            }
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    free(pData);
    return 0;
}
int main()
{
    int nRet = MV_OK;
```

```
void* handle = NULL;
do
{

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

    // Select device and create handle
      nRet = MV_CC_CreateHandle(&handle, stDeviceList.pDeviceInfo[0]);
    if (MV_OK != nRet)
    {
        printf("Create Handle fail! nRet [0x%x]\n", nRet);
        break;
    }
    if (m_stDevList.nDeviceNum == 0)
    {
        printf("no camera found!\n");
        return 0;
    }

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

    // Detect the optimal network packet size (It works only for GigE camera)
    if (stDeviceList.pDeviceInfo[0]->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 triggering 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 packet 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;
MV_CC_RECORD_PARAM stRecordPar = {0};
memset(&stParam, 0, sizeof(MVCC_INTVALUE));
nRet = MV_CC_GetIntValue(handle, "Width", &stParam);
if (MV_OK != nRet)
{
   printf("Get Width fail! nRet [0x%x]\n", nRet);
   break;
}
stRecordPar.nWidth = stParam.nCurValue;
nRet = MV_CC_GetIntValue(handle, "Height", &stParam);
if (MV_OK != nRet)
{
   printf("Get Height fail! nRet [0x%x]\n", nRet);
   break;
}
stRecordPar.nHeight = stParam.nCurValue;
MVCC_ENUMVALUE stEnumValue;
nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &stEnumValue);
if (MV_OK != nRet)
{
   printf("Get Width fail! nRet [0x%x]\n", nRet);
   break;
}
stRecordPar.enPixelType = MvGvspPixelType(stEnumValue.nCurValue);
MVCC_FLOATVALUE stFloatValue;
nRet = MV_CC_GetFloatValue(handle, "AcquisitionFrameRate", &stFloatValue);
if (MV_OK != nRet)
{
   printf("Get Float value fail! nRet [0x%x]\n", nRet);
   break;
}

// Frame rate (1/16-120)fps
```

```
stRecordPar.fFrameRate = stFloatValue.fCurValue;

// Bit rate kbps(128kbps-16Mbps)
stRecordPar.nBitRate = 1000;

// Video format (supports only AVI format)
stRecordPar.enRecordFmtType = MV_FormatType_AVI;
stRecordPar.strFilePath= "./Recording.avi";
nRet = MV_CC_StartRecord(handle,&stRecordPar);
if (MV_OK != nRet)
{
   printf("Start Record fail! nRet [0x%x]\n", nRet);
   break;
}

// Start acquisition
  nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
   printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
   break;
}
unsigned int nThreadID = 0;
void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , handle, 0 , &nThreadID );
if (NULL == hThreadHandle)
{
   break;
}
printf("Press a key to stop grabbing.\n");
WaitForKeyPress();
g_bExit = true;
Sleep(1000);

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

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

```
        break;
    }

    // Destroy handle and release resource
     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;
}
```

3.5.6 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

Example

The following sample code is for reference only.

```
#include <stdio.h>

#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"
bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

// Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * (g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    MV_CC_INPUT_FRAME_INFO stInputFrameInfo = {0};
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n"
            stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
            stInputFrameInfo.pData = pData;
            stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
            nRet = MV_CC_InputOneFrame(pUser, &stInputFrameInfo);
            if (MV_OK != nRet)
            {
                printf("Input one frame fail! nRet [0x%x]\n", nRet);
            }
        }
```

```
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    free(pData);
    return 0;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {
        // Enumerate device
         MV_CC_DEVICE_INFO_LIST stDeviceList;
        memset(&stDeviceList, 0, sizeof(MV_CC_DEVICE_INFO_LIST));
        nRet = MV_CC_EnumDevices(MV_GIGE_DEVICE | MV_USB_DEVICE, &stDeviceList);
        if (MV_OK != nRet)
        {
            printf("Enum Devices fail! nRet [0x%x]\n", nRet);
            break;
        }
        if (m_stDevList.nDeviceNum == 0)
        {
            printf("no camera found!\n");
            return 0;
        }

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

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

        // Detect the optimal network packet size (It works only for GigE camera)
        if (stDeviceList.pDeviceInfo[0]->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 triggering 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 packet 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;
MV_CC_RECORD_PARAM stRecordPar = {0};
memset(&stParam, 0, sizeof(MVCC_INTVALUE));
nRet = MV_CC_GetIntValue(handle, "Width", &stParam);
if (MV_OK != nRet)
{
    printf("Get Width fail! nRet [0x%x]\n", nRet);
    break;
}
stRecordPar.nHeight = stParam.nCurValue;
MVCC_ENUMVALUE stEnumValue;
nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &stEnumValue);
if (MV_OK != nRet)
{
    printf("Get Width fail! nRet [0x%x]\n", nRet);
    break;
}
stRecordPar.enPixelType = MvGvspPixelType(stEnumValue.nCurValue);
MVCC_FLOATVALUE stFloatValue;
nRet = MV_CC_GetFloatValue(handle, "AcquisitionFrameRate", &stFloatValue);
```

```
if (MV_OK != nRet)
{
   printf("Get Float value fail! nRet [0x%x]\n", nRet);
   break;
}

// Frame rate (1/16-120)fps
stRecordPar.fFrameRate = stFloatValue.fCurValue;

// Bit rate kbps(128kbps-16Mbps)
stRecordPar.nBitRate = 1000;

// Video format (supports only AVI format)
stRecordPar.enRecordFmtType = MV_FormatType_AVI;
stRecordPar.strFilePath= "./Recording.avi";
nRet = MV_CC_StartRecord(handle,&stRecordPar);
if (MV_OK != nRet)
{
   printf("Start Record fail! nRet [0x%x]\n", nRet);
   break;
}

// Start acquisition
  nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
   printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
   break;
}
unsigned int nThreadID = 0;
void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , handle, 0 , &nThreadID );
if (NULL == hThreadHandle)
{
   break;
}
printf("Press a key to stop grabbing.\n");
WaitForKeyPress();
g_bExit = true;
Sleep(1000);

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

```
    }

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

    // Destroy handle and release resource
      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;
}
```

3.5.7 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

Example

The following sample code is for reference only.

```
#include <stdio.h>

#include <process.h>
#include <conio.h>
#include "MvCameraControl.h"

bool g_bExit = false;
unsigned int g_nPayloadSize = 0;

// Wait for key input
void WaitForKeyPress(void)
{
    while(!_kbhit())
    {
        Sleep(10);
    }
    _getch();
}
static  unsigned int __stdcall WorkThread(void* pUser)
{
    int nRet = MV_OK;
    MV_FRAME_OUT_INFO_EX stImageInfo = {0};
    memset(&stImageInfo, 0, sizeof(MV_FRAME_OUT_INFO_EX));
    unsigned char * pData = (unsigned char *)malloc(sizeof(unsigned char) * (g_nPayloadSize));
    if (pData == NULL)
    {
        return 0;
    }
    unsigned int nDataSize = g_nPayloadSize;
    MV_CC_INPUT_FRAME_INFO stInputFrameInfo = {0};
    while(1)
    {
        nRet = MV_CC_GetOneFrameTimeout(pUser, pData, nDataSize, &stImageInfo, 1000);
        if (nRet == MV_OK)
        {
            printf("Get One Frame: Width[%d], Height[%d], nFrameNum[%d]\n"
            stImageInfo.nWidth, stImageInfo.nHeight, stImageInfo.nFrameNum);
            stInputFrameInfo.pData = pData;
            stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
            nRet = MV_CC_InputOneFrame(pUser, &stInputFrameInfo);
            if (MV_OK != nRet)
            {
                printf("Input one frame fail! nRet [0x%x]\n", nRet);
            }
```

```
        }
        else
        {
            printf("No data[0x%x]\n", nRet);
        }
        if(g_bExit)
        {
            break;
        }
    }
    free(pData);
    return 0;
}
int main()
{
    int nRet = MV_OK;
    void* handle = NULL;
    do
    {

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

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

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

        // Detect the optimal network packet size (It works only for GigE camera)
```

```
if (stDeviceList.pDeviceInfo[0]->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 triggering 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 packet 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;
MV_CC_RECORD_PARAM stRecordPar = {0};
memset(&stParam, 0, sizeof(MVCC_INTVALUE));
nRet = MV_CC_GetIntValue(handle, "Width", &stParam);
if (MV_OK != nRet)
{
    printf("Get Width fail! nRet [0x%x]\n", nRet);
    break;
}
stRecordPar.nWidth = stParam.nCurValue;
nRet = MV_CC_GetIntValue(handle, "Height", &stParam);
if (MV_OK != nRet)
{
    printf("Get Height fail! nRet [0x%x]\n", nRet);
    break;
}
stRecordPar.nHeight = stParam.nCurValue;
MVCC_ENUMVALUE stEnumValue;
```

```
nRet = MV_CC_GetEnumValue(handle, "PixelFormat", &stEnumValue);
if (MV_OK != nRet)
{
   printf("Get Width fail! nRet [0x%x]\n", nRet);
   break;
}
stRecordPar.enPixelType = MvGvspPixelType(stEnumValue.nCurValue);
MVCC_FLOATVALUE stFloatValue;
nRet = MV_CC_GetFloatValue(handle, "AcquisitionFrameRate", &stFloatValue);
if (MV_OK != nRet)
{
   printf("Get Float value fail! nRet [0x%x]\n", nRet);
   break;
}

// Frame rate (1/16-120)fps
stRecordPar.fFrameRate = stFloatValue.fCurValue;

// Bit rate kbps(128kbps-16Mbps)
stRecordPar.nBitRate = 1000;

// Video format (supports only AVI format)
stRecordPar.enRecordFmtType = MV_FormatType_AVI;
stRecordPar.strFilePath= "./Recording.avi";
nRet = MV_CC_StartRecord(handle,&stRecordPar);
if (MV_OK != nRet)
{
   printf("Start Record fail! nRet [0x%x]\n", nRet);
   break;
}

// Start acquisition
  nRet = MV_CC_StartGrabbing(handle);
if (MV_OK != nRet)
{
   printf("Start Grabbing fail! nRet [0x%x]\n", nRet);
   break;
}
unsigned int nThreadID = 0;
void* hThreadHandle = (void*) _beginthreadex( NULL , 0 , WorkThread , handle, 0 , &nThreadID );
if (NULL == hThreadHandle)
{
   break;
}
printf("Press a key to stop grabbing.\n");
WaitForKeyPress();
g_bExit = true;
Sleep(1000);

// Stop acquisition
  nRet = MV_CC_StopGrabbing(handle);
if (MV_OK != nRet)
```

```
        {
            printf("Stop Grabbing fail! nRet [0x%x]\n", nRet);
            break;
        }
        nRet = MV_CC_StopRecord(handle);
        if (MV_OK != nRet)
        {
            printf("Stop record fail! nRet [0x%x]\n", nRet);
            break;
        }

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

        // Destroy handle and release resource
          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;
}
```

3.6 Advanced Settings

3.6.1 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, unit: ms, range: [0,10000].

Return Value

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

Remarks

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

Example

The following sample code is for reference only.

```
#include "MvCameraControl.h"

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

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

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

    //Select the first found online device and create device handle
```

```
    int nDeviceIndex = 0;

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

    nRet = MV_CC_CreateHandle(&m_handle, &m_stDevInfo);

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

    //Connect device
    unsigned int nAccessMode = MV_ACCESS_Exclusive;
    unsigned short nSwitchoverKey = 0;

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

    //Set GVCP command timeout time
    unsigned int nMillisec = 100; //ms
    nRet = MV_GIGE_SetGvcpTimeout(m_handle, nMillisec);
    if (MV_OK != nRet)
    {
        printf("error: SetGvcpTimeout fail [%x]\n", nRet);
        return;
    }

    //...other processing

    //Shut device and release resource
    nRet = MV_CC_CloseDevice(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: CloseDevice fail [%x]\n", nRet);
        return;
    }

    //Destroy handle and release resource
    nRet = MV_CC_DestroyHandle(m_handle);
    if (MV_OK != nRet)
    {
        printf("error: DestroyHandle fail [%x]\n", nRet);
        return;
    }
}
```

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_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

4.1.15 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

4.1.16 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

4.1.17 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.

4.1.18 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.

4.1.19 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.

4.1.20 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

4.1.21 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.

4.1.22 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

4.1.23 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.

4.1.24 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

4.1.25 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.

4.1.26 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.

4.1.27 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

4.2 Enumeration

4.2.1 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

4.2.2 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

4.2.3 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

4.2.4 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)
}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 Camera

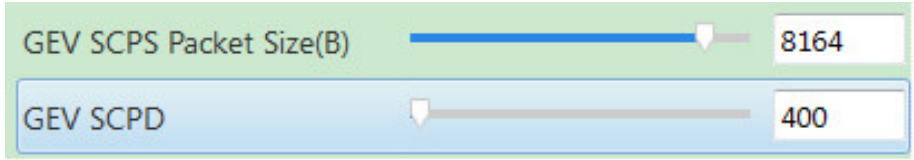
5.1.1 Why is there packet loss?

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

Improper understanding of API ***MV_CC_GetOneFrameTimeout*** . 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 USB3Vision 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** or **SuperSpeed**.

Appendix A. Appendixes

A.1 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
MV_OK	0	Succeeded.
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_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.

A.2 MvCameraNode

Most open properties of camera, including name, definition, data type, value range, and access mode of each node are described in the sheet below. And the actual supported nodes depend on the camera type.

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
Device Control	DeviceType	Ienumeration	0: Transmitter 1: Receiver 2: Transceiver 3: Peripheral	R	Device Type
	DeviceScanType	Ienumeration	0: Areascan 1: Linescan	R/(W)	Device sensor scanning type, show that is the line scan camera or area scan camera
	DeviceVendorName	Istring	Any Null-terminated String	R	Device manufacturer name
	DeviceModelName	Istring	Any Null-terminated String	R	Device type
	DeviceManufacturerInfo	Istring	Any Null-terminated String	R	Device manufacturer information
	DeviceVersion	Istring	Any Null-terminated String	R	Device version

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	DeviceFirmwareVersion	Istring	Any Null-terminated String	R	Firmware version
	DeviceSerialNumber	Istring	Any Null-terminated String	R	Device serial number
	DeviceID	Istring	Any Null-terminated String	R	Device ID
	DeviceUserID	Istring	Any Null-terminated String	R/W	User-defined name
	DeviceUptime	Iinteger	≥0	R	Device running time
	BoardDeviceType	Iinteger	≥0	R	Device type
	DeviceConnectionSelector	Iinteger	≥0	R/(W)	DeviceConnectionSelector
	DeviceConnectionSpeed	Iinteger	≥0, UnitMbps	R	DeviceConnectionSpeed
	DeviceConnectionStatus	Ienumeration	0: Active 1: Inactive	R	DeviceConnectionStatus
	DeviceLinkSelector	Iinteger	≥0	R/(W)	DeviceLinkSelector
	DeviceLinkSpeed	Iinteger	≥0	R	DeviceLinkSpeed
	DeviceLinkConnectionCount	Iinteger	≥0	R	DeviceLinkConnectionCount
	DeviceLinkHeartbeatMode	Ienumeration	0: Off 1: On	R/W	Whether to need a heartbeat
	DeviceLinkHeartbeatTimeout	Iinteger	500-600000	R/W	Heartbeat timeout
	DeviceStreamChannelCount	Iinteger	≥0	R	StreamChannelCount

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	DeviceStreamChannelSelector	Iinteger	≥0	R/W	StreamChannelSelector
	DeviceStreamChannelType	Ienumeration	0: Transmitter 1: Receiver	R	StreamChannelType
	DeviceStreamChannelLink	Iinteger	≥0	R/(W)	StreamChannelLinkCount
	DeviceStreamChannelEndianness	Ienumeration	0: Little 1: Big	R/(W)	ImageDataEndianness
	DeviceStreamChannelPacketSize	Iinteger	Related to camera. Generally range in 220-220, step 8 ;	R/(W)	ReceiverStreanDataPacketSize
	DeviceEventChannelCount	Iinteger	≥0	R	Device supported
	DeviceCharacterSet	Ienumeration	1: UTF-8 2: ASCII	R	DeviceRegisterUseTheCharacterSet
	DeviceReset	Icommand	-	W	ResetDevice
	DeviceTemperatureSelector	Ienumeration	0: Sensor 1: Mainboard	R/W	TheSelectedDeviceTemperatureMeasurement
	DeviceTemperature	Ifloat	Unit, degree	R	TheSelectedDeviceTemperature
	FindMe	Icommand	-	W	Find current device
	DeviceMaxThroughput	Iinteger	≥0	R	DeviceMaxThroughput(bandwidth)
	WidthMax	Iinteger	＞ 0	R	ImageWidthMax, the data after the binning

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	HeightMax	Iinteger	＞0	R	ImageHeightMax, the data after the binning
	RegionSelector	Ienumeration	0: Region0 1: Region1 2: Region2 8: All	R/(W)	ROISelector
	RegionDestination	Ienumeration	0: Stream0 1: Stream1 2: Stream2	R/(W)	The ROI corresponds to the stream
	Width	Iinteger	＞0	R/(W)	the Width of ROI
	Height	Iinteger	＞0	R/(W)	the Height of ROI
	OffsetX	Iinteger	≥0	R/W	the OffsetX of ROI
	OffsetY	Iinteger	≥0	R/W	the OffsetY of ROI
	ReverseX	Iboolean	True False	R/W	WhetherToNeedReverseX
	ReverseY	Iboolean	True False	R/W	WhetherToNeedReverseY
	ReverseScanDirection	Iboolean	≥0	R/(W)	ReverseScanDirection
	PixelFormat	Ienumeration	0x01080001: Mono8 0x01100003: Mono10 0x010C0004: Mono10Packed 0x01100005: Mono12 0x010C0006: Mono12Packed	R/(W)	TheImagePixelFormat, Different types of camera support different pixel format, take the actual as the standard.

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			0x01100007: Mono16 0x02180014: RGB8Packed 0x02100032: YUV422_8 0x0210001F: YUV422_8_UYVY 0x02180020: YUV8_UYV 0x020C001E: YUV411_8_ UYYVYY 0x01080008: BayerGR8 0x01080009: BayerRG8 0x0108000A: BayerGB8 0x0108000B: BayerBG8 0x0110000c: BayerGR10 0x0110000d: BayerRG10 0x0110000e: BayerGB10 0x0110000f: BayerBG10 0x010C0029: BayerBG10Packed		

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			0x010C0026: BayerGR10Packed 0x010C0027: BayerRG10Packed 0x010C0028: BayerGB10Packed 0x01100010: BayerGR12 0x01100011: BayerRG12 0x01100012: BayerGB12 0x01100013: BayerBG12 0x010C002D: BayerBG12Packed 0x010C002A: BayerGR12Packed 0x010C002B: BayerRG12Packed 0x010C002C: BayerGB12Packed 0x0110002E: BayerGR16 0x0110002F: BayerRG16 0x01100030: BayerGB16 0x01100031: BayerBG16		

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	PixelSize	Ienumeration	8 : Bpp8 10: Bpp10 12: Bpp12 16: Bpp16 24: Bpp24 32: Bpp32	R/(W)	a pixel contains the number of bits
	ImageCompressionMode	Ienumeration	0: Off 1: JPEG	R/W	ImageCompressionMode
	ImageCompressionQuality	Iinteger	≥50	R/(W)	ImageCompressionQuality
	TestPatternGeneratorSelector	Ienumeration	8: Sensor 0: Region0 1: Region1 2: Region2	R/W	TestPatternGeneratorSelector
	TestPattern[TestPatternGeneratorSelector]	Ienumeration	0 : Off 1: Black 2: White 3: GreyHorizontalRamp 4: GreyVerticalRamp 5: GreyHorizontalRampMoving 6: GreyVerticalRampMoving	R/W	TestImageSelector

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			7: HorizontalLineMoving 8: VerticalLineMoving 9: ColorBar 10: FrameCounter 11: MonoBar 12: TestImage12 13: TestImage13 14: ObliqueMonoBar 15: ObliqueColorBar 16: GradualMonoBar		
	BinningSelector	Ienumeration	0: Region0 1: Region1 2: Region2	R/W	PixelBinningSelector
	BinningHorizontal[BinningSelector]	Ienumeration	1: BinningHorizontal1 2: BinningHorizontal2 3: BinningHorizontal3 4: BinningHorizontal4	R/W	PixelBinningHorizontal
	BinningVertical[BinningSelector]	Ienumeration	1: BinningVertical1 2: BinningVertical2	R/W	PixelBinningVertical

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			3: BinningVertical3 4: BinningVertical4		
	DecimationHorizontal	Ienumeration	1: DecimationHorizontal1 2: DecimationHorizontal2 3: DecimationHorizontal3 4: DecimationHorizontal4	R/W	HorizontalPixelSampling
	DecimationVertical	Ienumeration	1: DecimationVertical1 2: DecimationVertical2 3: DecimationVertical3 4: DecimationVertical4	R/W	VerticalPixelSampling
	Deinterlacing	Ienumeration	0: Off 1: LineDuplication 2: Weave	R/W	Controls how the device performs de-interlacing

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	FrameSpecInfoSelector	Ienumeration	0 : Timestamp 1: Gain 2: Exposure 3: BrightnessInfo 4: WhiteBalance 5: Framecounter 6: ExtTriggerCount 7: LineInputOutput 8: ROIPosition	R/(W)	WatermarkInformationSelector
	FrameSpecInfo	Iboolean	True False	R/W	Whether to use the watermark informaton
AcquisitionControl	AcquisitionMode	Ienumeration	0: SingleFrame 1: MultiFrame 2: Continuous	R/(W)	AcquisitionMode, SingleFrame /MultiFrame / Continuous
	AcquisitionStart	Icommand	-	(R)/W	AcquisitionStart
	AcquisitionStop	Icommand	-	(R)/W	AcquisitionStop
	AcquisitionBurstFrameCount	Iinteger	≥1	R/(W)	AcquisitionBurstFrameCount
	AcquisitionFrameRate	Ifloat	≥0.0, Unitfps	R/W	The value is in effect when Trigger Mode is off
	AcquisitionFrameRateEnable	Iboolean		R/W	Set the frame rate whether to effect
	AcquisitionLineRate	Iinteger	≥1	R/W	LineRateSet
	AcquisitionLineRateEnable	Iboolean	True	R/W	LineRateControlEnable

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			False		
	ResultingLineRate	Iinteger	≥0, Unithz	R	ResultingLineRate
	ResultingFrameRate	Ifloat	≥0.0, Unitfps	R	The actual frame rate of the camera acquisition
	TriggerSelector	Ienumeration	0 : AcquisitionStart 1: AcquisitionEnd 2: AcquisitionActive 3: FrameStart 4: FrameEnd 5: FrameActive 6: FrameBurstStart 7: FrameBurstEnd 8: FrameBurstActive 9: LineStart 10: ExposureStart 11: ExposureEnd 12: ExposureActive	R/W	TriggerSelector
	TriggerMode[TriggerSelector]	Ienumeration	0: Off 1: On	R/W	TriggerMode
	TriggerSoftware[TriggerSelector]	Icommand	-	(R)/W	Perform a soft trigger
	TriggerSource[TriggerSelector]	Ienumeration	0: Line0 1: Line1 2: Line2 3.Line3	R/W	TriggerSource

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			4: Counter0 7: Software 8: FrequencyConverter		
	TriggerActivation[TriggerSelector]	Ienumeration	0: RisingEdge 1: FallingEdge 2.LevelHigh 3.LevelLow	R/W	TriggerActivation: RisingEdge FallingEdge LevelHigh LevelLow
	TriggerDelay[TriggerSelector]	Ifloat	≥0.0, Unitus	R/W	TriggerDelay
	TriggerCacheEnable	Iboolean	1 0	R/W	Whether to enable caching
	SensorShutterMode	Ienumeration	0: GlobalReset 1: TriggerRolling	R/W	Sensor exposure mode
	ExposureMode	Ienumeration	0: Timed 1: TriggerWidth	R/W	ExposureModeSelecton
	ExposureTime	Ifloat	≥0.0, Unitus	R/W	ExposureTime
	ExposureAuto	Ienumeration	0: Off 1: Once 2.Continuous	R/W	ExposureAuto
	AutoExposureTimeLowerLimit	Iinteger	≥2, Unitus	R/W	AutoExposureTimeLowerLimit
	AutoExposureTimeUpperLimit	Iinteger	≥2, Unitus	R/W	AutoExposureTimeUpperLimit
	GainShutPrior	Ienumeration	0: Shut 1: Gain	R/W	Sets the priority of the Exposure and Gain

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	FrameTimeoutEnable	Iboolean	0: Off 1: On	R/W	FrameTimeoutEnable
	FrameTimeoutTime	Iinteger	≥87, Unitms	R/W	FrameTimeoutTime
	HDREnable	Iboolean	1 0	R/W	Whether to enable wide dynamic
	HDRSelector	Iinteger	≥0, ≤3	R/W	HDRSelector
	HDRShuter	Iinteger	≥32	R/W	HDRShuter
	HDRGain	Ifloat	≥0	R/W	HDRGain
DigitalIOControl	LineSelector	Ienumeration	0: Line0 1: Line1 2: Line2 3: Line3 4: Line4	R/W	I/Oselection
	LineMode[LineSelector]	Ienumeration	0: Input 1: Output 2: Trigger 8: Strobe	R/W	I/Omode
	LineInverter[LineSelector]	Iboolean	1 0	R/W	I/OLevelConversion
	LineTermination	Iboolean	1 0	R/W	I/Osingle ended differential selection
	LineStatus[LineSelector]	Iboolean	-	R/(W)	I/Ostatus
	LineStatusAll	Iinteger	≥0	R	AlII/Ostatus

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	LineSource[LineSelector]	Ienumeration	0: ExposureActive	R/W	Output event source
	StrobeEnable	Iboolean	1 0	R/W	Enables the strobe signal to be output to the selected line
	LineDebouncerTime	Iinteger	-	R/W	I/O DebouncerTime
	StrobeLineDuration	Iinteger	≥0	R/W	OutputLevelDuration, Unit: us
	StrobeLineDelay	Iinteger	≥0	R/W	OuputDelay, Unit: us
	StrobeLinePreDelay	Iinteger	≥0	R/W	PreDelay, Unit: us
Counter And Timer Control	CounterSelector	Ienumeration	0: Counter0 1: Counter1 2: Counter2	R/W	CounterSelector
	CounterEventSource[CounterSelector]	Ienumeration	0: Off 11: Line0 12: Line1 13: Line2 1: AcquisitionTrigger 2: AcquisitionStart 3: AcquisitionEnd 4: FrameTrigger 5: FrameStart 6: FrameEnd 7: FrameBurstStart 8: FrameBurstEnd	R/W	CounterEventSource

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			9: FrameBurstEnd 10: LineEnd		
	CounterResetSource[CounterSelector]	Ienumeration	0: Off 1: CounterTrigger 3: Software 5: FrameTrigger 6: FrameStart	R/W	CounterResetSource
	CounterReset[CounterSelector]	Icommand	-	(R)/W	CounterReset
	CounterValue[CounterSelector]	Iinteger	≥1	R/W	CounterValue
	CounterCurrentValue	Iinteger	-	R	CounterCurrentValu
Analog Control	Gain[GainSelector]	Ifloat	≥0.0, UnitdB	R/W	Gain
	GainAuto[GainSelector]	Ienumeration	0: Off 1: Once 2.Continuous	R/W	GainAuto
	AutoGainLowerLimit	Ifloat	≥0.0, UnitdB	R/W	AutoGainLowerLimit
	AutoGainUpperLimit	Ifloat	≥0.1, UnitdB	R/W	AutoGainUpperLimit
	ADCGainEnable	Iboolean	0: Off 1: On	R/W	ADCGainEnable
	DigitalShift	Ifloat	≥0.0	R	DigitalShiftAdjustment
	DigitalShiftEnable	Iboolean	0: Off 1: On	R/W	DigitalShiftEnable

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	Brightness	Iinteger	≥0	R/W	Brightness
	BlackLevel[BlackLevelSelector]	Ifloat	≥0.0	R/W	BlackLevelAdjustment
	BlackLevelEnable	Iboolean	0: Off 1: On	R/W	BlackLeveAdjustmentlEnable
	BlackLevelAuto[BlackLevelSelector]	Ienumeration	0: Off 1: Continuous 2: Once	R/W	BlackLevelAdjustmentMode
	BalanceWhiteAuto	Ienumeration	0: Off 1: Continuous 2: Once	R/W	BalanceWhiteAuto
	BalanceRatioSelector	Ienumeration	0: Red 1: Green 2. Blue	R/W	WhiteBalanceRatioSelection
	BalanceRatio[BalanceRatioSelector]	Iinteger	≥0	R	WhiteBalanceRatio
	Gamma	Ifloat	＞ 0.0	R/W	GammaAdjustment
	GammaSelector	Ienumeration	1: User 2: sRGB	R/W	GammaSelection
	GammaEnable	Iboolean	0: Off 1: On	R/W	GammaEnable
	Sharpness	Iinteger	≥0	R/W	ImageSharpness
	SharpnessEnable	Iboolean		R/W	Enabel/DisableSharpness
	SharpnessAuto	Ienumeration	0: Off 1: Continuous 2: Once	R/W	SharpnessAutoType

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	Hue	Iinteger	≥0	R	HueAdjustment
	HueEnable	Iboolean	0: Off 1: On	R/W	HueEnable
	HueAuto	Ienumeration	0: Off 1: Continuous 2: Once	R/W	Gray automatic regulation type
	Saturation	Iinteger	≥0	R	SaturationAdjustment
	SaturationEnable	Iboolean	0: Off 1: On	R/W	SaturationEnable
	SaturationAuto	Ienumeration	0: Off 1: Continuous 2: Once	R/W	SaturationAuto
	DigitalNoiseReductionMode	Ienumeration	0: Off 1: Normal 2: Expert	R/W	DigitalNoiseReductionMode
	NoiseReduction	Iinteger	≥1	R/W	NoiseReductionValue
	AirspaceNoiseReduction	Iinteger	≥1	R/W	AirspaceNoiseReduction
	TemporalNoiseReduction	Iinteger	≥1	R/W	TemporalNoiseReduction
	AutoFunctionAOISelector	Ienumeration	0: AOI1 1: AOI2	R/W	AutoFunctionAOISelector
	AutoFunctionAOIWidth	Iinteger	≥0	R/W	AutoFunctionAOIWidth
	AutoFunctionAOIHeight	Iinteger	≥0	R/W	AutoFunctionAOIHeight

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	AutoFunctionAOIOffsetX	Iinteger	≥0	R	AutoFunctionAOIOffsetX
	AutoFunctionAOIOffsetY	Iinteger	≥0	R	AutoFunctionAOIOffsetY
	AutoFunctionAOIUsageIntensity	Iboolean	0: Off 1: On	R/W	Automatic exposure according to AOI area
	AutoFunctionAOIUsageWhiteBalance	Iboolean	0: Off 1: On	R	Automatic white balance according to AOI area
LUT Control	LUTSelector	Ienumeration	0: Luminance 1: Red 2: Green 3: Blue	R/W	Brightness、R\G\B
	LUTEnable[LUTSelector]	Iboolean	True False	R/W	Enable
	LUTIndex[LUTSelector]	Iinteger	≥0	R/W	Index
	LUTValue[LUTSelector][LUTIndex]	Iinteger	Device-specific	R/W	Value
	LUTValueAll[LUTSelector]	Register	Device-specific	R/W	AllLUTValue
Encoder Control	EncoderSelector	Ienumeration	0 : Encoder0 1: Encoder1 2: Encoder2	R/W	EncoderSelector
	EncoderSourceA	Ienumeration	0: Line0 1: Line1 2: Line2 3: Line3	R/W	EncoderSourceASelector

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	EncoderSourceB	Ienumeration	0: Line0 1: Line1 2: Line2 3: Line3	R/W	EncoderSourceBSelector
	EncoderTriggerMode	Ienumeration	0: AnyDirection 1: ForwardOnly	R/W	EncoderTriggerMode
	EncoderCounterMode	Ienumeration	0: IgnoreDirection 1: FollowDirection	R/W	EncoderCounterMode
	EncoderCounter	Iinteger	≥0	R	Encoder Counter value adjustment
	EncoderCounterMax	Iinteger	≥1	R/W	EncoderCounterMax
	EncoderCounterReset	Icommand	-	R/W	EncoderCounterReset
	EncoderMaxReverseCounter	Iinteger	≥1	R/W	EncoderMaxReverseCounter
	EncoderReverseCounterReset	Icommand	-	R/W	EncoderReverseCounterReset
FrequencyConverterControl	InputSource	Ienumeration	0: Line0 1: Line1 2: Line2 3: Line3	R/W	Frequency divider input source
	SignalAlignment	Ienumeration	0 : RisingEdge 1: FallingEdge	R/W	Frequency divider signal direction
	PreDivider	Iinteger	≥1	R/W	PreDividerAdjustment
	Multiplier	Iinteger	≥1	R/W	MultiplierAdjustment

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	PostDivider	Iinteger	≥1	R/W	PostDividerAdjustment
Shading Correction	ShadingSelector	Ienumeration	0: FPNCCorrection 1: PRNUCCorrection	R/W	ShadingSelector
	ActivateShading	Icommand	-	R/(W)	ActivateShading
	NUCEnable	Iboolean	0: Off 1: On	R/W	NUCEnableSwitch
	FPNCEnable	Iboolean	1: On 0: Off	R	FPNCStateSwitch
	PRNUCEnable	Iboolean	0: Off 1: On	R	PRNUCtateSwitch
User Set Control	UserSetCurrent	Iinteger	>=0	R	CurrentUserParameters
	UserSetSelector	Ienumeration	0: Default 1: User set 1 2: User set 2 3: User set 3	R/W	Set load parameters
	UserSetLoad[UserSetSelector]	Icommand	-	R/W	Load
	UserSetSave[UserSetSelector]	Icommand	-	(R)/W	UserParametersSave
	UserSetDefault	Ienumeration	0: Default 1: User set 1 2: User set 2 3: User set 3	R/W	Default

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
Transport Layer Control	PayloadSize	Iinteger	≥0	R	The size of a frame of data
	GevVersionMajor	Iinteger	GEV Main Version Number	R	GEVVersionMajor
	GevVersionMinor	Iinteger	GEV Deputy Version Number	R	GEVVersionMinor
	GevDeviceModeIsBigEndian	Iboolean	0: not BigEndian 1: Is BigEndian	R	BigEndian
	GevDeviceModeCharacterSet	Ienumeration	1: UTF8	R	CharacterSet
	GevInterfaceSelector	Iinteger	>=0	R/W	GEVInterfaceSelector
	GevMACAddress	Iinteger	Mac Address	R	MACAddress
	GevSupportedOptionSelector	Ienumeration	31 : UserDefinedName 30: SerialNumber 29: HeartbeatDisable 28: LinkSpeed 27 : CCPApplicationSocket 26: ManifestTable 25: TestData 24: DiscoveryAckDelay 23 : DiscoveryAckDelay Writable	R/W	Selects the GEV option to interrogate for existing support

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			22: ExtendedStatusCodes 21: PrimaryApplicationSwitchover 6 : Action 5: PendingAck 4: EventData 3: Event 2 : PacketResend 1: WriteMem 0: CommandsConcatenation 34: IPConfigurationLLA 33: IPConfigurationDHCP 32: IPConfigurationPersistentIP 63: PAUSEFrameReception 66: StreamChannelSourceSocket		

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	GevSupportedOption[GevSupportedOptionSelector]	Iboolean	0: Off 1: On	R	Indicates whether or not the selected GEV option is supported
	GevCurrentIPConfigurationLLA	Iboolean	0: Off 1: On	R	WhetherIPisLLA
	GevCurrentIPConfigurationDHCP[GevInterfaceSelector]	Iboolean	0: Off 1: On	R/W	WhetherIPisDHCP
	GevCurrentIPConfigurationPersistentIP[GevInterfaceSelector]	Iboolean	0: Off 1: On	R/W	WhetherIPisStaticIP
	GevPAUSEFrameReception[GevInterfaceSelector]	Iboolean	0: Off 1: On	R/W	Controls whether incoming PAUSE Frames are handled on the given logical link
	GevCurrentIPAddress[GevInterfaceSelector]	Iinteger	IP Address	R	IPAddress
	GevCurrentSubnetMask[GevInterfaceSelector]	Iinteger	Subnet Mask	R	SubnetMask
	GevCurrentDefaultGateway[GevInterfaceSelector]	Iinteger	Default Gateway	R	DefaultGateway
	GevFirstURL	Istring	-	R	XMLFirstURL
	GevSecondURL	Istring	-	R	XMLSecondURL
	GevNumberOfInterfaces	Iinteger	≥0	R	GEVNumberOfInterfaces

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	GevPersistentIPAddress[GevInterfaceSelector]	Iinteger	≥0	R/W	PersistentIPAddress
	GevPersistentSubnetMask[GevInterfaceSelector]	Iinteger	≥0	R/W	PersistentSubnetMask
	GevPersistentDefaultGateway[GevInterfaceSelector]	Iinteger	≥0	R/W	PersistentDefaultGateway
	GevLinkSpeed	Iinteger	≥0	R	LinkSpeed
	GevMessageChannelCount	Iinteger	≥0	R	MessageChannelCount
	GevStreamChannelCount	Iinteger	≥0	R	StreamChanne
	GevHeartbeatTimeout	Iinteger	≥0	R/W	HeartbeatTimeout
	GevGVCPHeartbeatDisable	Iboolean	0: Off 1: On	R/W	HeartbeatDisable
	GevTimestampTickFrequency	Iinteger	≥0, Unithz	R	TimestampTickFrequency
	GevTimestampControlLatch	Icommand	-	W	TimestampControlLatch
	GevTimestampControlReset	Icommand	-	W	TimestampControlReset
	GevTimestampControlLatchReset	Icommand	-	W	TimestampControlLatchReset
	GevTimestampValue	Iinteger	-	R	TimestampValue

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
	GevCCP	Ienumeration	0: OpenAcess 1: ExclusiveAccess 2: ControlAccess	R/W	App terminal control authority
	GevStreamChannelSelector	Iinteger	>=0	R/W	StreamChannelSelector
	GevSCPInterfaceIndex[GevStreamChannelSelector]	Iinteger	>=0	R/(W)	GEVInterfaceIndex
	GevSCPHostPort[GevStreamChannelSelector]	Iinteger	>=0	R/W	HostPort
	GevSCPDirectionGevStreamChannelSelector]	Iinteger	>=0	R	StreamChannelDirection
	GevSCPSFireTestPacket[GevStreamChannelSelector]	Iboolean	0: Off 1: On	R/W	Fire Test PacketEnable
	GevSCPSDoNotFragment[GevStreamChannelSelector]	Iboolean	0: Off 1: On	R/W	Fire Test PacketEnable
	GevSCPSBigEndian[GevStreamChannelSelector]	Iboolean	0: Off 1: On	R/W	Stream data size
	PacketUnorderSupport	Iboolean	0: Off 1: On	R/W	Whether to support GVSP package to send out-of-order
	GevSCPSPacketSize	Iinteger	＞ 0, related to camera. Generally	R/(W)	NetworkPacketSize

Function	Name (key) Node A[Node B]: Indicates that node A depends on node B	Data Type	Numerical Value Definition	Access Mode R: Read W: Write (W): Write without Acquisition	Description
			range in 220-220, step 8 ；		
	GevSCPD[GevStreamChannelSelector]	Iinteger	≥0	R/W	ContractDelay
	GevSCDA[GevStreamChannelSelector]	Iinteger	IP Address	R/W	Destination address for streaming data
	GevSCSP[GevStreamChannelSelector]	Iinteger	Port Number	R	Source port for streaming data
	TLParamsLocked	Iinteger	≥0, ≤1	R/W	When the flow is 1

