



Machine Vision Camera SDK (DotNet)

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 Development Environment	1
1.3 Update History	2
1.4 Notice	6
Chapter 2 Connect Device	8
Chapter 3 Image Acquisition and Display	10
3.1 Get Image Directly	10
3.2 Get Image in Callback Function	13
Chapter 4 API Reference	17
4.1 General	17
4.1.1 MvCamCtrl.NET::MyCamera::MV_CC_GetSDKVersion_NET	17
4.1.2 MvCamCtrl.NET::MyCamera::MV_CC_EnumerateTls_NET	17
4.1.3 MvCamCtrl.NET::MyCamera::MV_CC_EnumDevices_NET	19
4.1.4 MvCamCtrl.NET::MyCamera::MV_CC_EnumDevicesEx_NET	20
4.1.5 MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceAccessible_NET	22
4.1.6 MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET	26
4.1.7 MvCamCtrl.NET::MyCamera::MV_CC_CreateDeviceWithoutLog_NET	29
4.1.8 MvCamCtrl.NET::MyCamera::MV_CC_DestroyDevice_NET	31
4.1.9 MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET	34
4.1.10 MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET	37
4.1.11 MvCamCtrl.NET::MyCamera::MV_CC_GetDeviceInfo_NET	40
4.2 Parameter Settings	44
4.2.1 MvCamCtrl.NET::MyCamera::MV_CC_GetBoolValue_NET	44
4.2.2 MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET	47
4.2.3 MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET	51

4.2.4 MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET	54
4.2.5 MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValueByString_NET	58
4.2.6 MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET	64
4.2.7 MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET	67
4.2.8 MvCamCtrl.NET::MyCamera::MV_CC_GetIntValueEx_NET	71
4.2.9 MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET	74
4.2.10 MvCamCtrl.NET::MyCamera::MV_CC_GetStringValue_NET	77
4.2.11 MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET	81
4.2.12 MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET	84
4.2.13 MvCamCtrl.NET::MyCamera::MV_CC_ReadMemory_NET	88
4.2.14 MvCamCtrl.NET::MyCamera::MV_CC_WriteMemory_NET	91
4.3 Functional	95
4.3.1 General APIs	95
4.3.2 GigE APIs	172
4.3.3 CameraLink Camera	208
4.3.4 GenTL APIs	226
4.4 Image Acquisition	229
4.4.1 MvCamCtrl.NET::MyCamera::MV_CC_ClearImageBuffer_NET	229
4.4.2 MvCamCtrl.NET::MyCamera::MV_CC_FreeImageBuffer_NET	238
4.4.3 MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET	242
4.4.4 MvCamCtrl.NET::MyCamera::MV_CC_GetImageForBGR_NET	248
4.4.5 MvCamCtrl.NET::MyCamera::MV_CC_GetImageForRGB_NET	252
4.4.6 MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET	260
4.4.7 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackEx_NET	265
4.4.8 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForBGR_NET	270
4.4.9 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForRGB_NET	275
4.4.10 MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET	280
4.4.11 MvCamCtrl.NET::MyCamera::MV_CC_StopGrabbing_NET	283

4.5 Image Processing	287
4.5.1 MvCamCtrl.NET::MyCamera::MV_CC_DisplayOneFrame_NET	287
4.5.2 MvCamCtrl.NET::MyCamera::MV_CC_SaveImageEx2	292
4.5.3 MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET	293
4.5.4 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCvtQuality_NET	299
4.5.5 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaValue_NET	302
4.5.6 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaParam_NET	303
4.5.7 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMPParam_NET	304
4.5.8 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMPParamEx_NET	304
4.5.9 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCLUTParam_NET	305
4.5.10 MvCamCtrl.NET::MyCamera::MV_CC_ImageContrast_NET	305
4.5.11 MvCamCtrl.NET::MyCamera::MV_CC_ImageSharpen_NET	306
4.5.12 MvCamCtrl.NET::MyCamera::MV_CC_ColorCorrect_NET	306
4.5.13 MvCamCtrl.NET::MyCamera::MV_CC_NoiseEstimate	307
4.5.14 MvCamCtrl.NET::MyCamera::MV_CC_SpatialDenoise_NET	307
4.5.15 MvCamCtrl.NET::MyCamera::MV_CC_LSCCalib_NET	308
4.5.16 MvCamCtrl.NET::MyCamera::MV_CC_LSCCorrect_NET	308
4.5.17 MvCamCtrl.NET::MyCamera::MV_CC_HB_Decode_NET	309
4.5.18 MvCamCtrl.NET::MyCamera::MV_CC_RotateImage_NET	309
4.5.19 MvCamCtrl.NET::MyCamera::MV_CC_FlipImage_NET	309
4.5.20 MvCamCtrl.NET::MyCamera::MV_CC_StartRecord_NET	310
4.5.21 MvCamCtrl.NET::MyCamera::MV_CC_InputOneFrame_NET	319
4.5.22 MvCamCtrl.NET::MyCamera::MV_CC_StopRecord_NET	327
4.5.23 MvCamCtrl.NET::MyCamera::MV_CC_SaveImageToFile_NET	336
4.5.24 MvCamCtrl.NET::MyCamera::MV_CC_SavePointCloudData_NET	337
4.6 Advanced Settings	338
4.6.1 MvCamCtrl.NET::MyCamera::MV_GIGE_SetGvcptimeout_NET	338
4.6.2 MvCamCtrl.NET::MyCamera::MV_GIGE_GetGvcptimeout_NET	342

4.6.3 MvCamCtrl.NET::MyCamera::MV_GIGE_GetRetryGvcpTimes_NET	348
4.6.4 MvCamCtrl.NET::MyCamera::MV_GIGE_SetRetryGvcpTimes_NET	354
4.6.5 MvCamCtrl.NET::MyCamera::MV_GIGE_SetDiscoveryMode_NET	360
4.6.6 MvCamCtrl.NET::MyCamera::MV_USB_GetTransferSize_NET	361
4.6.7 MvCamCtrl.NET::MyCamera::MV_USB_SetTransferSize_NET	369
4.6.8 MvCamCtrl.NET::MyCamera::MV_USB_GetTransferWays_NET	376
4.6.9 MvCamCtrl.NET::MyCamera::MV_USB_SetTransferWays_NET	384
4.7 Camera Internal APIs	392
4.7.1 MvCamCtrl.NET::MyCamera::MV_CC_LocalUpgrade_NET	392
4.7.2 MvCamCtrl.NET::MyCamera::MV_CC_GetUpgradeProcess_NET	396
4.7.3 MvCamCtrl.NET::MyCamera::MV_XML_GetGenICamXML_NET	400
4.7.4 MvCamCtrl.NET::MyCamera::MV_XML_GetNodeInterfaceType_NET	403
4.7.5 MvCamCtrl.NET::MyCamera::MV_XML_GetNodeAccessMode_NET	409
Chapter 5 Data Structure and Enumeration	418
5.1 Data Structure	418
5.1.1 MV_ACTION_CMD_INFO	418
5.1.2 MV_ACTION_CMD_RESULT	419
5.1.3 MV_ACTION_CMD_RESULT_LIST	419
5.1.4 MV_ALL_MATCH_INFO	420
5.1.5 MV_CamL_DEV_INFO	420
5.1.6 MV_CC_CCM_PARAM	421
5.1.7 MV_CC_CCM_PARAM_EX	421
5.1.8 MV_CC_CLUT_PARAM	422
5.1.9 MV_CC_COLOR_CORRECT_PARAM	422
5.1.10 MV_CC_CONTRAST_PARAM	423
5.1.11 MV_CC_DEVICE_INFO	424
5.1.12 MV_CC_DEVICE_INFO_LIST	425
5.1.13 MV_CC_FILE_ACCESS	425

5.1.14 MV_CC_FILE_ACCESS_PROGRESS	426
5.1.15 MV_CC_FLIP_IMAGE_PARAM	426
5.1.16 MV_CC_FRAME_SPEC_INFO	427
5.1.17 MV_CC_GAMMA_PARAM	428
5.1.18 MV_CC_HB_DECODE_PARAM	428
5.1.19 MV_CC_INPUT_FRAME_INFO	429
5.1.20 MV_CC_LSC_CALIB_PARAM	429
5.1.21 MV_CC_LSC_CORRECT_PARAM	430
5.1.22 MV_CC_NOISE_ESTIMATE_PARAM	431
5.1.23 MV_CC_SPATIAL_DENOISE_PARAM	432
5.1.24 MV_CC_PIXEL_CONVERT_PARAM	434
5.1.25 MV_CC_RECORD_PARAM	436
5.1.26 MV_CC_RECT_I	437
5.1.27 MV_CC_ROTATE_IMAGE_PARAM	438
5.1.28 MV_CC_SHARPEN_PARAM	438
5.1.29 MV_DISPLAY_FRAME_INFO	439
5.1.30 MV_EVENT_OUT_INFO	440
5.1.31 MV_FRAME_OUT	441
5.1.32 MV_FRAME_OUT_INFO	441
5.1.33 MV_FRAME_OUT_INFO_EX	442
5.1.34 MV_GENTL_DEV_INFO	444
5.1.35 MV_GENTL_DEV_INFO_LIST	446
5.1.36 MV_GENTL_IF_INFO	446
5.1.37 MV_GENTL_IF_INFO_LIST	447
5.1.38 MV_GIGE_DEVICE_INFO	447
5.1.39 MV_IMAGE_BASIC_INFO	448
5.1.40 MV_MATCH_INFO_NET_DETECT	450
5.1.41 MV_MATCH_INFO_USB_DETECT	451

5.1.42 MV_NETTRANS_INFO	451
5.1.43 MV_RECORD_FORMAT_TYPE	452
5.1.44 MV_SAVE_IMAGE_PARAM	452
5.1.45 MV_SAVE_IMAGE_PARAM_EX	453
5.1.46 MV_SAVE_IMG_TO_FILE_PARAM	454
5.1.47 MV_SAVE_POINT_CLOUD_PARAM	455
5.1.48 MV_TRANSMISSION_TYPE_NET	456
5.1.49 MV_USB3_DEVICE_INFO	457
5.1.50 MV_XML_FEATURE_Boolean	459
5.1.51 MV_XML_FEATURE_Integer	460
5.1.52 MV_XML_NODE_FEATURE	461
5.1.53 MV_XML_NODES_LIST	462
5.1.54 MVCC_ENUMVALUE	462
5.1.55 MVCC_FLOATVALUE	463
5.1.56 MVCC_INTVALUE	463
5.1.57 MVCC_FLOATVALUE_EX	464
5.1.58 MVCC_STRINGVALUE	465
5.1.59 SPECIAL_INFO	465
5.2 Enumeration	466
5.2.1 EPixelType	466
5.2.2 MV_CAM_ACQUISITION_MODE	467
5.2.3 MV_CAM_BALANCEWHITE_AUTO	468
5.2.4 MV_CAM_EXPOSURE_AUTO_MODE	468
5.2.5 MV_CAM_GAIN_MODE	469
5.2.6 MV_CAM_GAMMA_SELECTOR	469
5.2.7 MV_CAM_TRIGGER_MODE	470
5.2.8 MV_CAM_TRIGGER_SOURCE	470
5.2.9 MV_CC_BAYER_NOISE_FEATURE_TYPE	471

5.2.10 MV_GIGE_EVENT	471
5.2.11 MV_GIGE_TRANSMISSION_TYPE	472
5.2.12 MV_CC_GAMMA_TYPE	473
5.2.13 MV_GRAB_STRATEGY	474
5.2.14 MV_IMG_FLIP_TYPE	474
5.2.15 MV_IMG_ROTATION_ANGLE	474
5.2.16 MV_SAVE_IAMGE_TYPE	475
5.2.17 MV_SAVE_POINT_CLOUD_FILE_TYPE	475
5.2.18 MV_XML_AccessMode	476
5.2.19 MV_XML_InterfaceType	477
5.2.20 MV_XML_Visibility	478
5.2.21 MvGvspPixelType	479
Chapter 6 FAQ (Frequently Asked Questions)	483
6.1 GigE Vision Camera	483
6.1.1 Why is there packet loss?	483
6.1.2 Why does link error occur in the normal compiled Demo?	483
6.1.3 Why can't I set the static IP under DHCP?	484
6.1.4 Why do I failed to perform the software trigger command when calling SDK?	484
6.1.5 Why does the camera often be offline?	484
6.1.6 Why is no permission returned when calling API MV_CC_OpenDevice_NET?	484
6.1.7 Why is there error code returned during debug process?	485
6.1.8 Why is no data error returned when calling API MV_CC_GetOneFrameTimeout_NET?	485
6.1.9 Why is there always no data when calling MV_CC_GetOneFrameTimeout_NET? ...	485
6.1.10 Why can't open the camera after finishing debugging abnormally?	486
6.2 USB3 Vision Camera	486
6.2.1 Why can't the MVS get the data or why is the frame rate far smaller the actual frame rare?	486
Appendix A. Error Code	487

Appendix B. Sample Code	493
B.1 Connect to Cameras via IP Address	493
B.2 Correct Color	497
B.3 Correct Lens Shading	503
B.4 Convert Pixel Format	511
B.5 Enhance Image	516
B.6 File Access	521
B.7 Get Images Directly	526
B.8 Get Images Directly with High Performance	531
B.9 Get Images in Callback Function	535
B.10 Get Images by Strategy	539
B.11 Get Images via Precision Time Protocol	546
B.12 Get Camera Events	552
B.13 Get Chunk Information	556
B.14 Import/Export Camera Feature File	561
B.15 Perform Basic Functions of CamLink Cameras	564
B.16 Recording	571
B.17 Save Images of 3D Cameras in Point Cloud Format	577
B.18 Set Multicast Mode	583
B.19 Spatial Denoising	588

Chapter 1 Overview

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

1.1 Introduction

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

1.2 Development Environment

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

Operating System

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

Development Folder Contents

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

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



Note

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

Prerequisites

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



Note

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

1.3 Update History

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

Summary of Changes in Version 3.4.0_Aug./2020

Version	Content
Version 3.4.0_Aug./2020	1. Added API for color correction: <i>MvCamCtrl.NET::MyCamera::MV_CC_ColorCorrect_NET</i> .
	2. Added API for setting gamma parameters of Bayer pattern: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaParam_NET</i> .
	3. Added API for enabling/disabling CCM and setting CCM parameters of Bayer pattern: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMParamEx_NET</i> .
	4. Added API for enabling/disabling CLUT and setting CLUT parameters of Bayer pattern: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCLUTParam_NET</i> .
	5. Added API for LSC calibration: <i>MvCamCtrl.NET::MyCamera::MV_CC_LSCCalib_NET</i> .
	6. Added API for LSC correction: <i>MvCamCtrl.NET::MyCamera::MV_CC_LSCCorrect_NET</i> .
	7. Added API for adjusting image contrast: <i>MvCamCtrl.NET::MyCamera::MV_CC_ImageContrast_NET</i> .
	8. Added API for adjusting image sharpness: <i>MvCamCtrl.NET::MyCamera::MV_CC_ImageSharpen_NET</i> .
	9. Added API for estimating noise: <i>MvCamCtrl.NET::MyCamera::MV_CC_NoiseEstimate</i> .
	10. Added API for spatial denoising: <i>MvCamCtrl.NET::MyCamera::MV_CC_SpatialDenoise_NET</i> .
	9. Added the sample code for correcting the color of the image of a camera with gamma, CCM, and CLUT: <i>Correct Color</i> .
	10. Added the sample code for enhancing the image of a camera by configuring contrast and sharpness: <i>Enhance Image</i> .
	11. Added the sample code for correcting lens shading: <i>Correct Lens Shading</i> .
	12. Added the sample code for denoising the image of a camera: <i>Spatial Denoising</i> .

Summary of Changes in Version 3.3.0_Mar./2020

Version	Content
Version 3.3.0_Mar./2020	1. Added API for setting device ACK packet type: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_SetDiscoveryMode_NET</i> .
	2. Added APIs for setting or getting GVSP streaming timeout: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_SetGvspTimeout_NET</i> , <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetGvspTimeout_NET</i> .
	3. Added APIs for setting or getting the maximum times one packet can be resent: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_SetResendMaxRetryTimes_NET</i> , <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetResendMaxRetryTimes_NET</i> .
	4. Added APIs for setting or getting the packet resending interval: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_SetResendTimeInterval_NET</i> , <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetResendTimeInterval_NET</i> .
	5. Added API for rotating pictures: <i>MvCamCtrl.NET::MyCamera::MV_CC_RotateImage_NET</i> .
	6. Added API for picture flip: <i>MvCamCtrl.NET::MyCamera::MV_CC_FlipImage_NET</i> .
	7. Added API for setting the gamma value after Bayer interpolation: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaValue_NET</i> .
	8. Added API for color correction after Bayer interpolation: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMPParam_NET</i> .
	9. Added API for lossless decoding: <i>MvCamCtrl.NET::MyCamera::MV_CC_HB_Decode_NET</i> .
	10. Added API for estimating noise based on pictures of Bayer pattern: .
	11. Added API for spatial noise reduction based on pictures of Bayer pattern: .
	12. Extended the pixel format enumeration <i>MvGvspPixelFormat</i> : added lossless decoding pixel formats.
	13. Delete the node sheet MvCameraNode.

Summary of Changes in Version 3.2.0_June/2019

Version	Content
Version 3.2.0_June/2019	1. Added API for getting multicast status: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetMulticastStatus_NET .</i>
	2. Added API for saving the 3D point cloud data: <i>MvCamCtrl.NET::MyCamera::MV_CC_SavePointCloudData_NET .</i>
	3. Added API for saving image to file: <i>MvCamCtrl.NET::MyCamera::MV_CC_SaveImageToFile_NET .</i>
	4. Added API for enumerating interfaces via GenTL: <i>MvCamCtrl.NET::MyCamera::MV_CC_EnumInterfacesByGenTL_NET .</i>
	5. Added API for enumerating devices via GenTL: <i>MvCamCtrl.NET::MyCamera::MV_CC_EnumDevicesByGenTL_NET .</i>
	6. Added API for creating a device handle via GenTL device information: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetMulticastStatus_NET .</i>
	7. Deleted the obsolete APIs.

Summary of Changes in Version 3.1.0_May/2019

Version	Content
Version 3.1.0_May/2019	1. Added API for setting streaming strategy: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetGrabStrategy_NET .</i>
	2. Added API for setting the output queue size: <i>MvCamCtrl.NET::MyCamera::MV_CC_SetOutputQueueSize_NET .</i>
	3. Added API for getting the current node type: <i>MvCamCtrl.NET::MyCamera::MV_XML_GetNodeInterfaceType_NET .</i>
	4. Added API for getting current node access mode: <i>MvCamCtrl.NET::MyCamera::MV_XML_GetNodeAccessMode_NET .</i>
	5. Added API for setting the GVCP command retransmission times: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_SetRetryGvcpTimes_NET .</i>
	6. Added API for getting the number of GVCP retransmission commands: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetRetryGvcpTimes_NET .</i>

Version	Content
	7. Added API for getting the GVCP command timeout: <i>MvCamCtrl.NET::MyCamera::MV_GIGE_GetGvcptimeout_NET .</i>
	8. Added API for clearing streaming data buffer: <i>MvCamCtrl.NET::MyCamera::MV_CC_ClearImageBuffer_NET .</i>
	9. Added API for setting the packet size of USB3 vision device: <i>MvCamCtrl.NET::MyCamera::MV_USB_SetTransferSize_NET .</i>
	10. Added API for getting the packet size of USB3 vision device: <i>MvCamCtrl.NET::MyCamera::MV_USB_GetTransferSize_NET .</i>
	11. Added API for setting the number of transmission channels for USB3 vision device: <i>MvCamCtrl.NET::MyCamera::MV_USB_SetTransferWays_NET .</i>
	12. Added API for getting the number of transmission channels for USB3 vision device: <i>MvCamCtrl.NET::MyCamera::MV_USB_GetTransferWays_NET .</i>

Summary of Changes in Version 3.0.0_April/2019

New document.

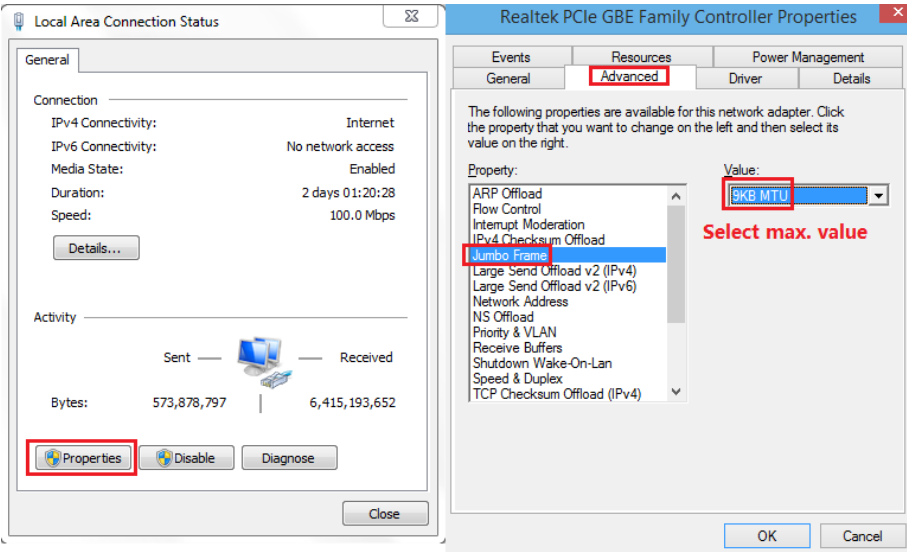
1.4 Notice

Install Camera Drive

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

NIC

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



Chapter 2 Connect Device

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

Steps

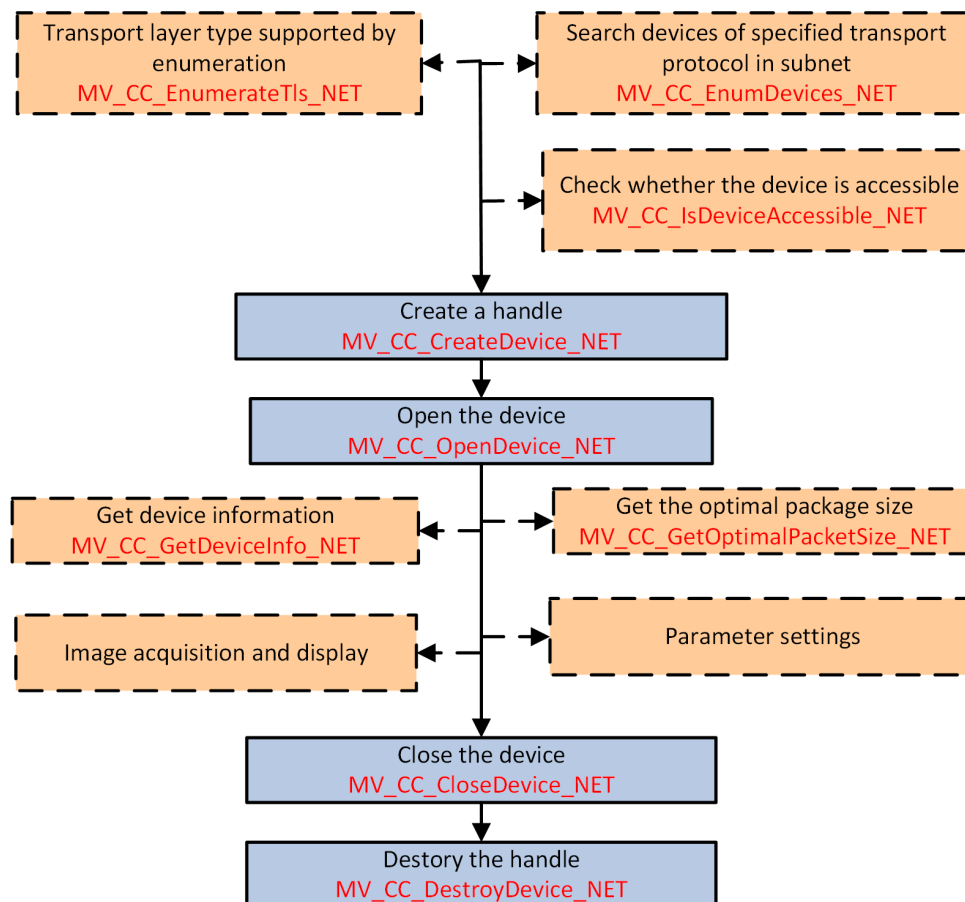


Figure 2-1 Programming Flow of Connecting Device

1. **Optional:** Call *MvCamCtrl.NET::MyCamera::MV_CC_EnumDevices_NET* 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 *stDevList*.
2. **Optional:** Call *MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceAccessible_NET* to check if the specified device is accessible before opening it.
3. Call *MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET* to create a device handle.
4. Call *MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET* to open the device.
5. **Optional:** Perform one or more of the following operations.

**Get Device
Information**

Call *MvCamCtrl.NET::MyCamera::MV_CC_GetDeviceInfo_NET*

**Get Optimal
Package Size**

Call *MvCamCtrl.NET::MyCamera::MV_CC_GetOptimalPacketSize_NET*

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 *MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET* to close the device.
8. Call *MvCamCtrl.NET::MyCamera::MV_CC_DestroyDevice_NET* to destroy the handle and release resources.

Chapter 3 Image Acquisition and Display

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

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

3.1 Get Image Directly

You can directly get image after starting getting stream via calling API `MV_CC_GetOneFrameTimeout_NET`. Asynchronous mode (thread or timer) and synchronous mode are both supported.

Steps

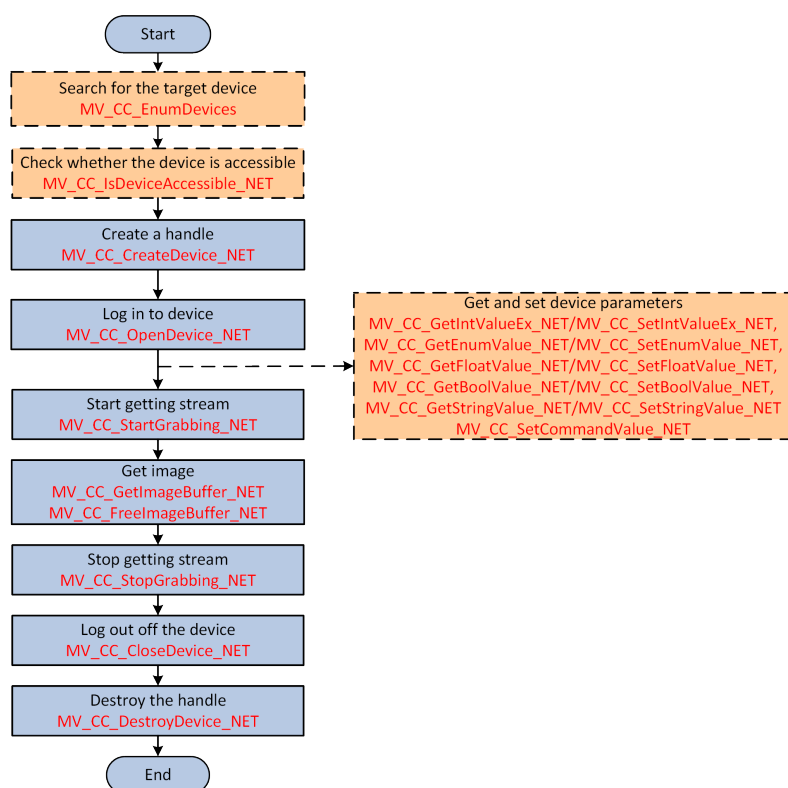


Figure 3-1 Programming Flow of Getting Image Directly

1. Call `MvCamCtrl.NET::MyCamera::MV_CC_EnumDevices_NET` 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` by `stDevList`.

2. **Optional:** Call *MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceAccessible_NET* to check if the specified device is accessible before opening it.
3. Call *MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET* 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 <i>MvCamCtrl.NET::MyCamera::MV_CC_GetBoolValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET</i>
Get/Set Camera Enum Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET</i>
Get/Set Camera Float Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET</i>
Get/Set Camera Int Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetIntValueEx_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET</i>
Get/Set Camera String Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetStringValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET</i>
Set Camera Command Node	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET</i>



Note

- You can get and set the acquisition mode including single frame acquisition, multi-frame acquisition, and continuous acquisition via the API *MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET* (handle, "AcquisitionMode", &stEnumValue) and *MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET* (handle, "AcquisitionMode", value).
- You can set triggering parameters.
 - a. Call *MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET* (handle, "TriggerMode", value) to set the triggering mode.
 - b. If the triggering mode is enabled, call *MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET* (handle, "TriggerSource", value) to set the triggering resource. The triggering source includes triggered by hardware and software.

- c. Call ***MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET*** (handle, "TriggerDelay", &stFloatValue) and ***MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET*** (handle, "TriggerDelay", value) to get and set the triggering delay time.
 - d. When triggered by software, call ***MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET*** (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:
MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET .
-

5. Call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start getting streams.
-



Note

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

6. Perform one of the following operations to acquire images.

- Call ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET*** repeatedly in the application layer to get the frame data with specified pixel format.
 - Call ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET*** in the application layer to get the frame data with specified pixel format and call ***MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET*** to release the buffer.
-



Note

- When getting the frame data, the application program should control the frequency of calling this API according to the frame rate.
- The differences of above two image acquisition methods are:
MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET should be used with ***MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET*** in pairs, the data pointer of **pstFrame** should be released by ***MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET*** .

Compared with ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET***, ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET*** is more efficient, and its stream buffer is allocated by SDK, while the stream buffer of ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET*** should be allocated by the developer.

- The above two methods and the method of acquiring image in callback function cannot be used at the same time.
 - The **pData** returns an address pointer, it is recommended to copy the data of **pData** to create another thread.
-

7. Acquire images.

- 1) Call ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET*** 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.

8. **Optional:** Call ***MvCamCtrl.NET::MyCamera::MV_CC_DisplayOneFrame_NET*** to input the window handle and start displaying.
9. Call ***MvCamCtrl.NET::MyCamera::MV_CC_StopGrabbing_NET*** to stop the acquisition or stop displaying.
10. Call ***MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET*** to close the device.
11. Call ***MvCamCtrl.NET::MyCamera::MV_CC_DestroyDevice_NET*** to destroy the handle and release resources.

3.2 Get Image in Callback Function

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

Steps

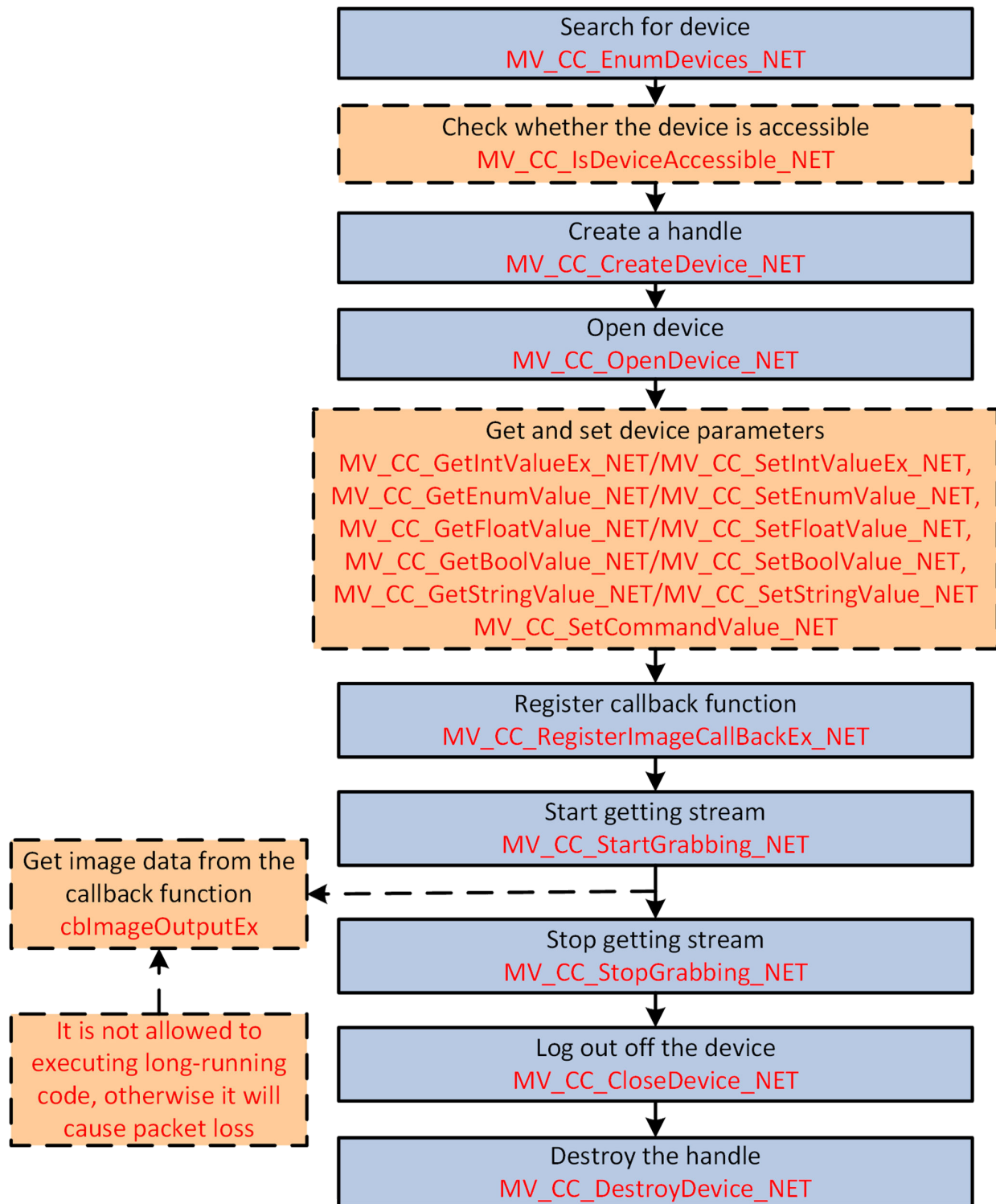


Figure 3-2 Programming Flow of Getting Image in Callback Function

1. Call ***MvCamCtrl.NET::MyCamera::MV_CC_EnumDevices_NET*** 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*** by ***stDevList***.
2. **Optional:** Call ***MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceAccessible_NET*** to check if the specified device is accessible before opening it.
3. Call ***MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET*** 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 <i>MvCamCtrl.NET::MyCamera::MV_CC_GetBoolValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET</i>
Get/Set Camera Enum Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET</i>
Get/Set Camera Float Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET</i>
Get/Set Camera Int Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetIntValueEx_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET</i>
Get/Set Camera String Node Value	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_GetStringValue_NET</i> / <i>MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET</i>
Set Camera Command Node	Call <i>MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET</i>

Note

- You can get and set the acquisition mode including single frame acquisition, multi-frame acquisition, and continuous acquisition via the API ***MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET*** (handle, "AcquisitionMode", &stEnumValue) and ***MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET*** (handle, "AcquisitionMode", value).
- You can set triggering parameters.
 - a. Call ***MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET*** (handle, "TriggerMode", value) to set the triggering mode.
 - b. If the triggering mode is enabled, call ***MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET*** (handle, "TriggerSource", value) to set the triggering resource. The triggering source includes triggered by hardware and software.

- c. Call ***MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET*** (handle, "TriggerDelay", &stFloatValue) and ***MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET*** (handle, "TriggerDelay", value) to get and set the triggering delay time.
 - d. When triggered by software, call ***MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET*** (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:
MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET ,
MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET .
-

5. Acquire images.

- 1) Call ***MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackEx_NET*** to set data callback function.
 - 2) Call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start getting steams.
-

Note

- Before starting the acquisition, you can call ***MvCamCtrl.NET::MyCamera::MV_CC_SetImageNodeNum_NET*** to set the number of image buffer nodes. When the number of obtained images is larger than this number, the earliest image data will be discarded automatically.
 - For original image data, you can call ***MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET*** to convert the image pixel format, or you can call ***MvCamCtrl.NET::MyCamera::MV_CC_SaveImageEx2*** ***MV_CC_SaveImageEx2_NET*** to convert the image to JPEG or BMP format and save as a file.
-

6. **Optional:** Call ***MvCamCtrl.NET::MyCamera::MV_CC_DisplayOneFrame_NET*** to input the window handle and start displaying.
7. Call ***MvCamCtrl.NET::MyCamera::MV_CC_StopGrabbing_NET*** to stop the acquisition or stop displaying.
8. Call ***MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET*** to close the device.
9. Call ***MvCamCtrl.NET::MyCamera::MV_CC_DestroyDevice_NET*** to destroy the handle and release resources.

Chapter 4 API Reference

4.1 General

4.1.1 MvCamCtrl.NET::MyCamera::MV_CC_GetSDKVersion_NET

Get SDK version No.

API Definition

```
static uint MV_CC_GetSDKVersion_NET(  
);
```

Return Value

Return SDK version No.:
| Main | Sub | Revision | Test
| 8bits | 8bits | 8bits | 8bits

Remarks

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

4.1.2 MvCamCtrl.NET::MyCamera::MV_CC_EnumerateTls_NET

Enumerate supported device type (transport layer type)

API Definition

```
static int MV_CC_EnumerateTls_NET(  
);
```

Return Value

Return supported device type, which is represented by bit, supporting multiple selection. The 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_CAMERA_LINK_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

C#

```
using System;
using System.IO;
using MvCamCtrl.NET;

namespace EnumerateTls
{
    class Program
    {
        static void Main(string[] args)
        {
            int nTransLayers = MyCamera.MV_CC_EnumerateTls_NET();
            if ((nTransLayers & MyCamera.MV_GIGE_DEVICE) == MyCamera.MV_GIGE_DEVICE)
            {
                Console.WriteLine("MV_GIGE_DEVICE");
            }
        }
    }
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        Get supported transport layer protocols
        int nTransLayers = dev.MV_CC_EnumerateTls_NET()
        If ((nTransLayers & MyCamera.MV_GIGE_DEVICE) = MyCamera.MV_GIGE_DEVICE) Then
            Console.WriteLine("MV_GIGE_DEVICE")
        End If

    End Sub
End Module
```

4.1.3 MvCamCtrl.NET::MyCamera::MV_CC_EnumDevices_NET

Enumerate all devices with specific transport protocol in the subnet.

API Definition

```
static int MV_CC_EnumDevices_NET(  
    uint                nLayerType,  
    ref MyCamera.MV_CC_DEVICE_INFO    stDevList  
);
```

Parameters

nLayerType

[IN] Transport layer protocol types, which is represented by bit, supporting multiple selections. The 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 `nLayerType == MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE`, it indicates that GigE device and USB3.0 device are both supported.

stDevList

[OUT] Matched device information list, see the structure ***MV_CC_DEVICE_INFO*** for details.

Return Values

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

Example

C#

```
using System;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace EnumDevices  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {
```

```
uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If
    End Sub
End Module
```

4.1.4 MvCamCtrl.NET::MyCamera::MV_CC_EnumDevicesEx_NET

Enumerate all devices with specified transport protocol and manufacturer in the subnet.

API Definition

```
static int MV_CC_EnumDevicesEx_NET(
    uint                nLayerType,
    ref MyCamera.MV_CC_DEVICE_INFO_LIST    stDevList,
    string                pManufacturerName
);
```

Parameters

nTLayerType

[IN] Transport layer protocol types, which is represented by bit, supporting multiple selections. The 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.

stDevList

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

pManufacturerName

[IN] Manufacturer name, e.g., "abc": enumerate abc cameras

Return Values

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

Example

C#

```
using System;
using System.IO;
using MvCamCtrl.NET;

namespace EnumDevicesEx
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevicesEx_NET(nTLayerType, ref stDevList, "abc");
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
            }
        }
    }
}
```

```
}  
}  
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading.Thread  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET  
  
Module Module1  
    Sub Main()  
        Dim dev As MyCamera = New MyCamera  
        Dim Info As String  
        Dim nRet As Int32 = MyCamera.MV_OK  
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST  
        'Enumerate devices  
        Dim nLayerType As Int32 = MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE  
        nRet = MyCamera.MV_CC_EnumDevicesEx_NET(nLayerType, stDeviceInfoList, "abc")  
        If MyCamera.MV_OK <> nRet Then  
            Console.WriteLine("Enumerating device failed!" + Convert.ToString(nRet))  
            Return  
        End If  
    End Sub  
End Module
```

4.1.5 MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceAccessible_NET

Check whether the device is accessible.

API Definition

```
static bool MV_CC_IsDeviceAccessible_NET(  
    ref MyCamera.MV_CC_DEVICE_INFO stDevInfo,  
    uint nAccessMode  
);
```

Parameters

stDevInfo

[IN] Information of specified device, see the structure **MV_CC_DEVICE_INFO** for details.

nAccessMode

[IN] Access mode, see details below:

Macro Definition	Value	Description
MyCamera.MV_ACCESS_Exclusive	1	Exclusive permission, for other Apps, only reading from CCP register is allowed.
MyCamera.MV_ACCESS_ExclusiveWithSwitch	2	Preempt permission from Mode 5, and then open as exclusive permission.
MyCamera.MV_ACCESS_Control	3	Control permission, for other Apps, all registers are allowed to read from.
MyCamera.MV_ACCESS_ControlWithSwitch	4	Preempt permission from Mode 5, and then open as control permission.
MyCamera.MV_ACCESS_ControlSwitchEnable	5	Open as the control permission which can be preempted.
MyCamera.MV_ACCESS_ControlSwitchEnableWithKey	6	Preempt permission from Mode 5, and then open as control permission which can be preempted.
MyCamera.MV_ACCESS_Monitor	7	Open device in read mode, which is applicable to control permission.

Return Value

Return true for accessible, and return false for no permission.

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.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace IsDeviceAccessible
```

```
{
class Program
{
    static void Main(string[] args)
    {
        uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            return;
        }
        Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            return;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;

        //Change the device information structure pointer to device information structure
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        MyCamera device = new MyCamera();

        //Create device
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Create device failed:{0:x8}", nRet);
            return;
        }

        //Check whether the specified device is accessible.
        uint nAccessMode = MyCamera.MV_ACCESS_Exclusive;
        bool bRet = MyCamera.MV_CC_IsDeviceAccessible_NET(ref stDevInfo, nAccessMode);
        if (true == bRet)
        {
            Console.WriteLine("Accessible");
        }
        else
        {
            Console.WriteLine("Unaccessible");
        }

        //Other process...

        //Destroy handle and release resources
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {

```

```
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Check whether the specified device is accessible
        Dim nAccessMode As UInt32 = MyCamera.MV_ACCESS_Exclusive
        Dim bRet As Boolean = MyCamera.MV_CC_IsDeviceAccessible_NET(stdevInfo, nAccessMode)
        If bRet = True Then
            Console.WriteLine("Access!")
        Else
            Console.WriteLine("Not Access!")
        End If

        '//Other process...

        'Stop streaming
        nRet = dev.MV_CC_StopGrabbing_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.1.6 MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET

Create handle.

API Definition

```
int MV_CC_CreateDevice_NET(
    ref MyCamera.MV_CC_DEVICE_INFO  stDevInfo
);
```

Parameters

stDevInfo

[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 Values

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

Remarks

- Create the required resources in the library and initialize the internal modules according to the device information. Create handle via this API, and log files will be automatically generated and saved in the folder of MvSdkLog under the directory of the current executable program. You can

call ***MvCamCtrl.NET::MyCamera::MV_CC_CreateDeviceWithoutLog_NET*** to create handle without generating logs.

- For version V2.4.1, added the function of log service, when it is enabled, the logs will be generated in the path: "C:\Users\{username}\MVS\MvSdkLog".

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace CreateDevice
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Other process...
```

```
//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        '//Other process...

        'Destroy handle
        nRet = dev.MV_CC_DestroyDevice_NET()
        If 0 <> nRet Then
```

```
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.1.7 MvCamCtrl.NET::MyCamera::MV_CC_CreateDeviceWithoutLog_NET

Create handle without generating logs.

API Definition

```
int MV_CC_CreateDeviceWithoutLog_NET(
    ref MyCamera.MV_CC_DEVICE_INFO  stDevInfo
);
```

Parameters

stDevInfo

[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 Values

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

Remarks

Create the required resources in the library and initialize the internal modules according to the device information. By default, log files will not be generated if you call this API to create handle. If you need logs, you can also call ***MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET*** to create handle, and the generated log files are saved in the folder of MvSdkLog under the directory of executable program.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace CreateDevice
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}
Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDeviceWithoutLog_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Other process...

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
```



```
Dim nRet As Int32 = MyCamera.MV_OK
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

'Enumerate device
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    MsgBox("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDeviceWithoutLog_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

//Other process...

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.1.8 MvCamCtrl.NET::MyCamera::MV_CC_DestroyDevice_NET

Destroy device instance and resources.

API Definition

```
int MV_CC_DestroyDevice_NET(
);
```

Return Values

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace DestroyDevice
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Other process...

            //Destroy handle and release resources
            nRet = device.MV_CC_DestroyDevice_NET();
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Destroy device failed:{0:x8}", nRet);
            }
        }
    }
}
```

```
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading.Thread  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET  
Module Module1  
    Sub Main()  
        Dim dev As MyCamera = New MyCamera  
        Dim Info As String  
        Dim nRet As Int32 = MyCamera.MV_OK  
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST  
  
        'Enumerate devices  
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),  
stDeviceInfoList)  
        If MyCamera.MV_OK <> nRet Then  
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))  
            Return  
        End If  
        If (0 = stDeviceInfoList.nDeviceNum) Then  
            Console.WriteLine("No Find Gige | Usb Device !")  
            Return  
        End If  
  
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO  
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),  
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)  
  
        'Create handle  
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)  
        If 0 <> nRet Then  
            Console.WriteLine("Create device failed!")  
        End If  
        Console.WriteLine("Create device succeed")  
  
        '//Other process...  
  
        'Destroy handle  
        nRet = dev.MV_CC_DestroyDevice_NET()  
        If 0 <> nRet Then  
            Console.WriteLine("Destroy device failed!")  
        End If  
        Console.WriteLine("Destroy device succeed!")  
    End Sub  
End Module
```

4.1.9 MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET

Open device (connect to device).

API Definition

```
int MV_CC_OpenDevice_NET(  
    uint      nAccessMode,  
    ushort    nSwitchoverKey  
);
```

Parameters

nAccessMode

[IN] Device access mode, the default mode is exclusive, see details in the following table:

Macro Definition	Value	Description
MyCamera.MV_ACCESS_Exclusive	1	Exclusive permission, for other Apps, only reading from CCP register is allowed.
MyCamera.MV_ACCESS_ExclusiveWithSwitch	2	Preempt permission from Mode 5, and then open as exclusive permission.
MyCamera.MV_ACCESS_Control	3	Control permission, for other Apps, all registers are allowed to read from.
MyCamera.MV_ACCESS_ControlWithSwitch	4	Preempt permission from Mode 5, and then open as control permission.
MyCamera.MV_ACCESS_ControlSwitchEnable	5	Open as the control permission which can be preempted.
MyCamera.MV_ACCESS_ControlSwitchEnableWithKey	6	Preempt permission from Mode 5, and then open as control permission which can be preempted.
MyCamera.MV_ACCESS_Monitor	7	Open device in read mode, which is applicable to control permission.

nSwitchoverKey

[IN] Key for switching permission. By default, it is none. It is valid when the access mode supporting permission switch (2/4/6 mode).

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- Find specific device and connect according to set device parameters.
- Inputting nAccessMode and nSwitchoverKey are optional when calling this API, and the default access mode is exclusive. The device does not support the following three modes: MyCamera.MV_ACCESS_ExclusiveWithSwitch, MyCamera.MV_ACCESS_ControlWithSwitch, and MyCamera.MV_ACCESS_ControlSwitchEnableWithKey.
- For USB3Vision camera, nAccessMode, nSwitchoverKey are invalid.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace OpenDevice
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
            nRet = device.MV_CC_OpenDevice_NET();
```

```
//nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
```

```
    stdevInfo =
CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),GetType(MyCamera.MV_CC_DEVICE_INFO)),
MyCamera.MV_CC_DEVICE_INFO)

    'Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Create device failed!")
    End If
    Console.WriteLine("Create device succeed")

    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed!")
    End If
    Console.WriteLine("Open device succeed!")

    //Other process...

    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Close device failed!")
    End If
    Console.WriteLine("Close device succeed!")

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

4.1.10 MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET

Shut the device.

API Definition

```
int MV_CC_CloseDevice_NET(
);
```

Return Values

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

Remarks

After calling ***MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET*** to connect to the device, you can call this API to disconnect the device and release the resources.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace CloseDevice
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }

            //Open device
            device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
            }
        }
    }
}
```



```
        return;
    }
    //Other process...

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
```

```
nRet = dev.MV_CC_CreateDevice_NET(stddevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")
//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.1.11 MvCamCtrl.NET::MyCamera::MV_CC_GetDeviceInfo_NET

Get the device information.

API Definition

```
int MV_CC_GetDeviceInfo_NET(
    ref MyCamera.MV_CC_DEVICE_INFO    pstDevInfo
);
```

Parameters

pstDevInfo

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace GetDeviceInfo
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDeviceInfo = new MyCamera.MV_CC_DEVICE_INFO();
nRet = device.MV_CC_GetDeviceInfo_NET(ref stDeviceInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get device info failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If
    End Sub
End Module
```

```
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)),
MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
nRet = dev.MV_CC_GetDeviceInfo_NET(stDeviceInfo)
If 0 <> nRet Then
    Console.WriteLine("Get DeviceInfo failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

4.2 Parameter Settings

4.2.1 MvCamCtrl.NET::MyCamera::MV_CC_GetBoolValue_NET

Get the value of camera boolean type node.

API Definition

```
int MV_CC_GetBoolValue_NET(  
    string      strKey,  
    ref bool    pbValue  
);
```

Parameters

strKey

[IN] Node name

pbValue

[OUT] Obtained node value

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {
```

```
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        return;
    }

    Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        return;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;

    //Change the device information structure pointer to device information structure
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    MyCamera device = new MyCamera();

    //Create device
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        return;
    }

    //Open device
    nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        return;
    }

    bool bValue = true;
    nRet = device.MV_CC_GetBoolValue_NET("ReverseX", ref bValue);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get Bool Value failed:{0:x8}", nRet);
        return;
    }

    //Other process...

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }
}
```

```
//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
    End Sub
End Module
```



```
End If
    Console.WriteLine("Open device succeed!")

    Dim bValue As Boolean
    nRet = dev.MV_CC_GetBoolValue_NET("ReverseX", bValue)
    If 0 <> nRet Then
        Console.WriteLine("Get Bool Value failed")
    End If

    //Other process...

    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Close device failed!")
    End If
    Console.WriteLine("Close device succeed!")

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.2 MvCamCtrl.NET::MyCamera::MV_CC_SetBoolValue_NET

Set the value of camera boolean type node.

API Definition

```
int MV_CC_SetBoolValue_NET(
    string    strKey,
    bool      bValue
);
```

Parameters

strKey

[IN] Node name

bValue

[IN] Node value

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

bool bValue = true;
nRet = device.MV_CC_SetBoolValue_NET("ReverseX", bValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Bool Value failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
```

```
'Enumerate device
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
    Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim bValue As Boolean = True
nRet = dev.MV_CC_SetBoolValue_NET("ReverseX", bValue)
If 0 <> nRet Then
    Console.WriteLine("Set Bool Value failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")
```

End Sub

End Module

4.2.3 MvCamCtrl.NET::MyCamera::MV_CC_GetEnumValue_NET

Get the value of camera Enum type node.

API Definition

```
int MV_CC_GetEnumValue_NET(  
    string          strKey,  
    ref MyCamera.MVCC_ENUMVALUE pEnumValue  
);
```

Parameters

strKey

[IN] Node name

pEnumValue

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
```

```
int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.MVCC_ENUMVALUE stEnumValue = new MyCamera.MVCC_ENUMVALUE();
nRet = device.MV_CC_GetEnumValue_NET("TriggerSource", ref stEnumValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Enum Value failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
}
```

```
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
```

```
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim stEnumValue As MyCamera.MVCC_ENUMVALUE = New MyCamera.MVCC_ENUMVALUE
nRet = dev.MV_CC_GetEnumValue_NET("TriggerSource", stEnumValue)
If 0 <> nRet Then
    Console.WriteLine("GetEnumValue failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.4 MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValue_NET

Set the value of camera Enum type node.

API Definition

```
int MV_CC_SetEnumValue_NET(
    string    strKey,
    uint      nValue
);
```

Parameters

strKey

[IN] Node name

nValue

[IN] Obtained node value

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
```

```
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        return;
    }

    //Open device
    nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        return;
    }

    uint enValue = (uint)MyCamera.MV_CAM_TRIGGER_SOURCE.MV_TRIGGER_SOURCE_SOFTWARE;
    nRet = device.MV_CC_SetEnumValue_NET("TriggerSource", enValue);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set Enum Value failed:{0:x8}", nRet);
        return;
    }

    //Other process...

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
```

```
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

'Enumerate device
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
    Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim enValue As Int32 = MyCamera.MV_CAM_TRIGGER_SOURCE.MV_TRIGGER_SOURCE_SOFTWARE
nRet = dev.MV_CC_SetEnumValue_NET("TriggerSource", enValue)
If 0 <> nRet Then
    Console.WriteLine("Set Enum Value failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
```

```
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.5 MvCamCtrl.NET::MyCamera::MV_CC_SetEnumValueByString_NET

Set the value of camera Enum type node.

API Definition

```
int MV_CC_SetEnumValueByString_NET(
    string strKey,
    string sValue
);
```

Parameters

strKey

[IN] Node name

sValue

[IN] Camera property string to be set

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;
namespace Events
{
    class Events
    {
        {
            public static MyCamera.cbEventdelegateEx EventCallback;
            public static MyCamera device;
            static void EventCallbackFunc(ref MyCamera.MV_EVENT_OUT_INFO pEventInfo, IntPtr pUser)
            {
```

```
        Console.WriteLine("EventName[" + pEventInfo.EventName + "], EventID[" + pEventInfo.nEventID + "]);
    }
    static void Main(string[] args)
    {

        // Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        int nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            return;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            return;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;

        // Print device information
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));
            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " + stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
                Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
                Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
                Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
            }
        }
        Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
        Int32 nDevIndex = Convert.ToInt32(Console.ReadLine());
        device = new MyCamera();
    }
}
```

```
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

// Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    return;
}

// Set Event of ExposureEnd On
nRet = device.MV_CC_SetEnumValueByString_NET("EventSelector", "ExposureEnd");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set EventSelector failed!");
    return;
}
nRet = device.MV_CC_SetEnumValueByString_NET("EventNotification", "On");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set EventNotification failed!");
    return;
}

// Register Event callback
EventCallback = new MyCamera.cbEventdelegateEx(EventCallbackFunc);
nRet = device.MV_CC_RegisterEventCallBackEx_NET("ExposureEnd", EventCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register event callback failed!");
    return;
}

// Start grab image
nRet = device.MV_CC_StartGrabbing_NET();
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}
Console.WriteLine("Push enter to exit");
Console.ReadLine();

// Stop grabbing
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

// Destroy handle
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    return;
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Events
    Dim dev As MyCamera = New MyCamera
    Private Sub cbEventdelegateFunc(ByRef pEventInfo As MyCamera.MV_EVENT_OUT_INFO, ByVal pUser As IntPtr)
        Dim Info As String
        Info = "EventName[" + pEventInfo.EventName + "], EventID[" + Convert.ToString(pEventInfo.nEventID) + "]"
        Console.WriteLine(Info)
    End Sub

Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
```

```
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
Dim cbCallback As MyCamera.cbEventdelegateEx = New MyCamera.cbEventdelegateEx(AddressOf
cbEventdelegateFunc)

'Enumerate device
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
    Return
End If
If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

'Print device information
Dim i As Int32
For i = 0 To stDeviceInfoList.nDeviceNum - 1
    Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
    stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nLayerType) Then
        Dim stGigInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigInfo, 0, stGigInfoPtr, 216)
        Dim stGigInfo As MyCamera.MV_GIGE_DEVICE_INFO
        stGigInfo = CType(Marshal.PtrToStructure(stGigInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
        Dim nlpByte1 As UInt32 = (stGigInfo.nCurrentIp And &HFF000000) >> 24
        Dim nlpByte2 As UInt32 = (stGigInfo.nCurrentIp And &HFF0000) >> 16
        Dim nlpByte3 As UInt32 = (stGigInfo.nCurrentIp And &HFF00) >> 8
        Dim nlpByte4 As UInt32 = (stGigInfo.nCurrentIp And &HFF)
        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigInfo.chUserDefinedName + "]IP[" + nlpByte1.ToString() +
"." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
        Console.WriteLine(Info)
    Else
        Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
        Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
        stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
        Console.WriteLine(Info)
    End If
Next
Console.WriteLine("please select a device")
Dim nIndex As Int32
nIndex = Console.ReadLine()
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```



```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stddevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
End If

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
nRet = dev.MV_CC_SetEnumValueByString_NET("EventSelector", "ExposureEnd")
If 0 <> nRet Then
    Console.WriteLine("Set Event Selector failed:{0:x8}", nRet)
End If
nRet = dev.MV_CC_SetEnumValueByString_NET("EventNotification", "On")
If 0 <> nRet Then
    Console.WriteLine("Set Event Notification failed:{0:x8}", nRet)
End If

'Register callback function for captured picture
nRet = dev.MV_CC_RegisterEventCallBackEx_NET("ExposureEnd", cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
End If

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
End If
Console.WriteLine("push enter to exit")
System.Console.ReadLine()

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)<
End If

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
```

End Sub

End Module

4.2.6 MvCamCtrl.NET::MyCamera::MV_CC_GetFloatValue_NET

Get the value of camera float type node.

API Definition

```
int MV_CC_GetFloatValue_NET(  
    string          strKey,  
    ref MyCamera.MVCC_FLOATVALUE pFloatValue  
);
```

Parameters

strKey

[IN] Node name

pFloatValue

[OUT] Obtained node value, see details in *MVCC_FLOATVALUE* .

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
```

```
int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("TriggerDelay", ref stFloatValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Float Value failed:{0:x8}", nRet);
    return;
}
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}
```

```
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim stFloatValue As MyCamera.MVCC_FLOATVALUE = New MyCamera.MVCC_FLOATVALUE
nRet = dev.MV_CC_GetFloatValue_NET("TriggerDelay", stFloatValue)
If 0 <> nRet Then
    Console.WriteLine("GetFloatValue failed")
End If
//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.7 MvCamCtrl.NET::MyCamera::MV_CC_SetFloatValue_NET

Set the value of camera float type node.

API Definition

```
int MV_CC_SetFloatValue_NET(
    string    strKey,
    float     fValue
);
```

Parameters

strKey

[IN] Node name

fValue

[IN] Node value

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

float fValue = 0;
nRet = device.MV_CC_SetFloatValue_NET("TriggerDelay", fValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Float Value failed:{0:x8}", nRet);
    return;
}
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
```

```
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
    Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim fValue As Double = 0
nRet = dev.MV_CC_SetFloatValue_NET("TriggerDelay", fValue)
If 0 <> nRet Then
    Console.WriteLine("Set FloatValue failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")
```


End Sub

End Module

4.2.8 MvCamCtrl.NET::MyCamera::MV_CC_GetIntValueEx_NET

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

API Definition

```
int MV_CC_GetIntValueEx_NET(  
    string                strKey,  
    ref MyCamera.MVCC_INTVALUE_EX  pIntValue  
);
```

Parameters

strKey

[IN] Node name

pIntValue

[OUT] Obtained node value, see details in *MVCC_FLOATVALUE_EX* .

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("TriggerDelay", ref stFloatValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Float Value failed:{0:x8}", nRet);
    return;
}
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}
```

```
//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
```

```
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

Dim stFloatValue As MyCamera.MVCC_FLOATVALUE = New MyCamera.MVCC_FLOATVALUE
nRet = dev.MV_CC_GetFloatValue_NET("TriggerDelay", stFloatValue)
If 0 <> nRet Then
    Console.WriteLine("GetFloatValue failed")
End If
//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.9 MvCamCtrl.NET::MyCamera::MV_CC_SetIntValueEx_NET

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

API Definition

```
int MV_CC_SetIntValueEx_NET(
    string    strKey,
    Int64     nValue
);
```

Parameters

strKey

[IN] Node name

nValue

[IN] Node value

Return Value

Return *MyCamera.MV_OK (0)* on success; 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. All the node values of "Integer" in the list can be set via this API. **strKey** corresponds to the Name column.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}
Int64 nValue = 1080;
nRet = device.MV_CC_SetIntValueEx_NET("Width", nValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Int Value failed:{0:x8}", nRet);
    return;
}

//...other processing
//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}
//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
```

```
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If
    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    'Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Create device failed!")
    End If
    Console.WriteLine("Create device succeed")
    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed!")
    End If
    Console.WriteLine("Open device succeed!")

    Dim nValue As Int64 = 1080
    nRet = dev.MV_CC_SetIntValueEx_NET("Width", nValue)
    If 0 <> nRet Then
        Console.WriteLine("Set Int Value failed")
    End If

    '//...other processing
    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Close device failed!")
    End If
    Console.WriteLine("Close device succeed!")
    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

4.2.10 MvCamCtrl.NET::MyCamera::MV_CC_GetStringValue_NET

Get the value of camera string type node.

API Definition

```
int MV_CC_GetStringValue_NET(
    string          strKey,
```

```
ref MyCamera.MVCC_STRINGVALUE pstValue
);
```

Parameters

strKey

[IN] Node name

pstValue

[OUT] Obtained node value, see details in ***MVCC_STRINGVALUE***.

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
```



```
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.MVCC_STRINGVALUE stStrValue = new MyCamera.MVCC_STRINGVALUE();
nRet = device.MV_CC_GetStringValue_NET("DeviceUserID", ref stStrValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get String Value failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        Dim stStrValue As MyCamera.MVCC_STRINGVALUE = New MyCamera.MVCC_STRINGVALUE
        nRet = dev.MV_CC_GetStringValue_NET("DeviceUserID", stStrValue)
        If 0 <> nRet Then
            Console.WriteLine("Get String Value failed")
        End If

        '//Other process...

        'Close camera
        nRet = dev.MV_CC_CloseDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.11 MvCamCtrl.NET::MyCamera::MV_CC_SetStringValue_NET

Set the value of camera string type node.

API Definition

```
int MV_CC_SetStringValue_NET(
    string    strKey,
    string    strValue
);
```

Parameters

strKey

[IN] Node name

strValue

[IN] Node value

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
```

```
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }

            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive, 0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                return;
            }

            nRet = device.MV_CC_SetStringValue_NET("DeviceUserID", "MyCamera");
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Set String Value failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stddevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
    Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
    Console.WriteLine("Open device succeed!")

nRet = dev.MV_CC_SetStringValue_NET("DeviceUserID", "MyCamera")
If 0 <> nRet Then
    Console.WriteLine("Set String Value failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.2.12 MvCamCtrl.NET::MyCamera::MV_CC_SetCommandValue_NET

Set the value of camera node with ICommand type.

API Definition

```
int MV_CC_SetCommandValue_NET(
    string    strKey
);
```

Parameters

strKey

[IN] Node name

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();
```

```
//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Setting command value failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
```



```
Dim dev As MyCamera = New MyCamera
Dim Info As String
Dim nRet As Int32 = MyCamera.MV_OK
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

'Enumerate device
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine ("Enumerating device failed."+ Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device failed.")
End If
    Console.WriteLine("Creating device succeeded.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening device failed.")
End If
    Console.WriteLine("Opening device succeeded.")

nRet = dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
If 0 <> nRet Then
    Console.WriteLine("Setting BalanceWhiteAuto failed.")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing device failed.")
End If
    Console.WriteLine("Closing device succeeded.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Destroying device failed.")
End If
    Console.WriteLine("Destroying device succeeded.")

End Sub

End Module
```

4.2.13 MvCamCtrl.NET::MyCamera::MV_CC_ReadMemory_NET

Read data from the device register.

API Definition

```
int MV_CC_ReadMemory_NET(
    IntPtr    pBuffer,
    long      nAddress,
    long      nLength
);
```

Parameters

pBuffer

[OUT] Data buffer, save the read memory size

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] Memory size to be read

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

Access the device and read the data from the register.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Param
{
```

```
class Program
{
    static void Main(string[] args)
    {
        uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            return;
        }
        Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            return;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;
        //Change the device information structure pointer to device information structure
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
        typeof(MyCamera.MV_CC_DEVICE_INFO));
        MyCamera device = new MyCamera();

        //Create device handle
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Create device failed:{0:x8}", nRet);
            return;
        }
        //Open device
        nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Open device failed:{0:x8}", nRet);
            return;
        }
        //Get Payload Size
        MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
        nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
            break;
        }
        uint nBufSize = stParam.nCurValue;

        //Read memory        IntPtr pBuffer = Marshal.AllocHGlobal(nBufSize);        Int32 nAddress = 0x0200;
        Int32 nLength = 536;        nRet = device.MV_CC_ReadMemory_NET(pBuffer, nAddress, nLength);        if
        (MyCamera.MV_OK != nRet)        {        Console.WriteLine("Read Memory failed:{0:x8}", nRet);
```

```
return;    } //Other process...
    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")
    End Sub
End Module
```

```
'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")
'Get Payload Size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
Exit Do
End If
Dim nPayloadSize As Int32 = stParam.nCurValue

'Read memory
Dim pBuffer As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim nAddress As Int32 = &H200
Dim nLength As Int32 = 536
nRet = dev.MV_CC_ReadMemory_NET(pBuffer, nAddress, nLength)
If 0 <> nRet Then
    Console.WriteLine("Read Memory failed!")
End If

//Other process...
'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("DemoClose camera failed!")
End If
Console.WriteLine("Demo Close camera succeeded!")
'Destroy handle

nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("DemoDestroy handle failed!")
End If
Console.WriteLine("Demo Destroy handle succeeded!")
End Sub
End Module
```

4.2.14 MvCamCtrl.NET::MyCamera::MV_CC_WriteMemory_NET

Write data into the device register.

API Definition

```
int MV_CC_WriteMemory_NET(
    IntPtr    pBuffer,
    long      nAddress,
```

```
long    nLength  
);
```

Parameters

pBuffer

[IN] Memory value to be write

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 *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enum device failed:{0:x8}", nRet);  
                return;  
            }  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;  
            //Change the device information structure pointer to device information structure
```

```
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));
        MyCamera device = new MyCamera();

        //Create device handle
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Create device failed:{0:x8}", nRet);
            return;
        }
        //Open device
        nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Open device failed:{0:x8}", nRet);
            return;
        }
        //Get Payload Size
        MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
        nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
            break;
        }
        uint nBufSize = stParam.nCurValue;

        //Write memory      IntPtr pBuffer = Marshal.AllocHGlobal(nBufSize);      pBuffer =
        Marshal.StringToHGlobalAnsi("YANG");      Int32 nAddress = 0x00000000000000e8;      Int32 nLength =
5120;      nRet = device.MV_CC_WriteMemory_NET(pBuffer, nAddress, nLength);      if (MyCamera.MV_OK !=
nRet)      {      Console.WriteLine("Write memory failed:{0:x8}", nRet);      return;      }

        //Other process...
        //Close device
        nRet = device.MV_CC_CloseDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Close device failed{0:x8}", nRet);
            return;
        }

        //Destroy handle and release resources
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")
        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")
        'Get Payload Size
        Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
        nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
        If (MyCamera.MV_OK <> nRet) Then
            Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
        Exit Do
        End If
        Dim nPayloadSize As Int32 = stParam.nCurValue

        'Write memory        Dim pBuffer As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
        Dim nAddress As Int32 = &HE8        Dim nLength As Int32 = 5120        nRet =
dev.MV_CC_WriteMemory_NET(pBuffer, nAddress, nLength)        If 0 <> nRet Then
Console.WriteLine("Write Memory failed!")        End If//Other process...
```



```
'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")
'Destroy handle

nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

4.3 Functional

4.3.1 General APIs

MvCamCtrl.NET::MyCamera::MV_CC_FileAccessRead_NET

Read files from camera.

API Definition

```
int MV_CC_FileAccessRead_NET(
    ref MyCamera.MV_CC_FILE_ACCESS  pstFileAccess
);
```

Parameters

pstFileAccess

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

Return Value

Return *MyCamera.MV_OK (0)* on success; 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.

Example

C#

```
using System;

using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Threading;
using System.Runtime.InteropServices;
using System.IO;
namespace ParametrizeCamera_FileAccess
{
    class Program
    {
        public static MyCamera device;
        public static int g_nRet = MyCamera.MV_OK;
        static void FileAccessProgress()
        {
            int nRet = MyCamera.MV_OK;
            MyCamera.MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = new
MyCamera.MV_CC_FILE_ACCESS_PROGRESS();
            while (true)
            {
                //Get progress of file access
                nRet = device.MV_CC_GetFileAccessProgress_NET(ref stFileAccessProgress);
                Console.WriteLine("State = {0:x8},Completed = {1},Total = {2}", nRet , stFileAccessProgress.nCompleted ,
stFileAccessProgress.nTotal);
                if (nRet != MyCamera.MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted
== stFileAccessProgress.nTotal))
                {
                    break;
                }
                Thread.Sleep(50);
            }
        }
        static void FileAccessThread()
        {
            MyCamera.MV_CC_FILE_ACCESS stFileAccess = new MyCamera.MV_CC_FILE_ACCESS();
            stFileAccess.pUserFileName = "UserSet1.bin";
            stFileAccess.pDevFileName = "UserSet1";
            //Read mode
            g_nRet = device.MV_CC_FileAccessRead_NET(ref stFileAccess);
            if (MyCamera.MV_OK != g_nRet)
            {
                Console.WriteLine("File Access Read failed:{0:x8}", g_nRet);
            }
        }
    }
}
```

```
static void Main(string[] args)
{
    device = new MyCamera();
    int nRet = MyCamera.MV_OK;
    do
    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList;
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}"
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General information of device
        //Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));
            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)

            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
                Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
                Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
                Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
            }
        }
        Console.WriteLine("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
    }
}
```

```
Int32 nDevIndex = Convert.ToInt32(Console.ReadLine());
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
//Read mode
Console.WriteLine("Read to file");
Thread hReadHandle = new Thread(FileAccessThread);
hReadHandle.Start();
Thread.Sleep(5);
Thread hReadProgressHandle = new Thread(FileAccessProgress);
hReadProgressHandle.Start();
hReadProgressHandle.Join();
hReadHandle.Join();
if (MyCamera.MV_OK == g_nRet)
{
    Console.WriteLine("File Access Read Success");
}
// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
// Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
}while (false);
if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
```

```
    }  
  }  
  Console.WriteLine("Press enter to exit");  
  Console.ReadKey();  
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET
```

```
Module ParametrizeCamera_FileAccess
```

```
    Dim dev As MyCamera = New MyCamera  
    Dim g_nRet As Int32 = MyCamera.MV_OK  
    Sub FileAccessProgress()  
        Dim nRet As Int32 = MyCamera.MV_OK  
        Dim stFileAccessProgress As MyCamera.MV_CC_FILE_ACCESS_PROGRESS = New  
MyCamera.MV_CC_FILE_ACCESS_PROGRESS()
```

```
        While (True)  
            'Get progress of file access  
            nRet = dev.MV_CC_GetFileAccessProgress_NET(stFileAccessProgress)  
            Console.WriteLine("State = {0:x8},Completed = {1},Total = {2}", nRet,  
stFileAccessProgress.nCompleted,stFileAccessProgress.nTotal)
```

```
        If (nRet <> MyCamera.MV_OK Or (stFileAccessProgress.nCompleted <> 0 And stFileAccessProgress.nCompleted =  
stFileAccessProgress.nTotal)) Then
```

```
            Return
```

```
        End If
```

```
        Thread.Sleep(50)
```

```
    End While
```

```
End Sub
```

```
Sub FileAccessThread()
```

```
    Dim stFileAccess As MyCamera.MV_CC_FILE_ACCESS = New MyCamera.MV_CC_FILE_ACCESS()
```

```
    stFileAccess.pUserFileName = "UserSet1.bin"
```

```
    stFileAccess.pDevFileName = "UserSet1"
```

```
    'Read mode
```

```
    g_nRet = dev.MV_CC_FileAccessRead_NET(stFileAccess)<
```

```
    If (MyCamera.MV_OK <> g_nRet) Then
```

```
        Console.WriteLine("File Access Read failed:{0:x8}", g_nRet)
```

```
    End If
```

```
End Sub
```

```
Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then

            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
            End If

        If (0 = stDeviceInfoList.nDeviceNum) Then<
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
            End If

        'Print device information
        Dim i As Int32<
        For i = 0 To stDeviceInfoList.nDeviceNum - 1
            Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
            stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

            If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
                Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
                stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
                Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
                Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
                Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
                Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)
                Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" + nlpByte1.ToString() +
"." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
                Console.WriteLine(Info)
            Else
                Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
                Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
                stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
                Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
                Console.WriteLine(Info)
            End If

            Next
        End While
    End Sub
```

```
Console.WriteLine("please select a device")
Dim nIndex As Int32
nIndex = Console.ReadLine()
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
Exit Do
End If

'Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

'Read mode
Console.WriteLine("Read to file")
Dim hReadHandle As New System.Threading.Thread(AddressOf FileAccessThread)
hReadHandle.Start()
Thread.Sleep(5)
Dim hReadProgressHandle As New System.Threading.Thread(AddressOf FileAccessProgress)
hReadProgressHandle.Start()
hReadProgressHandle.Join()
hReadHandle.Join()
If 0 = g_nRet Then
    Console.WriteLine("File Access Read Success")
End If

Console.WriteLine("")
'Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then
'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If

End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub
End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_FileAccessWrite_NET

Write local files to the camera.

API Definition

```
int MV_CC_FileAccessWrite_NET(
    ref MyCamera.MV_CC_FILE_ACCESS    pstFileAccess
);
```

Parameters

pstFileAccess

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

This API should be called after connecting to device.

Example

C#

```
using System;

using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Threading;
using System.Runtime.InteropServices;
using System.IO;
namespace ParametrizeCamera_FileAccess
{
    class Program
    {
        public static MyCamera device;
        public static int g_nRet = MyCamera.MV_OK;
        static void FileAccessProgress()
        {
```



```
int nRet = MyCamera.MV_OK;
MyCamera.MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = new
MyCamera.MV_CC_FILE_ACCESS_PROGRESS();
while (true)
{
    // Get progress of file access
    nRet = device.MV_CC_GetFileAccessProgress_NET(ref stFileAccessProgress);
    Console.WriteLine("State = {0:x8},Completed = {1},Total = {2}", nRet , stFileAccessProgress.nCompleted ,
stFileAccessProgress.nTotal);
    if (nRet != MyCamera.MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted
== stFileAccessProgress.nTotal))
    {
        break;
    }
    Thread.Sleep(50);
}
}
static void FileAccessThread()
{
    MyCamera.MV_CC_FILE_ACCESS stFileAccess = new MyCamera.MV_CC_FILE_ACCESS();
    stFileAccess.pUserFileName = "UserSet1.bin";
    stFileAccess.pDevFileName = "UserSet1";
    // Write mode
    g_nRet = device.MV_CC_FileAccessWrite_NET(ref stFileAccess);
    if (MyCamera.MV_OK != g_nRet)
    {
        Console.WriteLine("File Access Write failed:{0:x8}", g_nRet);
    }
}
static void Main(string[] args)
{
    device = new MyCamera();
    int nRet = MyCamera.MV_OK;
    do
    {
        // Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList;
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}"
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device information
        // Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
```

```
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));
    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
Int32 nDevIndex = Convert.ToInt32(Console.ReadLine());
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
// Write mode
Console.WriteLine("Write to file");
Thread hWriteHandle = new Thread(FileAccessThread);
hWriteHandle.Start();
Thread.Sleep(5);
Thread hWriteProgressHandle = new Thread(FileAccessProgress);
hWriteProgressHandle.Start();
```

```
hWriteProgressHandle.Join();
hWriteHandle.Join();
if (MyCamera.MV_OK == g_nRet)
{
    Console.WriteLine("File Access Write Success");
}
// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
// Destroy handle
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
}while (false);
if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module1

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
```

```
'Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
        Return
    End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find ABC Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

Dim stFileAccessWrite As MyCamera.MV_CC_FILE_ACCESS = New MyCamera.MV_CC_FILE_ACCESS()
stFileAccessWrite.pUserFileName = "UserSet1.txt"
stFileAccessWrite.pDevFileName = "UserSet1"
g_nRet = dev.MV_CC_FileAccessWrite_NET(stFileAccessWrite)
If 0 <> nRet Then

    Console.WriteLine("FileAccess Write failed")

End If

//...other process
'Stop streaming
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")
```

```
End Sub
```

```
End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_FeatureLoad_NET

Import camera feature files.

API Definition

```
int MV_CC_FeatureLoad_NET(
    string    pFileName
);
```

Parameters

pFileName

[IN] Feature file name.

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

Import the local features to the camera.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Param
{
```

```
class Program
{
    static void Main(string[] args)
    {
        uint nLayerType = MyCamera.MV_GIGE_DEVICE;
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

        int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
            return;
        }

        Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            return;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;

        //Change the device information structure pointer to device information structure
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        MyCamera device = new MyCamera();

        //Create device
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Create device failed:{0:x8}", nRet);
            return;
        }

        //Open device
        nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Open device failed:{0:x8}", nRet);
            return;
        }

        nRet = device.MV_CC_FeatureLoad_NET("CameraFile");
        if(MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Loading feature files failed:{0:x8}", nRet);
            return;
        }
    }
}
```

```
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device failed.")
End If
Console.WriteLine("The device is created.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening device failed.")
End If
Console.WriteLine("The device is open.")

nRet = dev.MV_CC_FeatureLoad_NET("CameraFile")
If 0 <> nRet Then
    Console.WriteLine("Loading camera feature files failed.")
End If
Console.WriteLine("The camera feature files are loaded.")

//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stopping acquisition failed.")
End If
Console.WriteLine("Acquisition is started.")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing device failed.")
End If
Console.WriteLine("The device is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")
```

End Sub

End Module

MvCamCtrl.NET::MyCamera::MV_CC_FeatureSave_NET

Save the camera feature files.

API Definition

```
int MV_CC_FeatureSave_NET(  
    string  pFileName  
);
```

Parameters

pFileName

[IN] Feature file name.

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

Save the features of each node of camera to the local PC.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
        }  
    }  
}
```

```
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_FeatureSave_NET("CameraFile");
if(MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Saving feature files failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Creating device failed.")
        End If
        Console.WriteLine("The device is created.")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Opening device failed.")
        End If
        Console.WriteLine("Opening device succeed.")

        nRet = dev.MV_CC_FeatureSave_NET("CameraFile")
        If 0 <> nRet Then
            Console.WriteLine("Saving camera feature files failed.")
        End If
        Console.WriteLine("The camera feature files are saved.")
    End Sub
End Module
```

```
//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stopping acquisition failed.")
End If
Console.WriteLine("Acquisition is started.")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing device failed.")
End If
Console.WriteLine("The device is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_GetAllMatchInfo_NET

Get the information of all types.

API Definition

```
int MV_CC_GetAllMatchInfo_NET(
    ref MyCamera.MV_ALL_MATCH_INFO    pstInfo
);
```

Parameters

pstInfo

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- Input information type (see nType in the structure of MyCamera.MV_ALL_MATCH_INFO) to the API to get the corresponding information (returned by the parameter plInfo in the structure of MyCamera. **MV_ALL_MATCH_INFO**).
- The prerequisite of calling this API is the obtained information types. Before getting the MyCamera.MV_MATCH_TYPE_NET_DETECT (MyCamera. **MV_MATCH_INFO_NET_DETECT**) information of GigE device, the acquisition must be enabled; Before getting the MyCamera.MV_MATCH_TYPE_USB_DETECT (MyCamera. **MV_MATCH_INFO_USB_DETECT**) information of USB3Vision camera, the device must be opened.
- This API is not supported by CameraLink device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace GGetAllMatchInfo
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
```

```
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

//Start the acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Starting the acquisition failed:{0:x8}", nRet);
    return;
}
Thread.Sleep(10 * 1000);

MyCamera.MV_ALL_MATCH_INFO stMatchImfo = new MyCamera.MV_ALL_MATCH_INFO();
MyCamera.MV_MATCH_INFO_NET_DETECT stMatchInfoNetDetect = new
MyCamera.MV_MATCH_INFO_NET_DETECT();

stMatchImfo.nInfoSize =
(uint)System.Runtime.InteropServices.Marshal.SizeOf(typeof(MyCamera.MV_MATCH_INFO_NET_DETECT));
stMatchImfo.nType = MyCamera.MV_MATCH_TYPE_NET_DETECT;
int size = Marshal.SizeOf(stMatchInfoNetDetect);
stMatchImfo.pInfo = Marshal.AllocHGlobal(size);
Marshal.StructureToPtr(stMatchInfoNetDetect, stMatchImfo.pInfo, false);

nRet = device.MV_CC_GetAllMatchInfo_NET(ref stMatchImfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get all match info failed:{0:x8}", nRet);
    return;
}

stMatchInfoNetDetect =
(MyCamera.MV_MATCH_INFO_NET_DETECT)Marshal.PtrToStructure(stMatchImfo.pInfo,
typeof(MyCamera.MV_MATCH_INFO_NET_DETECT));

Marshal.FreeHGlobal(stMatchImfo.pInfo);

//Other process...

//Stop the acquisition
nRet = device.MV_CC_StopGrabbing_NET();
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stopping the acquisition failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If
    End Sub
End Module
```

```
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device handle failed.")
End If
Console.WriteLine("The device handle is created.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening camera failed.")
End If
Console.WriteLine("The camera is open.")

'Start acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Starting acquisition failed:{0:x8}", nRet)
End If
Console.WriteLine("The acquisition is started.")

Sleep(10 * 1000)

Dim stMatchImfo As MyCamera.MV_ALL_MATCH_INFO = New MyCamera.MV_ALL_MATCH_INFO()
Dim stMatchInfoNetDetect As MyCamera.MV_MATCH_INFO_NET_DETECT = New
MyCamera.MV_MATCH_INFO_NET_DETECT

stMatchImfo.nInfoSize =
System.Runtime.InteropServices.Marshal.SizeOf(GetType(MyCamera.MV_MATCH_INFO_NET_DETECT))
stMatchImfo.nType = MyCamera.MV_MATCH_TYPE_NET_DETECT
Dim size As Int32 = Marshal.SizeOf(stMatchInfoNetDetect)
stMatchImfo.pInfo = Marshal.AllocHGlobal(size)
Marshal.StructureToPtr(stMatchInfoNetDetect, stMatchImfo.pInfo, False)

nRet = dev.MV_CC_GetAllMatchInfo_NET(stMatchImfo)
If 0 <> nRet Then
    Console.WriteLine("Get all match info failed:{0:x8}", nRet)
End If

stMatchInfoNetDetect = Marshal.PtrToStructure(stMatchImfo.pInfo,
GetType(MyCamera.MV_MATCH_INFO_NET_DETECT))
Marshal.FreeHGlobal(stMatchImfo.pInfo)

//Other process...

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
```



```
If 0 <> nRet Then
    Console.WriteLine("Stopping acquisition failed:{0:x8}", nRet)
End If
Console.WriteLine("The acquisition is stopped.")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

MvCamCtrl.NET::MvCamera::MV_CC_GetFileAccessProgress_NET

Get the progress of importing and exporting camera parameters.

API Definition

```
public int MV_CC_GetFileAccessProgress_NET(
    ref MyCamera.MV_CC_FILE_ACCESS_PROGRESS    pstFileAccessProgress
);
```

Parameters

pstFileAccessProgress

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;

using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Threading;
using System.Runtime.InteropServices;
```

```
using System.IO;
namespace ParametrizeCamera_FileAccess
{
    class Program
    {
        public static MyCamera device;
        public static int g_nRet = MyCamera.MV_OK;
        static void FileAccessProgress()
        {
            int nRet = MyCamera.MV_OK;
            MyCamera.MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = new
MyCamera.MV_CC_FILE_ACCESS_PROGRESS();
            while (true)
            {
                //Get file access progress
                nRet = device.MV_CC_GetFileAccessProgress_NET(ref stFileAccessProgress);
                Console.WriteLine("State = {0:x8},Completed = {1},Total = {2}", nRet , stFileAccessProgress.nCompleted ,
stFileAccessProgress.nTotal);
                if (nRet != MyCamera.MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted
== stFileAccessProgress.nTotal))
                {
                    break;
                }
                Thread.Sleep(50);
            }
        }
        static void FileAccessThread()
        {
            MyCamera.MV_CC_FILE_ACCESS stFileAccess = new MyCamera.MV_CC_FILE_ACCESS();
            stFileAccess.pUserFileName = "UserSet1.bin";
            stFileAccess.pDevFileName = "UserSet1";
            //Write mode
            g_nRet = device.MV_CC_FileAccessWrite_NET(ref stFileAccess);
            if (MyCamera.MV_OK != g_nRet)
            {
                Console.WriteLine("File Access Write failed:{0:x8}", g_nRet);
            }
        }
        static void Main(string[] args)
        {
            device = new MyCamera();
            int nRet = MyCamera.MV_OK;
            do
            {
                // Enumerate devices
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList;
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}"
                    break;
                }
            }
        }
    }
}
```

```
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo;
//General device information
//Print device information
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));
    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)

    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
Int32 nDevIndex = Convert.ToInt32(Console.ReadLine());
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
//Write mode
Console.WriteLine("Write to file");
Thread hWriteHandle = new Thread(FileAccessThread);
hWriteHandle.Start();
Thread.Sleep(5);
Thread hWriteProgressHandle = new Thread(FileAccessProgress);
hWriteProgressHandle.Start();
hWriteProgressHandle.Join();
hWriteHandle.Join();
if (MyCamera.MV_OK == g_nRet)
{
    Console.WriteLine("File Access Write Success");
}
// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
}while (false);
if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
```

```
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find ABC Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")
        'Open device
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        Dim stFileAccessWrite As MyCamera.MV_CC_FILE_ACCESS_PROGRESS = New
MyCamera.MV_CC_FILE_ACCESS_PROGRESS()
        g_nRet = dev.MV_CC_GetFileAccessProgress_NET(stFileAccessProgress)
        If nRet != MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted ==
stFileAccessProgress.nTotal Then
            break
        End If

        '//...other processing
        'Stop image acquisition
        nRet = dev.MV_CC_StopGrabbing_NET()
        If 0 <> nRet Then
            Console.WriteLine("Stop grabbing failed!")
        End If
        Console.WriteLine("Start grabbing succeed!")

        'Close device
```

```
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")
'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_InvalidateNodes_NET

Clear GenICam node cache.

API Definition

```
int MV_CC_InvalidateNodes_NET(
);
```

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Param
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

Int64 nValue = 1080;
nRet = device.MV_CC_SetIntValueEx_NET("Width", nValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Int Value failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_InvalidateNodes_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("InvalidateNodes failed:{0:x8}", nRet);
    return;
}

//...other processing
```

```
//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Module1

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    'Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
        Return
    End If

    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device !")
        Return
    End If

    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```



```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

Dim nValue As Int64 = 1080
nRet = dev.MV_CC_SetIntValueEx_NET("Width", nValue)
If 0 <> nRet Then
    Console.WriteLine("Set Int Value failed")
End If

nRet = dev.MV_CC_InvalidateNodes_NET()
If 0 <> nRet Then
    Console.WriteLine("InvalidateNodes failed")
End If

//...other processing

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_IsDeviceConnected_NET

Check if device is connected.

API Definition

```
bool MV_CC_IsDeviceConnected_NET(
);
```

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace Program
{
    class Program
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }
                MyCamera.MV_CC_DEVICE_INFO stDevInfo; //General device information
                stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
                if (m_stDevList.nDeviceNum == 0)
                {
                    Console.WriteLine("no camera found!\n");
                    return;
                }
            }
        }
    }
}
```

```
}

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

bool blsConnect = device.MV_CC_IsDeviceConnected_NET();
if (blsConnect)
{
    Console.WriteLine("Device is connected");
    break;
}
else
{
    Console.WriteLine("Device is unconnected");
    break;
}
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resource
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Program
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Do While (True)

            'Enumerate device
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
            If MyCamera.MV_OK <> nRet Then
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)
                Exit Do
            End If
            If (0 = stDeviceInfoList.nDeviceNum) Then
                Console.WriteLine("No Find Gige | Usb Device !")
                Exit Do
            End If
            Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
            stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
            If (0 = m_stDeviceInfoList.nDeviceNum)
            Then
                MsgBox("No Find Gige | Usb Device !")
                Return
            End If

            'Create device handle
            nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
            If 0 <> nRet Then
                Console.WriteLine("Create device failed:{0:x8}", nRet)
                Exit Do
            End If

            'Open device
            nRet = dev.MV_CC_OpenDevice_NET()
            If 0 <> nRet Then
                Console.WriteLine("Open device failed:{0:x8}", nRet)
                Exit Do
            End If
            Dim blsConnect As Boolean
            blsConnect = dev.MV_CC_IsDeviceConnected_NET()
            If blsConnect Then
                Console.WriteLine("Device is connected", nRet)
            Else
                Console.WriteLine("Device is unconnected", nRet)
            End If
        End Do
    End Sub
End Module
```

```
Exit Do
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_RegisterAllEventCallBack_NET

Register the callback function for all events.

API Definition

```
int MV_CC_RegisterAllEventCallBack_NET(
    cbEventdelegate    cbEvent,
    IntPtr             pUser
);
```

Parameters

cbEvent

[IN] Callback function for receiving events.

```
void cbEventdelegate(
    ref MV_EVENT_OUT_INFO pEventInfo,
    IntPtr                pUser
);
```

pEventInfo

[OUT] Output event information, see details in **MV_EVENT_OUT_INFO**

pUser

[OUT] User data

pUser

[IN] User data

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace RegisterEventCallBack
{
    class Program
    {
        static void EventCallBack(ref MyCamera.MV_EVENT_OUT_INFO pEventInfo, IntPtr pUser)
        {
            Console.WriteLine("EventName:"+pEventInfo.EventName);
        }

        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();
```

```
//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}
//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.cbEventdelegate EvCallback;
EvCallback = new MyCamera.cbEventdelegate(EventCallBack);
nRet = device.MV_CC_RegisterAllEventCallBack_NET(EvCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register Event CallBack failed{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Event_Callback
```

```
Dim dev As MyCamera = New MyCamera
Dim pBufForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)'You should allocate the memory size
according to camera resolution
Dim m_bytImageBuffer(1024 * 1024 * 60) As Byte
Dim m_bytImageBufferLen As Int32 = 1024 * 1024 * 60
Private Sub EventCallBack(ByRef pEventInfo As MyCamera.MV_EVENT_OUT_INFO, ByVal pUser As IntPtr)
Console.WriteLine("EventName:" + pEventInfo.EventName) End Sub
Sub Main()
Dim Info As String
Dim nRet As Int32 = MyCamera.MV_OK
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
Dim EvCallback As MyCamera.cbEventdelegate = New MyCamera.cbEventdelegate(AddressOf EventCallBack)
'Enumerate devices
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
Return
End If
If (0 = stDeviceInfoList.nDeviceNum) Then
Console.WriteLine("No Find Gige | Usb Device !")
Return
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
Console.WriteLine("Create device failed:{0:x8}", nRet)
End If
'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
nRet = dev.MV_CC_RegisterAllEventCallBack_NET(EvCallback, IntPtr.Zero)
If 0 <> nRet Then
Console.WriteLine("Register All Event CallBack failed")
End If
'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
End If
Console.WriteLine("push enter to exit")
System.Console.ReadLine()
'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
End If
```



```
'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
'Destroy handle

nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
End Sub
```

MvCamCtrl.NET::MvCamera::MV_CC_RegisterEventCallBackEx_NET

Register single event callback function.

API Definition

```
public int MV_CC_RegisterEventCallBackEx_NET(
    string          pEventName,
    MyCamera.cbEventdelegateEx cbEvent,
    IntPtr          pUser
);
```

Parameters

pEventName

[IN] Event name

cbEvent

[IN] Callback function for receiving the event.

```
void cbEventdelegate(
    ref MyCamera.MV_EVENT_OUT_INFO pEventInfo,
    IntPtr          pUser
);
```

pEventInfo

[OUT] Output event information, see details in **MV_EVENT_OUT_INFO**

pUser

[OUT] User data

pUser

[IN] User data

Return Value

Return *MyCamera.MV_OK (0)* on success; 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.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace RegisterEventCallBack
{
    class Program
    {
        static void EventCallBack(ref MyCamera.MV_EVENT_OUT_INFO pEventInfo, IntPtr pUser)
        {
            Console.WriteLine("EventName:"+pEventInfo.EventName);
        }

        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
```

```
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.cbEvent EvCallback;
EvCallback = new MyCamera.cbEventdelegate(EventCallBack);
nRet = device.MV_CC_RegisterEventCallBackEx_NET(EvCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register Event CallBack failed{0:x8}", nRet);
    return;
}

//other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Event_Callback
    Dim dev As MyCamera = New MyCamera
    Dim pBufferForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)
    'You should allocate the memory size according to camera resolution
    Dim m_byteImageBuffer(1024 * 1024 * 60) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 60
    Private Sub EventCallBack(ByRef pEventInfo As MyCamera.MV_EVENT_OUT_INFO, ByVal pUser As IntPtr)
        Console.WriteLine("EventName:" + pEventInfo.EventName)
    End Sub
```

```
Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim EvCallback As MyCamera.cbEventdelegate = New MyCamera.cbEventdelegate(AddressOf EventCallBack)
    'Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
        Return
    End If
    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find ABC Gige | Usb Device !")
        Return
    End If
    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    ' Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Create device failed:{0:x8}", nRet)
    End If
    'Open device
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed:{0:x8}", nRet)
    End If
    nRet = dev.MV_CC_RegisterEventCallBackEx_NET(EvCallback, IntPtr.Zero)
    If 0 <> nRet Then
        Console.WriteLine("Register All Event CallBack failed")
    End If
    'Start getting stream
    nRet = dev.MV_CC_StartGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    End If
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
    'Stop getting stream
    nRet = dev.MV_CC_StopGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    End If
    'Close device
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed:{0:x8}", nRet)
    End If
    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_RegisterExceptionCallBack_NET

Register callback function for exception information.

API Definition

```
int MV_CC_RegisterExceptionCallBack_NET(
    cbExceptiondelegate    cbException,
    IntPtr                pUser
);
```

Parameters

cbException

[IN] Callback function for receiving exception message

```
void cbExceptiondelegate(
    uint    nMsgType,
    IntPtr  pUser
);
```

nMsgType

[OUT] Exception information type, see details in the following table:

Macro Definition	Value	Description
MV_EXCEPTION_DEV_DISCONNECT	0x00008001	Device disconnected.

pUser

[IN] User data

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- Call this API after calling **MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET** to open device. You can get the exception information from the callback function if the device is abnormally disconnected. But for GigE device, after offline, you should call

MvCamCtrl.NET::MyCamera::MV_CC_CloseDevice_NET first to close the device, and then call ***MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET*** to open the device again.

- This API is not supported by CameraLink device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET; namespace RegisterExceptionCallBack
{
    class Program
    {
        static void ExceptionCallBack(UInt32 nMsgType, IntPtr pUser)
        {
            Console.WriteLine("Device disconnected!");
        }

        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();      int nRet
            = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;          //Change the device information structure pointer to
device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));          MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
MyCamera.cbExceptiondelegate ExCallback;
ExCallback = new MyCamera.cbExceptiondelegate(ExceptionCallBack);    nRet =
device.MV_CC_RegisterExceptionCallBack_NET(ExCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register Exception CallBack failed{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Exception_Callback
    Dim dev As MyCamera = New MyCamera
    Dim pBufferForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)'You should allocate the memory size
    according to camera resolution
    Dim m_byteImageBuffer(1024 * 1024 * 60) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 60
    Private Sub ExceptionCallBack(ByVal nMsgType As Integer, ByVal pUser As IntPtr)
        Console.WriteLine("Device disconnected!")
    End Sub
    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim exCallBack As MyCamera.cbExceptiondelegate = New MyCamera.cbExceptiondelegate(AddressOf
ExceptionCallBack)
        'Enumerate devices
```

```
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
    Return
End If
If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
End If
'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
nRet = dev.MV_CC_RegisterExceptionCallBack_NET(exCallBack, IntPtr.Zero)
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
End If
Console.WriteLine("push enter to exit")
System.Console.ReadLine()
'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
End If
'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
'Destroy handle

nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
End Sub
End Module
```


MvCamCtrl.NET::MyCamera::MV_CC_SetImageNodeNum_NET

Set number of image buffer nodes.

API Definition

```
int MV_CC_SetImageNodeNum_NET(  
    uint    nNum  
);
```

Parameters

nNum

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

Return Values

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- This API must be called before calling **MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET** to start capture.
- This API is not supported by CameraLink device.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace SetImageNodeNum  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
        }  
    }  
}
```

```
Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_SetImageNodeNum_NET(30);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set image NodeNum failed:{0:x8}", nRet);
    return;
}

//Start capture
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start capture failed:{0:x8}", nRet);
    return;
}

//Other process...

//Stop capture
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
}
```

```
        return;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroying device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
```

```
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stddevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device handle failed.")
End If
Console.WriteLine("The device handle is created.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening device failed.")
End If
Console.WriteLine("The device is open.")

nRet = dev.MV_CC_SetImageNodeNum_NET(30)
If 0 <> nRet Then
    Console.WriteLine("Set image NodeNum failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")
```

```
End Sub
```

```
End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_SetGrabStrategy_NET

Set the streaming strategy.

API Definition

```
int MV_CC_SetGrabStrategy_NET(
    ref MyCamera.MV_GRAB_STRATEGY enGrabStrategy
);
```

Parameters

enGrabStrategy



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

Return Value

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

Remarks

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

Macro Definition	Description
OneByOne	Get image frames one by one in the chronological order, it is the default strategy.
LatestImagesOnly	Only get the latest one frame from the output buffer list, and clear the rest images in the list.
LatestImages	<p>Get the latest image from the output buffer list, and the quantity of frames depends on the parameter OutputQueueSize, value range: [1,ImageNodeNum]. If the OutputQueueSize values "1", the strategy is same to "LatestImagesOnly", and if the OutputQueueSize values "ImageNodeNum", the strategy is same to "OneByOne".</p> <p> Note</p> <ul style="list-style-type: none">• You can set the OutputQueueSize via API .• You can set the ImageNodeNum via API
UpcomingImage	<p>Ignore all the images in the output buffer list and wait for the next upcoming frame.</p> <p> Note</p> <p>This strategy is supported only by GigE camera.</p>

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;
using MvCamCtrl.NET;
```

```
namespace GrabImage
{
    class GrabStrategies
    {
        public static void UpcomingThread(object obj)
        {
            Thread.Sleep(3000);

            MyCamera device = obj as MyCamera;
            device.MV_CC_SetCommandValue_NET("TriggerSoftware");
        }

        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;           //General device information

                //Print device information
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
                {
                    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

                    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
                    {
                        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                        uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                        uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                        uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                        uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
```

```
        Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("Device Number : " + stUsb3DeviceInfo.chModelName);
    }
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Set trigger mode and trigger source
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerMode", "On");
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Mode failed:{0:x8}", nRet);
    break;
}
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerSource", "Software");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet);
    break;
}

UInt32 nImageNodeNum = 5;
//Set the number of image nodes
nRet = device.MV_CC_SetImageNodeNum_NET(nImageNodeNum);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set number of image node fail:{0:x8}", nRet);
    break;
}

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

Console.WriteLine("*****");

Console.Write("Please Input Grab Strategy:");
UInt32 nGrabStrategy = 0;
try
{
    nGrabStrategy = (UInt32)Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

//U3V device does not support UpcomingImage
if (nGrabStrategy == (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage
    && MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
{
    Console.Write("U3V device not support UpcomingImage\n");
    break;
}

switch(nGrabStrategy)
{
case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne:
{
```



```
        Console.WriteLine("Grab using the MV_GrabStrategy_OneByOne default strategy\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
        break;
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImagesOnly\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
        break;
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImages\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
        break;
    }
    //Set output queue size
    nRet = device.MV_CC_SetOutputQueueSize_NET(2);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
        break;
    }
    break;
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
    }
```

```
        Thread hUpcomingThread = new Thread(UpcomingThread);
        hUpcomingThread.Start(device);
    }
    break;
default:
    Console.WriteLine("Input error!Use default strategy:MV_GrabStrategy_OneByOne\n");
    break;
}

//Start grabbing image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

//Send trigger software command
for (UInt32 i = 0; i < nImageNodeNum; i++)
{
    nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Send Trigger Software command fail:{0:x8}", nRet);
        break;
    }
    Thread.Sleep(500); //Make sure that the trigger software command takes effect and the last frame data has
    been stored in buffer list
}

MyCamera.MV_FRAME_OUT stOutFrame = new MyCamera.MV_FRAME_OUT();
if (nGrabStrategy != (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage)
{
    while(true)
    {
        nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 0);
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth)
+ "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
+ "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);
        }
        else
        {
            break;
        }

        nRet = device.MV_CC_FreeImageBuffer_NET(ref stOutFrame);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
        }
    }
}
```

```
    }
    }
}
else//Only for upcoming
{
    nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 5000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
            "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
            + "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);

        nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}

//Stop grabbing image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
```

```
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Runtime.InteropServices
Imports System.Threading
Imports System.Net.IPAddress
Imports MvCamCtrl.NETModule GrabStrategies
Dim dev As MyCamera = New MyCamera

Sub UpcomingThread()
    Thread.Sleep(3000)

    dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
End Sub

Sub Main()
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    Do While (True)
        ' Enumerate device
        nRet = MyCamera.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If

        ' Print device information
        Dim i As Int32
        For i = 0 To stDeviceInfoList.nDeviceNum - 1
            Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
            stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
```

```
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
        Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
        Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
        stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
        Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
        Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
        Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
        Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

        Console.WriteLine("DEV[" + Convert.ToString(i) + "]: NAME[" + stGigeInfo.chUserDefinedName + "]")
        Console.WriteLine("IP[" + nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." +
nlpByte4.ToString() + "]")
        Console.WriteLine("")
    Else
        Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
        Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
        stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)

        Console.WriteLine("U3V[" + Convert.ToString(i) + "]: NAME[" + stUsbInfo.chUserDefinedName + "]")
        Console.WriteLine("Model[" + stUsbInfo.chSerialNumber + "]")
        Console.WriteLine("")
    End If
Next

Console.WriteLine("please select a device:")
Dim nIndex As Integer
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
```

```
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Set trigger mode and trigger source
nRet = dev.MV_CC_SetEnumValueByString_NET("TriggerMode", "On")
If 0 <> nRet Then
    Console.WriteLine("Set Trigger Mode failed:{0:x8}", nRet)
    Exit Do
End If
nRet = dev.MV_CC_SetEnumValueByString_NET("TriggerSource", "Software")
If 0 <> nRet Then
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet)
    Exit Do
End If

Dim nImageNodeNum As UInt32 = 5
' Set the number of image nodes
nRet = dev.MV_CC_SetImageNodeNum_NET(nImageNodeNum)
If 0 <> nRet Then
    Console.WriteLine("Set number of image node fail:{0:x8}", nRet)
    Exit Do
End If

Console.WriteLine("*****")
Console.WriteLine("0.MV_GrabStrategy_OneByOne;    1.MV_GrabStrategy_LatestImagesOnly;  ")
Console.WriteLine("2.MV_GrabStrategy_LatestImages;  3.MV_GrabStrategy_UpcomingImage;  ")
Console.WriteLine("*****")

Console.WriteLine("Please Input Grab Strategy:")
Dim nGrabStrategy As Int32
Try
    nGrabStrategy = Console.ReadLine()
Catch ex As Exception
```

```
        Console.WriteLine("Invalid input!")
    Exit Do
End Try

If nGrabStrategy = MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage And
MyCamera.MV_USB_DEVICE = stdevInfo.nTLayerType Then
    Console.WriteLine("U3V device not support UpcomingImage")
    Exit Do
End If

Select Case nGrabStrategy
Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne
    Console.WriteLine("Grab using the MV_GrabStrategy_OneByOne default strategy")
    nRet = dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If

Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly
    Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImagesOnly")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If

Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages
    Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImages")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If

    ' Set output queue size
    nRet = dev.MV_CC_SetOutputQueueSize_NET(2)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If

Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage
    Console.WriteLine("Grab using strategy MV_GrabStrategy_UpcomingImage")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If
```

```
Dim hUpcomingThread As New System.Threading.Thread(AddressOf UpcomingThread)
hUpcomingThread.Start()

Case Else
    Console.WriteLine("Input error!Use default strategy:MV_GrabStrategy_OneByOne")
End Select

' Start grabbing image
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet)
    Exit Do
End If

For i = 0 To nImageNodeNum - 1
    nRet = dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
    If 0 <> nRet Then
        Console.WriteLine("Send Trigger Software command fail:{0:x8}", nRet)
        Exit Do
    End If
    Threading.Thread.Sleep(500)
Next

Dim stOutFrame As MyCamera.MV_FRAME_OUT = New MyCamera.MV_FRAME_OUT
If nGrabStrategy <> MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage Then
    While (True)
        nRet = dev.MV_CC_GetImageBuffer_NET(stOutFrame, 0)
        If 0 = nRet Then
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
                "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight) + "], FrameNum[" +
                Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "])")
        Else
            Exit While
        End If

        nRet = dev.MV_CC_FreelImageBuffer_NET(stOutFrame)
        If 0 <> nRet Then
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet)
        End If
    End While
Else
    nRet = dev.MV_CC_GetImageBuffer_NET(stOutFrame, 5000)
    If 0 = nRet Then
        Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
            "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight) + "], FrameNum[" +
            Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "])")

        nRet = dev.MV_CC_FreelImageBuffer_NET(stOutFrame)
        If 0 <> nRet Then
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet)
        End If
    End If
End If
```



```
Else
    Console.WriteLine("No data:{0:x8}", nRet)
End If
End If

' Stop grabbing image
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed:{0:x8}", nRet)
Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CC_SetOutputQueueSize_NET

Set the output queue size.

API Definition

```
int MV_CC_SetOutputQueueSize_NET(
    uint   nOutputQueueSize
);
```

Parameters

nOutputQueueSize

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

Return Value

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

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;
using MvCamCtrl.NET;
namespace GrabImage
{
    class GrabStrategies
    {
        public static void UpcomingThread(object obj)
        {
            Thread.Sleep(3000);

            MyCamera device = obj as MyCamera;
            device.MV_CC_SetCommandValue_NET("TriggerSoftware");
        }

        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
```

```
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;          //General device information

    //Print device information
    for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
    {
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
            (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
            typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
            stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
            (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
            typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
            stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("Device Number : " + stUsb3DeviceInfo.chModelName);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
    try
    {
        nDevIndex = Convert.ToInt32(Console.ReadLine());
    }
    catch
```

```
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Set trigger mode and trigger source
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerMode", "On");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Mode failed:{0:x8}", nRet);
    break;
}
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerSource", "Software");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet);
    break;
}

UInt32 nImageNodeNum = 5;
//Set the number of image nodes
nRet = device.MV_CC_SetImageNodeNum_NET(nImageNodeNum);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set number of image node fail:{0:x8}", nRet);
    break;
}
```

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

Console.WriteLine("*****");

    Console.Write("Please Input Grab Strategy:");
    UInt32 nGrabStrategy = 0;
    try
    {
        nGrabStrategy = (UInt32)Convert.ToInt32(Console.ReadLine());
    }
    catch
    {
        Console.Write("Invalid Input!\n");
        break;
    }

    //U3V device does not support UpcomingImage
    if (nGrabStrategy == (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage
        && MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        Console.Write("U3V device not support UpcomingImage\n");
        break;
    }

    switch(nGrabStrategy)
    {
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne:
        {
            Console.Write("Grab using the MV_GrabStrategy_OneByOne default strategy\n");
            nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
                break;
            }
            break;
        case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly:
            {
                Console.Write("Grab using strategy MV_GrabStrategy_LatestImagesOnly\n");
                nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
                    break;
                }
            }
        }
    }
```

```
        break;
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImages\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }

        //Set output queue size
        nRet = device.MV_CC_SetOutputQueueSize_NET(2);

        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
    }
    break;
    case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }

        Thread hUpcomingThread = new Thread(UpcomingThread);
        hUpcomingThread.Start(device);
    }
    break;
default:
    Console.WriteLine("Input error!Use default strategy:MV_GrabStrategy_OneByOne\n");
    break;
}

//Start grabbing image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

//Send trigger software command
for (UInt32 i = 0; i < nImageNodeNum; i++)
```

```
{
    nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Send Trigger Software command fail:{0:x8}", nRet);
        break;
    }
    Thread.Sleep(500); // Make sure that the trigger software command takes effect and the last frame data has
    been stored in buffer list
}

MyCamera.MV_FRAME_OUT stOutFrame = new MyCamera.MV_FRAME_OUT();
if (nGrabStrategy != (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage)
{
    while(true)
    {
        nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 0);
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth)
+ "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
+ "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);
        }
        else
        {
            break;
        }

        nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
        }
    }
}
else // Only for upcoming
{
    nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 5000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
        "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);

        nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
        }
    }
    else
    {

```

```
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}

//Stop grabbing image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```


Module GrabStrategies

```
Dim dev As MyCamera = New MyCamera

Sub UpcomingThread()
    Thread.Sleep(3000)

    dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
End Sub

Sub Main()
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    Do While (True)
        ' Enumerate device
        nRet = MyCamera.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If

        ' Print device information
        Dim i As Int32
        For i = 0 To stDeviceInfoList.nDeviceNum - 1
            Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
            stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
            If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
                Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
                stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
                Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
                Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
                Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
                Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

                Console.WriteLine("DEV[" + Convert.ToString(i) + "]: NAME[" + stGigeInfo.chUserDefinedName + "]")
                Console.WriteLine("IP[" + nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." +
nlpByte4.ToString() + "]")
                Console.WriteLine("")
            Else
                Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
```

```
Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)

    Console.WriteLine("U3V[" + Convert.ToString(i) + "]: NAME[" + stUsbInfo.chUserDefinedName + "]")
    Console.WriteLine("Model[" + stUsbInfo.chSerialNumber + "]")
    Console.WriteLine("")
End If
Next

Console.Write("please select a device:")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If
```

```
' Set trigger mode and trigger source
nRet = dev.MV_CC_SetEnumValueByString_NET("TriggerMode", "On")
If 0 <> nRet Then
    Console.WriteLine("Set Trigger Mode failed:{0:x8}", nRet)
    Exit Do
End If
nRet = dev.MV_CC_SetEnumValueByString_NET("TriggerSource", "Software")
If 0 <> nRet Then
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet)
    Exit Do
End If

Dim nImageNodeNum As UInt32 = 5
' Set number of image nodes
nRet = dev.MV_CC_SetImageNodeNum_NET(nImageNodeNum)
If 0 <> nRet Then
    Console.WriteLine("Set number of image node fail:{0:x8}", nRet)
    Exit Do
End If

Console.WriteLine("*****")
Console.WriteLine("0.MV_GrabStrategy_OneByOne; 1.MV_GrabStrategy_LatestImagesOnly; ")
Console.WriteLine("2.MV_GrabStrategy_LatestImages; 3.MV_GrabStrategy_UpcomingImage; ")

Console.WriteLine("*****")

Console.WriteLine("Please Input Grab Strategy:")
Dim nGrabStrategy As Int32
Try
    nGrabStrategy = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Exit Do
End Try

If nGrabStrategy = MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage And
MyCamera.MV_USB_DEVICE = stdevInfo.nTLayerType Then
    Console.WriteLine("U3V device not support UpcomingImage")
    Exit Do
End If

Select Case nGrabStrategy
Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne
    Console.WriteLine("Grab using the MV_GrabStrategy_OneByOne default strategy")
    nRet = dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
        Exit Do
    End If
```

```
Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly
    Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImagesOnly")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
    Exit Do
End If

Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages
    Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImages")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
    Exit Do
End If

' Set output queue size
nRet = dev.MV_CC_SetOutputQueueSize_NET(2)
If 0 <> nRet Then
    Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
Exit Do
End If

Case MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage
    Console.WriteLine("Grab using strategy MV_GrabStrategy_UpcomingImage")
    nRet =
dev.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage)
    If 0 <> nRet Then
        Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet)
    Exit Do
End If

Dim hUpcomingThread As New System.Threading.Thread(AddressOf UpcomingThread)
hUpcomingThread.Start()

Case Else
    Console.WriteLine("Input error!Use default strategy:MV_GrabStrategy_OneByOne")
End Select

' Start grabbing image
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet)
Exit Do
End If

For i = 0 To nImageNodeNum - 1
    nRet = dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
    If 0 <> nRet Then
        Console.WriteLine("Send Trigger Software command fail:{0:x8}", nRet)
```

```
Exit Do
End If
Threading.Thread.Sleep(500)
Next

Dim stOutFrame As MyCamera.MV_FRAME_OUT = New MyCamera.MV_FRAME_OUT
If nGrabStrategy <> MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage Then
    While (True)
        nRet = dev.MV_CC_GetImageBuffer_NET(stOutFrame, 0)
        If 0 = nRet Then
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
                "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight) + "], FrameNum[" +
                Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "])"
        Else
            Exit While
        End If

        nRet = dev.MV_CC_FreeImageBuffer_NET(stOutFrame)
        If 0 <> nRet Then
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet)
        End If
    End While
Else
    nRet = dev.MV_CC_GetImageBuffer_NET(stOutFrame, 5000)
    If 0 = nRet Then
        Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
            "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight) + "], FrameNum[" +
            Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "])"

        nRet = dev.MV_CC_FreeImageBuffer_NET(stOutFrame)
        If 0 <> nRet Then
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet)
        End If
    Else
        Console.WriteLine("No data:{0:x8}", nRet)
    End If
End If

' Stop grabbing image
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed:{0:x8}", nRet)
Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If
```

```
' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.3.2 GigE APIs

MvCamCtrl.NET::MyCamera::MV_CC_GetOptimalPacketSize_NET

Get the optimal size of packet.

API Definition

```
int MV_CC_GetOptimalPacketSize_NET(
);
```

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

- For GigEVision device, the optimal packet size is SCPS, and for USB3Vision device, the optimal packet size is that read from drive.
- This API should be called after calling **MvCamCtrl.NET::MyCamera::MV_CC_OpenDevice_NET** and before calling **MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET** .
- This API is not supported by CameraLink device.

MvCamCtrl.NET::MyCamera::MV_GIGE_ForceIpEx_NET

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

API Definition

```
int MV_GIGE_ForceIpEx_NET(  
    uint    nIP,  
    uint    nSubNetMask,  
    uint    nDefaultGateWay  
);
```

Parameters

nIP

[IN] Configured IP address

nSubNetMask

[IN] Subnet mask

nDefaultGateWay

[IN] Default gateway

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- This function is supported only by GigEVision cameras.
- If the device is in DHCP status, the device will reboot after calling this API to force setting the network parameters.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace ForceIp  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
```

```
int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

uint nIP = 0x0a0f0536;
uint nSubNetMask = 0xffffffff;
uint nDefaultGateWay = 0x0a0f05fe;
nRet = device.MV_GIGE_ForceIpEx_NET(nIP, nSubNetMask, nDefaultGateWay);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Force IP failed:{0:x8}", nRet);
    return;
}

//Other process...

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```


Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Creating device handle failed.")
        End If
        Console.WriteLine("The device handle is created.")

        Dim nIP As UInt64 = &HA0F0536
        Dim nSubNetMask As UInt64 = &HFFFFFF0
        Dim nDefaultGateWay As UInt64 = &HA0F05FE
        nRet = dev.MV_GIGE_ForceIpEx_NET(nIP, nSubNetMask, nDefaultGateWay)

        '//Other process...

        'Destroy handle
        nRet = dev.MV_CC_DestroyDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Destroying handle failed.")
        End If
    End Sub
End Module
```

```
Console.WriteLine("The handle is destroyed.")
```

```
End Sub
```

```
End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_GetGvspTimeout_NET

Get GVSP streaming timeout.

API Definition

```
public Int32 MV_GIGE_getGvspTimeout_NET(  
    UInt32      pnMillilsec  
);
```

Parameters

pnMillilsec

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

Return Value

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

MvCamCtrl.NET::MyCamera::MV_GIGE_SetGvspTimeout_NET

Set GVSP streaming timeout.

API Definition

```
public Int32 MV_GIGE_SetGvspTimeout_NET(  
    unsigned int      nMilLilsec  
);
```

Parameters

nMilLilsec

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

Return Value

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

MvCamCtrl.NET::MyCamera::MV_GIGE_GetResendMaxRetryTimes_NET

Get the maximum times one packet can be resent.

API Definition

```
Public int32 MV_GIGE_GetResendMaxRetryTimes_NET(  
    ref UInt32    pnRetryTimes  
);
```

Parameters

pnRetryTimes

The maximum times one packet can be resent.

Return Value

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

Remarks

You should call this API after enabling the function of resending packets by calling **MvCamCtrl.NET::MyCamera::MV_GIGE_SetResend_NET**.

MvCamCtrl.NET::MyCamera::MV_GIGE_SetResendMaxRetryTimes_NET

Set the maximum times one packet can be resent.

API Definition

```
public Int32 MV_GIGE_SetResendMaxRetryTimes_NET(  
    UInt32    nRetryTimes  
);
```

Parameters

nRetryTimes

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

Return Value

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

Remarks

You should call this API after enabling the function of resending packets by calling **MvCamCtrl.NET::MyCamera::MV_GIGE_SetResend_NET**.

MvCamCtrl.NET::MyCamera::MV_GIGE_GetResendTimeInterval_NET

Get the packet resending interval.

API Definition

```
public Int32 MV_GIGE_GetResendTimeInterval_NET(
    UInt32    pnMillilsec
);
```

Parameters

pnMillilsec

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

Return Value

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

Remarks

You should call this API after enabling the function of resending packets by calling **MvCamCtrl.NET::MyCamera::MV_GIGE_SetResend_NET**.

MvCamCtrl.NET::MyCamera::MV_GIGE_SetResendTimeInterval_NET

Set the packet resending interval.

API Definition

```
public Int32 MV_GIGE_SetResendTimeInterval_NET(
    UInt32    nMillilsec
);
```

Parameters

nMillilsec

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

Return Value

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

Remarks

You should call this API after enabling the function of resending packets by calling **MvCamCtrl.NET::MyCamera::MV_GIGE_SetResend_NET**.

MvCamCtrl.NET::MyCamera::MV_GIGE_SetTransmissionType_NET

Set transmission mode.

API Definition

```
int MV_GIGE_SetTransmissionType_NET(  
    ref MV_TRANSMISSION_TYPE_NET nTransmissionType  
);
```

Parameters

pstTransmissionType

Transmission mode, see the structure ***MV_TRANSMISSION_TYPE_NET*** for details.

Return Value

Return *MyCamera.MV_OK (0)* on success; 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 only valid for GigEVision camera.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Param  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
  
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
```

```
//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

uint nIp = 0xef000117;//239.0.1.23
uint nTransmissionType = (uint)MyCamera.MV_GIGE_TRANSMISSION_TYPE.MV_GIGE_TRANSTYPE_MULTICAST;
nRet = device.MV_GIGE_SetTransmissionType_NET(ref stTransmission);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Transmission Type fail! nRet [%x]\n", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Creating device failed.")
        End If
        Console.WriteLine("The device is created.")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Opening device failed.")
        End If
        Console.WriteLine("Opening device succeed.")

        Dim stTransmission As MyCamera.MV_CC_TRANSMISSION_TYPE = New
MyCamera.MV_CC_TRANSMISSION_TYPE
        stTransmission.nDestPort = 8787
        stTransmission.nDestIp = 0xef000117
        stTransmissionType.enTransmission =
```

```
(uint)MyCamera.MV_GIGE_TRANSMISSION_TYPE.MV_GIGE_TRANSTYPE_MULTICAST
    nRet = dev.MV_GIGE_SetTransmissionType_NET(stTransmission)

    If 0 <> nRet Then
        Console.WriteLine("Set Transmission Type failed")
    End If

    //Other process...

    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Closing device failed.")
    End If
    Console.WriteLine("The device is closed.")

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroying handle failed.")
    End If
    Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_GetMulticastStatus_NET

Get the device multicast status.

API Definition

```
Int32 MV_GIGE_GetMulticastStatus_NET(
    ref MV_CC_DEVICE_INFO    pstDevInfo
    Boolean                  pStatus
)
```

Parameters

pstDevInfo

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

pStatus

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

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

Example

C#

```
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE, ref
stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo =
(MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));
Boolean bState = false;
nRet = MyCamera.MV_GIGE_GetMulticastStatus_NET(ref stDevInfo, ref bState);
```

Example

VB

```
Dim stDevList As MyCamera.MV_CC_DEVICE_INFO_LIST = new MyCamera.MV_CC_DEVICE_INFO_LIST
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE, stDevList)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Enum device failed:{0:x8}", nRet)
    break
End If

Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum))
if (0 == stDevList.nDeviceNum) Then
    break;
End If

Dim stDevInfo As MyCamera.MV_CC_DEVICE_INFO = new MyCamera.MV_CC_DEVICE_INFO
Dim bState As bool= false;
nRet = MyCamera.MV_GIGE_GetMulticastStatus_NET(stDevInfo, bState)
```

MvCamCtrl.NET::MyCamera::MV_GIGE_GetNetTransInfo_NET

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

API Definition

```
int MV_GIGE_GetNetTransInfo_NET(  
    ref MyCamera.MV_NETTRANS_INFO  pstInfo  
);
```

Parameters

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 *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

- This API should be called after calling ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition.
- This API is supported only by GigEVision camera.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace NetTransInfo  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enum device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
        }  
    }  
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}

//Get Payload Size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
uint nBufSize = stParam.nCurValue;

IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    //Get one frame
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, ref FrameInfo, 1000);
    if (MyCamera.MV_OK == nRet)
    {

```

```
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" +
Convert.ToString(FrameInfo.nFrameNum));

        if ((nCount % 10) == 0)
        {
            MyCamera.MV_NETTRANS_INFO NetTransInfo = new MyCamera.MV_NETTRANS_INFO();
            nRet = device.MV_GIGE_GetNetTransInfo_NET(ref NetTransInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Get Net NetTrans info failed:{0:x8}", nRet);
                return;
            }
        }
        else
        {
            Console.WriteLine("No data:{0:x8}", nRet);
        }
    }
    Marshal.FreeHGlobal(pBufForDriver);

    //Other process...

    //Stop acquisition
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        return;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        'Start getting stream
        nRet = dev.MV_CC_StartGrabbing_NET()
        If 0 <> nRet Then
            Console.WriteLine("Start grabbing failed!")
        End If
        Console.WriteLine("Start grabbing succeed!")
    End Sub
End Module
```

'Image acquisition

'Get Payload Size

```
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
```

```
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
```

```
If (MyCamera.MV_OK <> nRet) Then
```

```
Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
```

```
Exit Do
```

```
End If
```

```
Dim nPayloadSize As Int32 = stParam.nCurValue
```

```
Dim pBufForDriver As IntPtr = Marshal.AllocHGlobal(nBufSize)
```

```
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
```

```
Do While nCount <> 10
```

```
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, FrameInfo, 1000)
```

```
    If MyCamera.MV_OK = nRet Then
```

```
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
```

```
Convert.ToString(FrameInfo.nHeight) + "FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
```

```
        If nCount Mod 10 = 0 Then
```

```
            Dim NetTransInfo As MyCamera.MV_NETTRANS_INFO = New MyCamera.MV_NETTRANS_INFO()
```

```
            nRet = dev.MV_GIGE_GetNetTransInfo_NET(NetTransInfo)
```

```
            If MyCamera.MV_OK <> nRet Then
```

```
                Console.WriteLine("Convert PixelType Failed:{0:x8}", nRet)
```

```
                Return
```

```
            End If
```

```
        End If
```

```
    Else
```

```
        Console.WriteLine("No data:{0:x8}", nRet)
```

```
    End If
```

```
Loop
```

```
Marshal.FreeHGlobal(pBufForDriver)
```

```
//Other process...
```

'Stop getting stream

```
nRet = dev.MV_CC_StopGrabbing_NET()
```

```
If 0 <> nRet Then
```

```
    Console.WriteLine("Stop grabbing failed!")
```

```
End If
```

```
Console.WriteLine("Start grabbing succeed!")
```

'Close camera

```
nRet = dev.MV_CC_CloseDevice_NET()
```

```
If 0 <> nRet Then
```

```
    Console.WriteLine("Close device failed!")
```

```
End If
```

```
Console.WriteLine("Close device succeed!")
```

'Destroy handle

```
nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
```

```
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")
```

```
End Sub
```

```
End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_IssueActionCommand_NET

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

API Definition

```
int MV_GIGE_IssueActionCommand_NET(
    ref MyCamera.MV_ACTION_CMD_INFO  pstActionCmdInfo,
    ref MyCamera.MV_ACTION_CMD_RESULT_LIST  pstActionCmdResults
);
```

Parameters

pstActionCmdInfo

[OUT] 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 *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

This API is supported only by GigEVision camera.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;
namespace Grab_ActionCommand
{
    class Grab_ActionCommand
```

```
{
    static bool g_bExit = false;
    static uint g_DeviceKey = 1;
    static uint g_GroupKey = 1;
    static uint g_GroupMask = 1;
    static uint g_nPayloadSize = 0;
    public static void ActionCommandWorkThread(object obj)
    {
        MyCamera device = obj as MyCamera;
        int nRet = MyCamera.MV_OK;
        MyCamera.MV_ACTION_CMD_INFO stActionCmdInfo = new MyCamera.MV_ACTION_CMD_INFO();
        MyCamera.MV_ACTION_CMD_RESULT_LIST stActionCmdResults = new
MyCamera.MV_ACTION_CMD_RESULT_LIST();
        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)
        {
            MyCamera.MV_ACTION_CMD_RESULT pResults = new MyCamera.MV_ACTION_CMD_RESULT();
            int size = Marshal.SizeOf(pResults);
            stActionCmdResults.pResults = Marshal.AllocHGlobal(size);
            Marshal.StructureToPtr(pResults, stActionCmdResults.pResults, false);
            nRet = device.MV_GIGE_IssueActionCommand_NET(ref stActionCmdInfo, ref stActionCmdResults);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Issue Action Command failed! nRet {0:x8}", nRet);
            }
            unsafe
            {
                MyCamera.MV_ACTION_CMD_RESULT stTempActionCmd = new MyCamera.MV_ACTION_CMD_RESULT();
                var len = Marshal.SizeOf(stTempActionCmd) * stActionCmdResults.nNumResults;
                byte* srcPtr = (byte*)stActionCmdResults.pResults.ToPointer();
                var targetPtr = Marshal.AllocHGlobal((int)len);
                byte* tmpPtr = (byte*)targetPtr.ToPointer();
                for (int i = 0; i < len; i++)
                {
                    *(tmpPtr + i) = *(srcPtr + i);
                }
                MyCamera.MV_ACTION_CMD_RESULT[] arrayMvActionCmdResult =
PtrToStructurs<MyCamera.MV_ACTION_CMD_RESULT>(targetPtr, (int)stActionCmdResults.nNumResults);
                Marshal.FreeHGlobal(targetPtr);
                for (uint i = 0; i < stActionCmdResults.nNumResults; i++)
                {
                    Console.WriteLine("Ip == " + arrayMvActionCmdResult[i].strDeviceAddress + ", Status == " +
Convert.ToString(arrayMvActionCmdResult[i].nStatus));
                }
            }
        }
    }
}
```



```
public unsafe static T[] PtrToStructurs<T>(IntPtr pt, int lenth)
{
    T[] structur = new T[lenth];
    for (int i = 0; i < lenth; i++)
    {
        IntPtr ptr = new IntPtr((int)pt + (i * Marshal.SizeOf(typeof(T))));
        structur[i] = (T)Marshal.PtrToStructure(ptr, typeof(T));
    }
    return structur;
}

public static void ReceivelImageWorkThread(object obj)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = obj as MyCamera;
    MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
    IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);
    if (pData == IntPtr.Zero)
    {
        return;
    }
    uint nDataSize = g_nPayloadSize;
    while (true)
    {
        nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
        if (nRet == MyCamera.MV_OK)
        {
            g_bExit = true;
            Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
            + "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);
        }
        else
        {
            Console.WriteLine("No data:{0:x8}", nRet);
        }
        if (g_bExit)
        {
            break;
        }
    }
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    do
    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo; // General information of device
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
if (m_stDevList.nDeviceNum == 0)
{
    printf("no camera found!\n");
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detect optimal packet size (it only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
}
```

```
}
else
{
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
}
}

//Set Action Device Key
nRet = device.MV_CC_SetIntValue_NET("ActionDeviceKey", g_DeviceKey);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Action Device Key failed! {0:x8}", nRet);
    break;
}

//Set Action Group Key
nRet = device.MV_CC_SetIntValue_NET("ActionGroupKey", g_GroupKey);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Action Group Key failed! {0:x8}", nRet);
    break;
}

//Set Action Group Mask
nRet = device.MV_CC_SetIntValue_NET("ActionGroupMask", g_GroupMask);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Action Group Mask fail! {0:x8}", nRet);
    break;
}

//Get packet size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
g_nPayloadSize = stParam.nCurValue;

//Start image acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}
Thread hActionCommandThreadHandle = new Thread(ActionCommandWorkThread);
hActionCommandThreadHandle.Start(device);
Thread hReceiveImageThreadHandle = new Thread(ReceiveImageWorkThread);
hReceiveImageThreadHandle.Start(device);
```

```
Console.WriteLine("Press enter to exit");
Console.ReadKey();
g_bExit = true;
Thread.Sleep(1000);

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resource
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module GrabImage

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim g_DeviceKey As UInt32 = 1
    Dim g_GroupKey As UInt32 = 1
    Dim g_GroupMask As UInt32 = 1
    Do While (True)

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If
        If (0 = m_stDeviceInfoList.nDeviceNum)
        Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed:{0:x8}", nRet)
            Exit Do
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Open device
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed:{0:x8}", nRet)
            Exit Do
        End If

        'Get optimal packet size (It only works for the GigE camera)
        If stdevInfo.nTLayerType = MyCamera.MV_GIGE_DEVICE Then
            Dim nPacketSize As Int32
            nPacketSize = dev.MV_CC_GetOptimalPacketSize_NET()
            If nPacketSize > 0 Then
                nRet = dev.MV_CC_SetIntValue_NET("GevSCSPPacketSize", nPacketSize)
                If 0 <> nRet Then
```

```
        Console.WriteLine("Warning: Set Packet Size failed:{0:x8}", nRet)
    End If
    Else
        Console.WriteLine("Warning: Get Packet Size failed:{0:x8}", nPacketSize)
    End If
End If

'Start image acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If
Dim stActionCmdInfo As MyCamera.MV_ACTION_CMD_INFO = New MyCamera.MV_ACTION_CMD_INFO()
Dim stActionCmdResults As MyCamera.MV_ACTION_CMD_RESULT_LIST = New
MyCamera.MV_ACTION_CMD_RESULT_LIST()
stActionCmdInfo.nDeviceKey = g_DeviceKey
stActionCmdInfo.nGroupKey = g_GroupKey
stActionCmdInfo.nGroupMask = g_GroupMask
stActionCmdInfo.pBroadcastAddress = "255.255.255.255"
stActionCmdInfo.nTimeOut = 100
stActionCmdInfo.bActionTimeEnable = 0
Dim pResults As MyCamera.MV_ACTION_CMD_RESULT = New MyCamera.MV_ACTION_CMD_RESULT()
Dim size As Int32 = Marshal.SizeOf(pResults)
stActionCmdResults.pResults = Marshal.AllocHGlobal(size)
Marshal.StructureToPtr(pResults, stActionCmdResults.pResults, False)
nRet = dev.MV_GIGE_IssueActionCommand_NET(stActionCmdInfo, stActionCmdResults)
If 0 <> nRet Then
    Console.WriteLine("Issue Action Command failed! nRet {0:x8}", nRet);
    Exit Do
End If

'Get packet size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
Dim m_bytelImageBuffer(1024 * 1024) As Byte
Dim m_bytelImageBufferLen As Int32 = 1024 * 1024

'Get one frame
Dim nCount As Int32 = 0
Do While nCount <> 10
    nCount = nCount + 1
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufferForDriver, nPayloadSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
```

```
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Dim stSaveParam As MyCamera.MV_SAVE_IMAGE_PARAM_EX = New
MyCamera.MV_SAVE_IMAGE_PARAM_EX()
Else
    Console.WriteLine("Get one frame failed:{0:x8}", nRet)
End If
Loop
Marshal.FreeHGlobal(pBufForDriver)

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

'Shut device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_SetNetTransMode_NET

Set the prior network mode.

API Definition

```
int MV_GIGE_SetNetTransMode_NET(  
    uint    nType  
);
```

Parameters

nType

[IN] Network mode, see details in the following table:

Definition	Value	Description
MyCamera.MV_NET_TRANS_DRIVER	0x00000001	Drive mode
MyCamera.MV_NET_TRANS_SOCKET	0x00000002	Socket Mode

Return Value

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

Remarks

By default, the network mode is Drive Mode, and this API is only supported by GigEVision camera.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace NetTransMode  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
        }  
    }  
}
```



```
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

uint nType = MyCamera.MV_NET_TRANS_SOCKET;//Socket Mode
nRet = device.MV_GIGE_SetNetTransMode_NET(nType);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Net TransMode failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
```

```
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading.Thread  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET
```

Module Module1

Sub Main()

Dim dev As MyCamera = New MyCamera

Dim Info As String

Dim nRet As Int32 = MyCamera.MV_OK

Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

'Enumerate devices

nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE), stDeviceInfoList)

If MyCamera.MV_OK <> nRet Then

Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))

Return

End If

If (0 = stDeviceInfoList.nDeviceNum) Then

Console.WriteLine("No GigE device found.")

Return

End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO

stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),

GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle

nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)

If 0 <> nRet Then

Console.WriteLine("Creating device handle failed.")

End If

Console.WriteLine("The device handle is created.")

'Open camera

nRet = dev.MV_CC_OpenDevice_NET()

If 0 <> nRet Then

Console.WriteLine("Opening camera failed.")

End If

Console.WriteLine("The camera is open.")

Dim nType As Int32 = MyCamera.MV_NET_TRANS_SOCKET

'Socket Mode

nRet = dev.MV_GIGE_SetNetTransMode_NET(nType)

```
If 0 <> nRet Then
    Console.WriteLine("Set Net TransMode failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_SetIpConfig_NET

Set IP address configuration mode.

API Definition

```
int MV_GIGE_SetIpConfig_NET(
    uint    nType
);
```

Parameters

nType

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

Definition	Value	Description
MyCamera.MV_IP_CFG_STATIC	0x05000000	Fixed IP Address Mode
MyCamera.MV_IP_CFG_DHCP	0x06000000	DHCP Auto Getting IP Address Mode
MyCamera.MV_IP_CFG_LLA	0x04000000	LLA(Link-local address), link local address

Return Value

Return *MyCamera.MV_OK (0)* on success; 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

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace SetIPConfig
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
            }
        }
    }
}
```

```
        return;
    }

    uint nType = MyCamera.MV_IP_CFG_STATIC; //fixed IP address mode
    nRet = device.MV_GIGE_SetIpConfig_NET(nType);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set IP Config failed:{0:x8}", nRet);
        return;
    }

    //...other processing

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Module1

```
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find ABC Gige | Usb Device !")
            Return
        End If
    End Sub
End Module
```

```
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

Dim nType As UInt32 = MyCamera.MV_IP_CFG_STATIC 'fixed IP address mode
nRet = dev.MV_GIGE_SetIpConfig_NET(nType)
If 0 <> nRet Then
    Console.WriteLine("Set IP Config failed")
End If

//...other processing

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_GIGE_SetResend_NET

Set parameters of resending packets.

API Definition

```
int MV_GIGE_SetResend_NET(
    uint    bEnable,
    uint    nMaxResendPercent,
    uint    nResendTimeout
);
```

Parameters

bEnable

[IN] Enable resending packet: 0-Disable, 1-Enable

nMaxResendPercent

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

nResendTimeout

[IN] Packet resending timeout, unit: millisecond

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

This API should be called after connecting to device, and it is only supported by GigEVision camera.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace SetResend
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
```

```
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_GIGE_SetResend_NET(1,1,100);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Resend failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

```
Module Module1
```



```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    'Enumerate devices
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE), stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enumerating device failed."+ Convert.ToString(nRet))
        Return
    End If

    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No GigE device found.")
        Return
    End If

    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
    GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

    'Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Creating device handle failed.")
    End If
    Console.WriteLine("The device handle is created.")

    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Opening camera failed.")
    End If
    Console.WriteLine("The camera is open.")

    nRet = dev.MV_GIGE_SetResend_NET(1, 1, 100)
    If 0 <> nRet Then
        Console.WriteLine("Set Resend failed")
    End If

    '//Other process...

    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Closing camera failed.")
    End If
    Console.WriteLine("The camera is closed.")

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

4.3.3 CameraLink Camera

MvCamCtrl.NET::MyCamera::MV_CAML_GetDeviceBauderate_NET

Get device baud rate.

API Definition

```
int MV_CAML_GetDeviceBauderate_NET(
    ref uint    pnCurrentBaudrate
);
```

Parameters

pnCurrentBaudrate

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

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

Return Values

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

Remarks

This API is supported only by CameraLink device.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace CamLBasicDemo
{
    class CamLBasicDemo
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            bool bDevConnected = false; //whether a device is connected
            do
            {
                //Enumerate device

                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_CAMERALINK_DEVICE, ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }
                MyCamera.MV_CC_DEVICE_INFO stDevInfo;
                stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
                typeof(MyCamera.MV_CC_DEVICE_INFO));
                if (m_stDevList.nDeviceNum == 0)
                {
                    Console.WriteLine("no camera found!\n");
                    return;
                }

                //Create device
                nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
                if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Get device baud rate
uint nCurrentBaudrate = 0;
nRet = device.MV_CAML_GetDeviceBauderate_NET(ref nCurrentBaudrate);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get device bauderate fail:{0:x8}", nRet);
    break;
}
Console.WriteLine("Current device bauderate:{0:x8}", nCurrentBaudrate);

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Open finish.");
bDevConnected = true;

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
bDevConnected = false;

//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("\n Close finish.");
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Make sure the device is shutted down
    if ( bDevConnected )
    {
        device.MV_CC_CloseDevice_NET();
        bDevConnected = false;
    }
}
```

```
        //Destroy handle
        device.MV_CC_DestroyDevice_NET();
    }
    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module GrabImage

Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        If (0 = m_stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed:{0:x8}", nRet)
            Exit Do
        End If

        'Open device
```

```
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Get device baud rate
Dim nCurrentBaudrate As UInteger
nCurrentBaudrate = 0
nRet = dev.MV_CAML_GetDeviceBauderate_NET(nCurrentBaudrate)
If 0 <> nRet Then
    Console.WriteLine("Get device bauderate fail:{0:x8}", nRet)
    Exit Do
End If
Console.WriteLine("Current device bauderate:{0:x8}", nCurrentBaudrate)

'Shut device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle and release resource
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CAML_GetSupportBauderates_NET

Get supported baud rate for connecting device and host.

API Definition

```
int MV_CAML_GetSupportBaudrates_NET(  
    ref uint    pnBaudrateAblity  
);
```

Parameters

pnBaudrateAblity

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

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

Return Value

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

Remarks

This API is supported only by CameraLink device.

Example

C#

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace CamLBasicDemo  
{  
    class CamLBasicDemo  
    {
```

```
static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    bool bDevConnected = false; //whether a device is connected
    do
    {
        //Enumerate device

        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_CAMERA_LINK_DEVICE, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }
        MyCamera.MV_CC_DEVICE_INFO stDevInfo;
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
        typeof(MyCamera.MV_CC_DEVICE_INFO));
        if (m_stDevList.nDeviceNum == 0)
        {
            Console.WriteLine("no camera found!\n");
            return;
        }

        // Create device
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Create device failed:{0:x8}", nRet);
            break;
        }

        /*****The following content is valid only for CameraLink device*****/
        //Get supported baud rate of connecting device and host
        uint nBaudrateAblity = 0;
        nRet = device.MV_CAML_GetSupportBauderates_NET(ref nBaudrateAblity);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Get supported bauderate fail:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Current device supported bauderate:{0:x8}", nBaudrateAblity);

        //Open device
        nRet = device.MV_CC_OpenDevice_NET();
        if (MyCamera.MV_OK != nRet)
```



```
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Open finish.");
bDevConnected = true;

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
bDevConnected = false;

//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("\n Close finish.");
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Make sure the device is shutted down
    if ( bDevConnected )
    {
        device.MV_CC_CloseDevice_NET();
        bDevConnected = false;
    }

    //Destroy handle and release resource
    device.MV_CC_DestroyDevice_NET();
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module GrabImage
```

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        ' Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        If (0 = m_stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed:{0:x8}", nRet)
            Exit Do
        End If

        'Open device
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed:{0:x8}", nRet)
            Exit Do
        End If

        '*****The following content is valid only for CameraLink device*****/
        'Get supported baud rate of connecting device and host
        Dim nBaudrateAblity As UInteger
        nBaudrateAblity = 0
        nRet = dev.MV_CAML_GetSupportBauderates_NET(nBaudrateAblity)
        If 0 <> nRet Then
            Console.WriteLine("Get supported bauderate fail:{0:x8}", nRet)
            Exit Do
        End If
        Console.WriteLine("Current device supported bauderate:{0:x8}", nBaudrateAblity)
    End Do
End Sub
```

```
'Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

MvCamCtrl.NET::MyCamera::MV_CAML_SetDeviceBauderate_NET

Set device baud rate.

API Definition

```
int MV_CAML_SetDeviceBauderate_NET(
    uint      nBaudrate
);
```

Parameters

nBaudrate

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

Macro Definition	Value	Description
MV_CAML_BAUDRATE_9600	0x00000001	9600 baud rate
MV_CAML_BAUDRATE_19200	0x00000002	19200 baud rate
MV_CAML_BAUDRATE_38400	0x00000004	38400 baud rate

Macro Definition	Value	Description
MV_CAML_BAUDRATE_57600	0x00000008	57600 baud rate
MV_CAML_BAUDRATE_115200	0x00000010	115200 baud rate
MV_CAML_BAUDRATE_230400	0x00000020	230400 baud rate
MV_CAML_BAUDRATE_460800	0x00000040	460800 baud rate
MV_CAML_BAUDRATE_921600	0x00000080	921600 baud rate
MV_CAML_BAUDRATE_AUTOMAX	0x40000000	The maximum self-adaptive baud rate

Return Value

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

Remarks

This API is supported only by CameraLink device.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace CamLBasicDemo
{
    class CamLBasicDemo
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            bool bDevConnected = false; //whether a device is connected
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_CAMERALINK_DEVICE, ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
```

```
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo;
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Set device baud rate
nRet = device.MV_CAML_SetDeviceBauderate_NET((uint)MyCamera.MV_CAML_BAUDRATE_115200);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set device bauderate fail:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Open finish.");
bDevConnected = true;

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
bDevConnected = false;
```

```
//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("\n Close finish.");
} while (false);
if (MyCamera.MV_OK != nRet)
{

    //Make sure the device is shutted down
    if ( bDevConnected )
    {
        device.MV_CC_CloseDevice_NET();
        bDevConnected = false;
    }

    // Destroy device
    device.MV_CC_DestroyDevice_NET();
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module GrabImage

Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
```

```
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
If (0 = m_stDeviceInfoList.nDeviceNum) Then
    MsgBox("No Find Gige | Usb Device !")
    Return
End If

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
Exit Do
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

'Set device baud rate
nRet = dev.MV_CAML_SetDeviceBauderate_NET(MyCamera.MV_CAML_BAUDRATE_115200)
If 0 <> nRet Then
    Console.WriteLine("Set device bauderate fail:{0:x8}", nRet)
Exit Do
End If

'Shut device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle and release resource
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
```

```
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub
```

```
End Module
```

MvCamCtrl.NET::MyCamera::MV_CAML_SetGenCPTimeOut_NET

Set the waiting time of serial port operation.

API Definition

```
int MV_CAML_SetGenCPTimeOut_NET(
    uint    nMillisec
);
```

Parameters

nMillisec

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace CamLBasicDemo
{
    class CamLBasicDemo
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            bool bDevConnected = false; //whether a device is connected
            do
            {
                // Enumerate deices
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_CAMERALINK_DEVICE, ref stDevList);
                if (MyCamera.MV_OK != nRet)
```



```
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo;
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Set the waiting time of serial port operation
nRet = device.MV_CAML_SetGenCPTimeOut_NET((uint)MyCamera.nMillisec);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set device bauderate fail:{0:x8}", nRet);
    break;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Open finish.");
bDevConnected = true;

// Shut down device
nRet = device.MV_CC_CloseDevice_NET();
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
bDevConnected = false;

// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("\n Close finish.");
} while (false);
if (MyCamera.MV_OK != nRet)
{
    // Make sure that the device is shutted down
    if ( bDevConnected )
    {
        device.MV_CC_CloseDevice_NET();
        bDevConnected = false;
    }

    // Destroy device
    device.MV_CC_DestroyDevice_NET();
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module GrabImage

Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        ' Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
```

```
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Enum Device failed:{0:x8}", nRet)
    Exit Do
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
If (0 = m_stDeviceInfoList.nDeviceNum) Then
    MsgBox("No Find Gige | Usb Device !")
    Return
End If

' Create device
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If
Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Set the witing time of serial port operation
nRet = dev.MV_CAML_SetGenCPTimeOut_NET(MyCamera.nMillisec)
If 0 <> nRet Then
    Console.WriteLine("Set device bauderate fail:{0:x8}", nRet)
    Exit Do
End If

' Shut down device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy device
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then
```

```
' Destroy device
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.3.4 GenTL APIs

MvCamCtrl.NET::MyCamera::MV_CC_CreateDeviceByGenTL_NET

Create a device handle via GenTL device information.

API Definition

```
Int32 MV_CC_CreateDeviceByGenTL_NET(
    ref MV_GENTL_DEV_INFO  stDevInfo
)
```

Parameters

stDevInfo

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

Before calling this API, you should call API

MvCamCtrl.NET::MyCamera::MV_CC_EnumDevicesByGenTL_NET to get the device information first.

Example

C#

```
MyCamera.MV_GENTL_DEV_INFO device =
(MyCamera.MV_GENTL_DEV_INFO)Marshal.PtrToStructure(m_stDeviceList.pDeviceInfo[cmbDeviceList.SelectedIndex],
    typeof(MyCamera.MV_GENTL_DEV_INFO));
int nRet = m_MyCamera.MV_CC_CreateDeviceByGenTL_NET(ref device);
if (MyCamera.MV_OK != nRet)
{
```

```
    return nRet;
}
```

Example

VB

```
Dim device As MyCamera.MV_GENTL_DEV_INFO = New MyCamera.MV_GENTL_DEV_INFO
device = CType(Marshal.PtrToStructure(m_stDeviceList.pDeviceInfo[0], GetType(MyCamera.MV_GENTL_DEV_INFO)),
MyCamera.MV_GENTL_DEV_INFO)
int nRet = m_MyCamera.MV_CC_CreateDeviceByGenTL_NET(device)
If (MyCamera.MV_OK <> nRet) Then
    return nRet
End If
```

MvCamCtrl.NET::MyCamera::MV_CC_EnumDevicesByGenTL_NET

Enumerate devices via GenTL interface.

API Definition

```
Int32 MV_CC_EnumDevicesByGenTL_NET(
    ref MV_GENTL_IF_INFO    stIfInfo,
    ref MV_GENTL_DEV_INFO_LIST stDevList
)
```

Parameters

stIfInfo

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

stDevList

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

Before calling this API, you should call API

MvCamCtrl.NET::MyCamera::MV_CC_EnumInterfacesByGenTL_NET to enumerate the interface first.

Example

C#

```
MyCamera.MV_GENTL_IF_INFO stIfInfo =
(MyCamera.MV_GENTL_IF_INFO)Marshal.PtrToStructure(m_stIfInfoList.pIfInfo[0],
typeof(MyCamera.MV_GENTL_IF_INFO));

MyCamera.MV_GENTL_DEV_INFO_LIST m_stDeviceList = new MyCamera.MV_GENTL_DEV_INFO_LIST();
```

```
int nRet = MyCamera.MV_CC_EnumDevicesByGenTL_NET(ref stIInfo, ref m_stDeviceList);
if (0 != nRet)
{
    return nRet;
}
```

Example

VB

```
Dim stIInfo As MyCamera.MV_GENTL_IF_INFO = New MyCamera.MV_GENTL_IF_INFO
stIInfo = CType(Marshal.PtrToStructure(m_stIInfoList.plIInfo(0), GetType(MyCamera.MV_GENTL_IF_INFO)),
MyCamera.MV_GENTL_IF_INFO)
Dim m_stDeviceList As MyCamera.MV_GENTL_DEV_INFO_LIST = new MyCamera.MV_GENTL_DEV_INFO_LIST
int nRet = MyCamera.MV_CC_EnumDevicesByGenTL_NET(m_stIInfoList, m_stDeviceList)
If (MyCamera.MV_OK <> nRet) Then
    return nRet
End If
```

MvCamCtrl.NET::MyCamera::MV_CC_EnumInterfacesByGenTL_NET

Enumerate interfaces via GenTL.

API Definition

```
Int32 MV_CC_EnumInterfacesByGenTL_NET(
    ref MV_GENTL_IF_INFO_LIST  stIInfoList,
    String                      pGenTLPath
);
```

Parameters

stIInfoList

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

pGenTLPath

[IN] GenTL CTI file path

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

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

Example

C#

```
MyCamera.MV_GENTL_IF_INFO_LIST m_stIfInfoList = new MyCamera.MV_GENTL_IF_INFO_LIST();
int nRet = MyCamera.MV_CC_EnumInterfacesByGenTL_NET(ref m_stIfInfoList, FileDialog.FileName);
if (0 != nRet)
{
    return nRet;
}
```

Example

VB

```
Dim m_stIfInfoList As MyCamera.MV_GENTL_IF_INFO_LIST = new MyCamera.MV_GENTL_IF_INFO_LIST
int nRet = MyCamera.MV_CC_EnumInterfacesByGenTL_NET(m_stIfInfoList,FileDialog.FileName)
If (MyCamera.MV_OK <> nRet) Then
    return nRet
End If
```

4.4 Image Acquisition

4.4.1 MvCamCtrl.NET::MyCamera::MV_CC_ClearImageBuffer_NET

Clear streaming data buffer.

API Definition

```
int MV_CC_ClearImageBuffer_NET(
);
```

Return Value

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

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Grab_Callback
{
    class Grab_Callback
    {
```

```
public static MyCamera.cbOutputExdelegate ImageCallback;
public static MyCamera device = new MyCamera();
static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
{
    Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
Convert.ToString(pFrameInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    do

    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

        //Print device information
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
```



```
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
    Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
}
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
}
```

```
        break;
    }

    //Register image callback
    ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
    nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Register image callback failed!");
        break;
    }

    //Start grabbing image
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    Console.WriteLine("Press a key to change the grabbing pattern");
    Console.ReadLine();
    //Clear the image buffer
    nRet = device.MV_CC_ClearImageBuffer_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Clear Image Buffer failed:{0:x8}", nRet);
        break;
    }

    //Set trigger mode to on
    nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 1);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set TriggerMode failed:{0:x8}", nRet);
        break;
    }

    //Set trigger source as software
    nRet = device.MV_CC_SetEnumValue_NET("TriggerSource", 7);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set trigger source failed:{0:x8}", nRet);
        break;
    }

    //Set trigger command
    nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine"Set Trigger Source failed:{0:x8}", nRet);
        break;
    }
```

```
}

Console.WriteLine("Press a key to stop grabbing");
Console.ReadLine();

//Stop grabbing image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
```

```
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

```
Module Grab_Callback
```

```
Dim dev As MyCamera = New MyCamera
Dim pBufForSaveImage As IntPtr
Dim nBufForSaveImage As Int32
Dim m_bytImageBuffer(1024 * 1024 * 1) As Byte
Dim m_bytImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
Private Sub cbOutputdelegateFunc(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO,
ByVal pUser As IntPtr)
    Console.WriteLine("Width:" + Convert.ToString(pFrameInfo.nWidth) + " Height:" +
Convert.ToString(pFrameInfo.nHeight) + " FrameNum:" + Convert.ToString(pFrameInfo.nFrameNum))
End Sub
```

```
Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)
```

```
Do While (True)
    'Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enum Device failed:{0:x8}", nRet)
        Return
    End If
```

```
    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device !")
        Return
    End If
```

```
    ' Print device information
    Dim i As Int32
    For i = 0 To stDeviceInfoList.nDeviceNum - 1
        Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
        stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
            Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
            Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
            Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
            stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
            Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
            Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
```

```
Dim nIpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
Dim nIpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nIpByte1.ToString() + "." + nIpByte2.ToString() + "." + nIpByte3.ToString() + "." + nIpByte4.ToString() + "]"
Console.WriteLine(Info)
Else
    Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
    Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
    stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
    Console.WriteLine(Info)
End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
Exit Do
```

```
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

' Get payload size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
Exit Do
End If

nBufForSaveImage = stParam.nCurValue * 3 + 2048
pBufForSaveImage = Marshal.AllocHGlobal(nBufForSaveImage)

' Register image callback
nRet = dev.MV_CC_RegisterImageCallBack_NET(cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
Exit Do
End If

' Start grabbing image
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
Exit Do
End If

Console.WriteLine("Press a key to change the grabbing pattern")
System.Console.ReadLine()

' Clear the image buffer
nRet = dev.MV_CC_ClearImageBuffer_NET()
If 0 <> nRet Then
    Console.WriteLine("Clear Image Buffer failed:{0:x8}", nRet)
Exit Do
End If

' Set trigger mode to on
nRet = dev.MV_CC_SetEnumValue_NET("TriggerMode", 1)
If 0 <> nRet Then
    Console.WriteLine("Set TriggerMode failed:{0:x8}", nRet)
Exit Do
End If

' Set trigger source as software
```

```
nRet = dev.MV_CC_SetEnumValue_NET("TriggerSource", 7)
If 0 <> nRet Then
    Console.WriteLine("Set trigger source failed:{0:x8}", nRet)
Exit Do
End If

' Set trigger command
nRet = dev.MV_CC_SetCommandValue_NET("TriggerSoftware")
If 0 <> nRet Then
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet)
Exit Do
End If

Console.WriteLine("Press a key to stop grabbing")
System.Console.ReadLine()

' Stop grabbing image
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub
```

End Module

4.4.2 MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET

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_NET).

API Definition

```
int MV_CC_FreelImageBuffer_NET(  
    ref MyCamera.MV_FRAME_OUT  pFrame  
);
```

Parameters

pFrame

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

Return Values

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

Remarks

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

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET; namespace Grab  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
```



```
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}

int nCount = 0;
MyCamera.MV_FRAME_OUT Frame = new MyCamera.MV_FRAME_OUT();
while (nCount++ != 10)
{
    //Get one image frame
```

```
nRet = device.MV_CC_GetImageBuffer_NET(ref Frame, 1000);
if (MyCamera.MV_OK == nRet)
{
    Console.WriteLine("Width:" + Convert.ToString(Frame.stFrameInfo.nWidth) + " Height:" +
Convert.ToString(Frame.stFrameInfo.nHeight)
        + " FrameNum:" + Convert.ToString(Frame.stFrameInfo.nFrameNum));
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}

nRet = device.MV_CC_FreelImageBuffer_NET(ref Frame);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Free Image Buffer failed{0:x8}", nRet);
}

//...other processing

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
```

```
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        'Start acquisition
        nRet = dev.MV_CC_StartGrabbing_NET()
        If 0 <> nRet Then
            Console.WriteLine("Start grabbing failed!")
        End If
        Console.WriteLine("Start grabbing succeed!")

        'Acquiring image
        Dim nCount As Int32 = 0
```

```
Dim Frame As MyCamera.MV_FRAME_OUT = New MyCamera.MV_FRAME_OUT

Do While nCount <> 10
    nCount = nCount + 1
    nRet = dev.MV_CC_GetImageBuffer_NET(Frame, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(Frame.stFrameInfo.nWidth) + " height:" +
Convert.ToString(Frame.stFrameInfo.nHeight) + "FrameNum:" + Convert.ToString(Frame.stFrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If

    nRet = dev.MV_CC_FreelImageBuffer_NET(Frame)
    If 0 <> nRet Then
        Console.WriteLine("Free Image Buffer failed!")
    End If
Loop

//...other processing

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
    Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
    Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.4.3 MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET

Get one image frame, supports getting chunk information and setting timeout.

API Definition

```
int MV_CC_GetImageBuffer_NET(  
    ref MyCamera.MV_FRAME_OUT    pFrame,  
    int                          nMsec  
);
```

Parameters

pFrame

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

nMsec

[IN] Timeout period, unit: millisecond

Return Values

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

Remarks

- Before calling this API to get image data frame, you should call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** 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 should be used with ***MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET***, after processing the acquired data, you should call ***MvCamCtrl.NET::MyCamera::MV_CC_FreelImageBuffer_NET*** to release the data pointer permission of pFrame.
- This API cannot be called to get stream after calling .
- This API is supported by both USB3Vision camera and GIGE camera.
- This API is not supported by CameraLink device.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Grab  
{  
    class Program  
    {  
  
        static void Main(string[] args)  
        {
```

```
uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}

int nCount = 0;
MyCamera.MV_FRAME_OUT Frame = new MyCamera.MV_FRAME_OUT();
while (nCount++ != 10)
```

```
{
    //Get one image frame
    nRet = device.MV_CC_GetImageBuffer_NET(ref Frame, 1000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Width:" + Convert.ToString(Frame.stFrameInfo.nWidth) + " Height:" +
Convert.ToString(Frame.stFrameInfo.nHeight)
            + " FrameNum:" + Convert.ToString(Frame.stFrameInfo.nFrameNum));
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }

    nRet = device.MV_CC_FreelImageBuffer_NET(ref Frame);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Free Image Buffer failed{0:x8}", nRet);
    }
}

//...other processing

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        'Start acquisition
        nRet = dev.MV_CC_StartGrabbing_NET()
        If 0 <> nRet Then
            Console.WriteLine("Start grabbing failed!")
        End If
        Console.WriteLine("Start grabbing succeed!")
    End Sub
End Module
```



```
'Acquiring image
Dim nCount As Int32 = 0

Dim Frame As MyCamera.MV_FRAME_OUT = New MyCamera.MV_FRAME_OUT

Do While nCount <> 10
    nCount = nCount + 1
    nRet = dev.MV_CC_GetImageBuffer_NET(Frame, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(Frame.stFrameInfo.nWidth) + " height:" +
Convert.ToString(Frame.stFrameInfo.nHeight) + "FrameNum:" + Convert.ToString(Frame.stFrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If

    nRet = dev.MV_CC_FreedImageBuffer_NET(Frame)
    If 0 <> nRet Then
        Console.WriteLine("Free Image Buffer failed!")
    End If
Loop

//...other processing

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed!")
    End If
    Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.4.4 MvCamCtrl.NET::MyCamera::MV_CC_GetImageForBGR_NET

Get one picture frame in BGR24 format, search for the frames in the memory and transform them to BGR24 format for return, supports setting timeout.

API Definition

```
int MV_CC_GetImageForBGR_NET(  
    IntPtr          pData,  
    uint            nDataSize,  
    ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo,  
    int             nMsec  
);
```

Parameters

pData

[IN] Address of buffer for saving pictures.

nDataSize

[IN] Buffer size

pFrameInfo

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

nMsec

[IN] Waiting timeout, unit: millisecond.

Return Value

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

Remarks

- When calling this API, it will search for data in the buffer. If there is data in the buffer, the data will be transformed to BGR24 format and return; if there is no data, error code will be returned. As transforming picture to BGR24 format generates time consumption, if the frame rate is too high, frame loss may occur.
- Before calling this API, you must call **MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET** to start the image acquisition. This API is repeatedly called to get frame data, so 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

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;
```

```
using MvCamCtrl.NET; namespace Grab
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();      int nRet
= MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;          //Change the device information structure pointer to
device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));          MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }
            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive, 0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                return;
            }
            //Start acquisition
            nRet = device.MV_CC_StartGrabbing_NET();
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
                return;
            }
            //Get Payload Size
            MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
            nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
                break;
            }
            uint nBufSize = stParam.nCurValue;
```

```
IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    //Get one frame
    nRet = device.MV_CC_GetImageForBGR_NET(pBufForDriver, nBufSize, ref FrameInfo, 1000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight)
        + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum));
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}
Marshal.FreeHGlobal(pBufForDriver);

//Other process...      //Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}
//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
```

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    'Enumerate devices
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
        Return
    End If
    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device !")
        Return
    End If
    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

    ' Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Create device failed!")
    End If
    Console.WriteLine("Create device succeed")

    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed!")
    End If
    Console.WriteLine("Open device succeed!")

    'Start image acquisition
    nRet = dev.MV_CC_StartGrabbing_NET()
    If 0 <> nRet Then

        Console.WriteLine("Start grabbing failed!")
    End If
    Console.WriteLine("Start grabbing succeed!")

    'Image acquisition

    'Get Payload Size
    Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
    nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
    If (MyCamera.MV_OK <> nRet) Then
        Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
        Exit Do
    End If
```

```
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
Do While nCount <> 10
    nRet = dev.MV_CC_GetImageForBGR_NET(pBufferForDriver, nBufSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) + "FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If
Loop
Marshal.FreeHGlobal(pBufferForDriver)

//Other process...
'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.4.5 MvCamCtrl.NET::MyCamera::MV_CC_GetImageForRGB_NET

Get one picture frame in RGB24 format, search for the frames in the memory and transform them to RGB24 format for return, supports setting timeout.

API Definition

```
int MV_CC_GetImageForRGB_NET(
    IntPtr          pData,
    uint            nDataSize,
```

```
ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo,  
int nMsec  
);
```

Parameters

pData

[IN] Address of buffer for saving pictures.

nDataSize

[IN] Buffer size

pFrameInfo

[OUT] Obtained frame information, which is in BGR24 format. See the structure ***MV_FRAME_OUT_INFO_EX*** for details.

nMsec

[IN] Waiting timeout, unit: millisecond.

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

- When calling this API, it will search for data in the buffer. If there is data in the buffer, the data will be transformed to BGR24 format and return; if there is no data, error code will be returned. As transforming picture to BGR24 format generates time consumption, if the frame rate is too high, frame loss may occur.
- Before calling this API, you must call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the image acquisition. This API is repeatedly called to get frame data, so 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

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Grab  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
```

```
int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}

//Get Payload Size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
```



```
}
uint nBufSize = stParam.nCurValue;

IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    //Get one frame
    nRet = device.MV_CC_GetImageForRGB_NET(pBufForDriver, nBufSize, ref FrameInfo,
    1000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight)
        + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum));
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}
Marshal.FreeHGlobal(pBufForDriver);

//Other process...

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        'Start getting stream
        nRet = dev.MV_CC_StartGrabbing_NET()
        If 0 <> nRet Then
            Console.WriteLine("Start grabbing failed!")
        End If
    End Sub
End Module
```

```
End If
Console.WriteLine("Start grabbing succeed!")

'Image acquisition

'Get Payload Size

Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
Exit Do

End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX

Do While nCount <> 10
    nRet = dev.MV_CC_GetImageForRGB_NET(pBufferForDriver, nBufSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) + "FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If
Loop
Marshal.FreeHGlobal(pBufferForDriver)

//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")
```

```
End Sub

End Module
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        'Start getting stream
        nRet = dev.MV_CC_StartGrabbing_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Image acquisition

'Get Payload Size

Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
Exit Do

End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim pBufForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX

Do While nCount <> 10
    nRet = dev.MV_CC_GetImageForRGB_NET(pBufForDriver, nBufSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) + "FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If
    Loop
    Marshal.FreeHGlobal(pBufForDriver)

//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
```

```
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.4.6 MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET

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

API Definition

```
int MV_CC_GetOneFrameTimeout_NET(
    IntPtr          pData,
    uint            nDataSize,
    ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo,
    int             nMsec
);
```

Parameters

pData

[IN] Address of buffer for saving pictures.

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 Values

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- Before calling this API, you must call **MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET** to start image acquisition. This API is repeatedly called to get frame data, so the upper-layer program should control the frequency of calling this API according to the frame rate. This API supports setting timeout, which can improve the stationarity of getting stream. If timed out, this API will not return until there is data.
- This API is supported both by USB3Vision and GIGE camera.
- This API is not supported by CameraLink device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Grab
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }

            /Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
            }
        }
    }
}
```

```
        return;
    }

    //Start acquisition
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        return;
    }

    //Get Payload Size
    MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
    nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
        break;
    }
    uint nBufSize = stParam.nCurValue;

    IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
    MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
    while (nCount++ != 10)
    {
        //Get one frame
        nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, ref FrameInfo, 1000);
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight)
                + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum));
        }
        else
        {
            Console.WriteLine("No data:{0:x8}", nRet);
        }
    }
    Marshal.FreeHGlobal(pBufForDriver);

    //Other process...

    //Stop acquisition
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        return;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
```



```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
    End Sub
End Module
```

```
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Image acquisition
Dim nCount As Int32 = 0
Dim nBufSize As Int32 = 1024 * 1024 * 50
Dim pBufForDriver As IntPtr = Marshal.AllocHGlobal(nBufSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX

Do While nCount <> 10
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Else
        Console.WriteLine("No data{0:x8}", nRet)
    End If
    Loop
    Marshal.FreeHGlobal(pBufForDriver)

    '//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
```

```
nRet = dev.MV_CC_DestroyDevice_NET()  
If 0 <> nRet Then  
    Console.WriteLine("Destroy device failed!")  
End If  
Console.WriteLine("Destroy device succeed!")  
  
End Sub  
  
End Module
```

4.4.7 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackEx_NET

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

Structure Definition

```
int MV_CC_RegisterImageCallBackEx_NET(  
    MyCamera.cbOutputExdelegate cbOutput,  
    IntPtr pUser  
);
```

Parameters

cbOutput

[IN] Image data callback function

```
void cbOutputExdelegate(  
    IntPtr pData,  
    ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo  
    IntPtr pUser  
);
```

pData

[OUT] Buffer address for saving image data

pFrameInfo

[OUT] Obtained frame information, including width, height, pixel format, chunk information and so on. See ***MV_FRAME_OUT_INFO_EX*** for details.

pUser

[IN] User data

pUser

[IN] User data

Return Values

Return *MyCamera.MV_OK* (0) on success; return **Error Code** on failure.

Remarks

- This API should be called after calling ***MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET***.
- Two image acquisition modes are available, and these two modes cannot be used at same time:
Mode 1: Call this API to set image callback function, and then call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition, The collected image will be returned in the configured callback function.
Mode 2: Call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition, and then repeatedly call ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET*** to get the frame data with specific pixel format in the application layer.
When getting frame data, the upper-layer program should control the frequency of calling this API according to the frame rate.
- This API is not supported in the MAC system.

See Also

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace RegisterImageCallBack
{
    class Program
    {
        static void CallbackEx(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth + ",Height:" +
                pFrameInfo.nHeight);
        }

        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
            }
        }
    }
}
```

```
        return;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;

    //Change the device information structure pointer to device information structure
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    MyCamera device = new MyCamera();

    //Create device handle
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        return;
    }

    //Open device
    nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        return;
    }

    MyCamera.cbOutputExdelegate ImageCallback;
    ImageCallback = new MyCamera.cbOutputExdelegate(CallbackEx);

    nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Register ImageCallBack failed{0:x8}", nRet);
        return;
    }

    //Start acquisition
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        return;
    }
    Thread.Sleep(10000);

    //Other process...

    //Stop acquisition
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
```

```
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        return;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Image_Callback

```
    Dim dev As MyCamera = New MyCamera
    Dim pBufForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)"You should allocate the memory size
according to camera resolution
    Dim m_byteImageBuffer(1024 * 1024 * 60) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 60

    Private Sub CallbackEx(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX, ByVal
pUser As IntPtr)
        Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth + ",Height:" +
pFrameInfo.nHeight)
    End Sub

    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim imgCallback As MyCamera.cbOutputExdelegate = New MyCamera.cbOutputExdelegate(AddressOf CallbackEx)
        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
```

```
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
End If

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If

nRet = dev.MV_CC_RegisterImageCallBackEx_NET(imgCallback, IntPtr.Zero)
If 0 <> nRet Then
    Console.WriteLine("Register ImageCallBack failed")
End If

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
End If

Console.WriteLine("push enter to exit")
System.Console.ReadLine()

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
End If

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If
```

```
'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If

End Sub

End Module
```

4.4.8 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForBGR_NET

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

API Definition

```
int MV_CC_RegisterImageCallBackForBGR_NET(
    MyCamera.cbOutputExdelegate    cbOutput,
    IntPtr                          pUser
);
```

Parameters

cbOutput

[IN] Callback function for BGR24 picture.

```
IntPtr          pData,
ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo
IntPtr          pUser
);
```

pData

[OUT] Buffer address for saving image data

pFrameInfo

[OUT] Obtained frame information, including width, height, pixel format, chunk information.

See ***MV_FRAME_OUT_INFO_EX*** for details.

pUser

[OUT] User data

pUser

[IN] User data

Return Value

Return *MyCamera.MV_OK* (0) on success; return **Error Code** on failure.

Remarks

- This API should be called after calling ***MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET***.
- Two image acquisition modes are available, and these two modes cannot be used at same time:
Mode 1: Call this API to set image callback function, and then call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition, The collected image will be returned in the configured callback function.
Mode 2: Call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition, and then repeatedly call ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageForBGR_NET*** to get the frame data in BGR24 format in the application layer.
When getting frame data, 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

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace RegisterImageCallBack
{
    class Program
    {
        static void CallbackBGR(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth + ",Height:" +
                pFrameInfo.nHeight);
        }

        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }
        }
    }
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.cbOutputExdelegate ImageCallback;
ImageCallback = new MyCamera.cbOutputExdelegate(CallbackBGR);

nRet = device.MV_CC_RegisterImageCallBackForBGR_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register ImageCallBack failed{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}
Thread.Sleep(10000);

//Other process...

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}
```

```
//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Image_Callback

```
    Dim dev As MyCamera = New MyCamera
    Dim pBufForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)"You should allocate the memory size
according to camera resolution
    Dim m_byteImageBuffer(1024 * 1024 * 60) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 60

    Private Sub CallbackBGR(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX, ByVal
pUser As IntPtr)      Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth +
",Height:" + pFrameInfo.nHeight)
    End Sub

    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim bgrCallback As MyCamera.cbOutputExdelegate = New MyCamera.cbOutputExdelegate(AddressOf
CallbackBGR)

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
```

```
Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
End If

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If

nRet = dev.MV_CC_RegisterImageCallBackForBGR_NET(bgrCallback, IntPtr.Zero)
If 0 <> nRet Then
    Console.WriteLine("Register ImageCallBack failed")
End If

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
End If

Console.WriteLine("push enter to exit")
System.Console.ReadLine()

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
End If

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
End If

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If

End Sub

End Module
```

4.4.9 MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForRGB_NET

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

API Definition

```
int MV_CC_RegisterImageCallBackForRGB_NET(
MyCamera.cbOutputExdelegate  cbOutput,
IntPtr                       pUser
);
```

Parameters

cbOutput

[IN] Callback function for RBG24 picture.

```
IntPtr           pData,
ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo
IntPtr           pUser
);
```

pData

[OUT] Buffer address for saving image data

pFrameInfo

[OUT] Obtained frame information, including width, height, pixel format, chunk information.
See **MV_FRAME_OUT_INFO_EX** for details.

pUser

[OUT] User data

pUser

[IN] User data

Return Value

Return *MyCamera.MV_OK* (0) on success; return **Error Code** on failure.

Remarks

- This API should be called after calling **MvCamCtrl.NET::MyCamera::MV_CC_CreateDevice_NET**.
- Two image acquisition modes are available, and these two modes cannot be used at same time:

Mode 1: Call this API to set image callback function, and then call

MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET to start the acquisition, The collected image will be returned in the configured callback function.

Mode 2: Call ***MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET*** to start the acquisition, and then repeatedly call ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageForRGB_NET*** to get the frame data in RGB24 format in the application layer.

When getting frame data, 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

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace RegisterImageCallBack
{
    class Program
    {
        static void CallbackRGB(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth + ",Height:" +
                pFrameInfo.nHeight);
        }

        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
                typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

MyCamera.cbOutputExdelegate ImageCallback;
ImageCallback = new MyCamera.cbOutputExdelegate(CallbackRGB);

nRet = device.MV_CC_RegisterImageCallBackForRGB_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register ImageCallBack failed{0:x8}", nRet);
    return;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}
Thread.Sleep(10000);

//Other process...

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{

```

```
        Console.WriteLine("Close device failed:{0:x8}", nRet);
        return;
    }

    //Destroy handle and release resources
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Image_Callback

```
    Dim dev As MyCamera = New MyCamera
    Dim pBufForSaveImage As IntPtr = Marshal.AllocHGlobal(1024 * 1024 * 60)"You should allocate the memory size
according to camera resolution
    Dim m_bytImageBuffer(1024 * 1024 * 60) As Byte
    Dim m_bytImageBufferLen As Int32 = 1024 * 1024 * 60

    Private Sub CallbackRGB(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX, ByVal
pUser As IntPtr)
        Console.WriteLine("FrameNum:" + pFrameInfo.nFrameNum + ",Width:" + pFrameInfo.nWidth + ",Height:" +
pFrameInfo.nHeight)
    End Sub

    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim rgbCallback As MyCamera.cbOutputExdelegate = New MyCamera.cbOutputExdelegate(AddressOf
CallbackRGB)

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed:{0:x8}", nRet)
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
```



```
        Console.WriteLine("No Find Gige | Usb Device !")
        Return
    End If

    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
    GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

    'Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Create device failed:{0:x8}", nRet)
    End If

    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed:{0:x8}", nRet)
    End If

    nRet = dev.MV_CC_RegisterImageCallBackForRGB_NET(rgbCallback, IntPtr.Zero)
    If 0 <> nRet Then
        Console.WriteLine("Register ImageCallBack failed")
    End If

    'Start getting stream
    nRet = dev.MV_CC_StartGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    End If

    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()

    'Stop getting stream
    nRet = dev.MV_CC_StopGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    End If

    'Close camera
    nRet = dev.MV_CC_CloseDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Open device failed:{0:x8}", nRet)
    End If

    'Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
```

End Sub

End Module

4.4.10 MvCamCtrl.NET::MyCamera::MV_CC_StartGrabbing_NET

Start the image acquisition.

API Definition

```
int MV_CC_StartGrabbing_NET(  
);
```

Return Values

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

This API is not supported by CameraLink device.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Grab  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
        }  
    }  
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

//Start the acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Starting the acquisition failed:{0:x8}", nRet);
    return;
}

//Other process...

//Stop the acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stopping the acquisition failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
```

```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroying device failed:{0:x8}", nRet);
        }
    }
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Creating device handle failed.")
        End If
        Console.WriteLine("The device handle is created.")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Opening camera failed.")
        End If
    End Sub
End Module
```

```
End If
Console.WriteLine("The camera is open.")

'Start acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Starting acquisition failed:{0:x8}", nRet)
End If
Console.WriteLine("The acquisition is started.")

//Other process...

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stopping acquisition failed:{0:x8}", nRet)
End If
Console.WriteLine("The acquisition is stopped.")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

4.4.11 MvCamCtrl.NET::MyCamera::MV_CC_StopGrabbing_NET

Stop the image acquisition.

API Definition

```
int MV_CC_StopGrabbing_NET(
);
```

Return Values

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

This API is not supported by CameraLink device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Grab
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Creating device failed:{0:x8}", nRet);
                return;
            }

            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

//Start the acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Starting the acquisition failed:{0:x8}", nRet);
    return;
}

//Other process...

//Stop the acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stopping the acquisition failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Module1

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    'Enumerate devices
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enumerating device failed."+ Convert.ToString(nRet))
        Return
    End If

    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device.")
        Return
    End If

    Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
    stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

    'Create handle
    nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
    If 0 <> nRet Then
        Console.WriteLine("Creating device handle failed.")
    End If
    Console.WriteLine("The device handle is created.")

    'Open camera
    nRet = dev.MV_CC_OpenDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Opening camera failed.")
    End If
    Console.WriteLine("The camera is open.")

    'Start acquisition
    nRet = dev.MV_CC_StartGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Starting acquisition failed:{0:x8}", nRet)
    End If
    Console.WriteLine("The acquisition is started.")

    '//Other process...

    'Stop acquisition
    nRet = dev.MV_CC_StopGrabbing_NET()
    If 0 <> nRet Then
        Console.WriteLine("Stopping acquisition failed:{0:x8}", nRet)
    End If
    Console.WriteLine("The acquisition is stopped.")
```



```
'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

4.5 Image Processing

4.5.1 MvCamCtrl.NET::MyCamera::MV_CC_DisplayOneFrame_NET

Display one frame.

API Definition

```
int MV_CC_DisplayOneFrame_NET(
    ref MyCamera.MV_DISPLAY_FRAME_INFO pDisplayInfo
);
```

Parameters

pDisplayInfo

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

Return Values

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

- You can display one frame on specified window via this API and ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET*** in pairs.
- This API is not supported by CameraLink device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Grab
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nTLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }

            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive, 0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                return;
            }
        }
    }
}
```

```
//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    return;
}

//Get Payload Size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
uint nBufSize = stParam.nCurValue;

IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    //Get one frame
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, ref FrameInfo, 1000);
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight)
        + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum));

        MyCamera.MV_DISPLAY_FRAME_INFO stDisplayInfo = new MyCamera.MV_DISPLAY_FRAME_INFO();
        stDisplayInfo.hWnd = handle;
        stDisplayInfo.pData = pBufForDriver;
        stDisplayInfo.nDataLen = FrameInfo.nFrameLen;
        stDisplayInfo.nHeight = FrameInfo.nHeight;
        stDisplayInfo.nWidth = FrameInfo.nWidth;
        stDisplayInfo.enPixelType = FrameInfo.enPixelType;
        nRet = device.MV_CC_DisplayOneFrame_NET(ref stDisplayInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Display One Frame failed:{0:x8}", nRet);
            continue;
        }
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}
Marshal.FreeHGlobal(pBufForDriver);

//Other process...
```

```
//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
        End If
    End Sub
End Module
```

```
Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Image acquisition

'Get Payload Size

Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)

Exit Do

End If
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX

Do While nCount <> 10
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufferForDriver, nPayloadSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
        Dim stDisplayInfo As MyCamera.MV_DISPLAY_FRAME_INFO = New MyCamera.MV_DISPLAY_FRAME_INFO()
        stDisplayInfo.hWnd = handle
        stDisplayInfo.pData = pBufferForDriver
        stDisplayInfo.nDataLen = FrameInfo.nFrameLen
```

```
stDisplayInfo.nHeight = FrameInfo.nHeight
stDisplayInfo.nWidth = FrameInfo.nWidth
stDisplayInfo.enPixelFormat = FrameInfo.enPixelFormat
nRet = dev.MV_CC_DisplayOneFrame_NET(stDisplayInfo)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Display One Frame failed:{0:x8}", nRet)
    Continue Do
End If

nCount = nCount + 1
Continue Do
End If

Loop
Marshal.FreeHGlobal(pBufForDriver)

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub

End Module
```

4.5.2 MvCamCtrl.NET::MyCamera::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_NET(  
    ref MV_SAVE_IMAGE_PARAM_EX  *pSaveParam  
);
```

Parameters

pSaveParam

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

- Once there is image data, you can call this API to convert the data.
- You can also call ***MvCamCtrl.NET::MyCamera::MV_CC_GetOneFrameTimeout_NET*** or ***MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackEx_NET*** or ***MvCamCtrl.NET::MyCamera::MV_CC_GetImageBuffer_NET*** to get one image frame and set the callback function, and then call this API to convert the format.

4.5.3 MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET

Convert pixel format.

API Definition

```
int MV_CC_ConvertPixelFormat_NET(  
    ref MyCamera.MV_CC_PIXEL_CONVERT_PARAM  pstCvtParam  
);
```

Parameters

pstCvtParam

[IN][OUT] Input and output parameters of pixel format conversion. See ***MV_CC_PIXEL_CONVERT_PARAM*** for details.

Return Values

Return *MyCamera.MV_OK (0)* on success; return ***Error Code*** on failure.

Remarks

Once there is image data, you can call this API to convert the data.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Grab
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }

            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
            typeof(MyCamera.MV_CC_DEVICE_INFO));

            MyCamera device = new MyCamera();

            //Create device handle
            nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Create device failed:{0:x8}", nRet);
                return;
            }

            //Open device
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
            }
        }
    }
}
```



```
        return;
    }

    //Start the acquisition
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        return;
    }

    //Get Payload Size
    MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
    nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
        break;
    }
    uint nBufSize = stParam.nCurValue;

    IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nBufSize);
    uint nSaveSize = 1024 * 1024 * 60; //You should allocate the memory size according to camera resolution
    IntPtr pBufForSave = Marshal.AllocHGlobal((int)nSaveSize);
    MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
    while (nCount++ != 10)
    {
        //Get one frame
        nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, ref FrameInfo, 1000);
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
                Convert.ToString(FrameInfo.nHeight)
                + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum));

            MyCamera.MV_PIXEL_CONVERT_PARAM stConverPixelParam = new
            MyCamera.MV_PIXEL_CONVERT_PARAM();
            stConverPixelParam.nWidth = FrameInfo.nWidth;
            stConverPixelParam.nHeight = FrameInfo.nHeight;
            stConverPixelParam.pSrcData = pBufForDriver;
            stConverPixelParam.nSrcDataLen = FrameInfo.nFrameLen;
            stConverPixelParam.enSrcPixelFormat = FrameInfo.enPixelFormat;
            stConverPixelParam.enDstPixelFormat = MyCamera.MvGvspPixelFormat.PixelType_Gvsp_RGB8_Packed;
            stConverPixelParam.pDstBuffer = pBufForSave;
            stConverPixelParam.nDstBufferSize = nSaveSize;
            nRet = device.MV_CC_ConvertPixelFormat_NET(ref stConverPixelParam);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Convert PixelType Failed!");
                return;
            }
        }
        // Save picture data to local directory
    }
```

```
        byte[] data = new byte[stConverPixelParam.nDstBufferSize];
        Marshal.Copy(pBufForSave, data, 0, (int)stConverPixelParam.nDstBufferSize);
        FileStream pFile = new FileStream("AfterConvert_RGB.raw", FileMode.Create);
        pFile.Write(data, 0, data.Length);
        pFile.Close();
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}
Marshal.FreeHGlobal(pBufForDriver);
Marshal.FreeHGlobal(pBufForSave);

//Other process...

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    return;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

```
Module Module1
```

```
Sub Main()
Dim dev As MyCamera = New MyCamera
Dim Info As String
Dim nRet As Int32 = MyCamera.MV_OK
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

'Enumerate devices
nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Enumerating device failed!" + Convert.ToString(nRet))
    Return
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed!")
End If
Console.WriteLine("Create device succeed")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")

'Start getting stream
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Image acquisition

'Get Payload Size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
```

```
Exit Do

End If

Dim nBufSize = stParam.nCurValue

Dim pBufForDriver As IntPtr = Marshal.AllocHGlobal(nBufSize)
Dim nSaveSize As Int32 = 1024 * 1024 * 60'You should allocate the memory size according to camera resolution
Dim pBufForSave = Marshal.AllocHGlobal(nSaveSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX

Do While nCount <> 10
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nBufSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("width:" + Convert.ToString(FrameInfo.nWidth) + " height:" +
Convert.ToString(FrameInfo.nHeight) +
        "FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
        Dim stConverPixelParam As MyCamera.MV_PIXEL_CONVERT_PARAM = New
MyCamera.MV_PIXEL_CONVERT_PARAM()
        stConverPixelParam.nWidth = FrameInfo.nWidth
        stConverPixelParam.nHeight = FrameInfo.nHeight
        stConverPixelParam.pSrcData = pBufForDriver
        stConverPixelParam.nSrcDataLen = FrameInfo.nFrameLen
        stConverPixelParam.enSrcPixelFormat = FrameInfo.enPixelFormat
        stConverPixelParam.enDstPixelFormat = MyCamera.MvGvspPixelFormat.PixelType_Gvsp_RGB8_Packed
        stConverPixelParam.pDstBuffer = pBufForSave
        stConverPixelParam.nDstBufferSize = nSaveSize

        nRet = dev.MV_CC_ConvertPixelFormat_NET(stConverPixelParam)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Convert PixelType Failed:{0:x8}", nRet)
            Continue Do
        End If

    End If

End If

Loop
Marshal.FreeHGlobal(pBufForDriver)
Marshal.FreeHGlobal(pBufForSave)

//Other process...

'Stop getting stream
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop grabbing failed!")
End If
Console.WriteLine("Start grabbing succeed!")

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
```

```
End If
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")
```

```
End Sub
```

```
End Module
```

4.5.4 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCvtQuality_NET

Set the interpolation method of Bayer format.

API Definition

```
int MV_CC_SetBayerCvtQuality_NET(
    uint BayerCvtQuality
);
```

Parameters

BayerCvtQuality

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

When calling **MvCamCtrl.NET::MyCamera::MV_CC_GetImageForRGB_NET** or , or call **MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForRGB_NET** or **MvCamCtrl.NET::MyCamera::MV_CC_RegisterImageCallBackForBGR_NET** to register callback function, the Bayer data will start interpolation according to the configured method.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Param
```

```
{
class Program
{
    static void Main(string[] args)
    {
        uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

        int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
            return;
        }

        Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            return;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;

        //Change the device information structure pointer to device information structure
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        MyCamera device = new MyCamera();

        //Create device handle
        nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Creating device failed:{0:x8}", nRet);
            return;
        }

        //Open device
        nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Opening device failed:{0:x8}", nRet);
            return;
        }

        uint nBayerCvtQuality = 1;
        nRet = device.MV_CC_SetBayerCvtQuality_NET(nBayerCvtQuality);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Bayer CvtQuality failed:{0:x8}", nRet);
            return;
        }
    }
}
```

```
//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
```

```
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device handle failed.")
End If
Console.WriteLine("The device handle is created.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening camera failed.")
End If
Console.WriteLine("The camera is open.")

Dim nBayerCvtQuality As UInt32 = 1
nRet = dev.MV_CC_SetBayerCvtQuality_NET(nBayerCvtQuality)
If 0 <> nRet Then
    Console.WriteLine("Setting Bayer CvtQuality failed")
End If

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub

End Module
```

4.5.5 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaValue_NET

Set the Gamma value after Bayer interpolation.

API Definition

```
public Int32 MV_CC_SetBayerGammaValue_NET(  
    float      fBayerGammaValue  
);
```

Parameters

fBayerGammaValue

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

Return Value

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

Remarks

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

4.5.6 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerGammaParam_NET

Set gamma parameters of Bayer pattern.

API Definition

```
public Int32 MV_CC_SetBayerGammaParam_NET(  
    MV_CC_GAMMA_PARAM  pstGammaParam  
);
```

Parameters

pstGammaParam

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

Return Value

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

Remarks

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

4.5.7 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMPParam_NET

Color correction after Bayer interpolation.

API Definition

```
int MV_CC_SetBayerCCMPParam_NET(  
    MV_CC_CCM_PARAM    pstCCMPParam  
);
```

Parameters

pstCCMPParam

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

Return Value

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

Remarks

After calling this API, you can call *MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET* or *MvCamCtrl.NET::MyCamera::MV_CC_SaveImageEx2* to convert Bayer format to RGB24/BGR24/RGBA32/BGRA32.

4.5.8 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCCMPParamEx_NET

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

API Definition

```
public Int32 MV_CC_SetBayerCCMPParamEx_NET(  
    MV_CC_CCM_PARAM_EX    pstCCMPParam  
);
```

Parameters

pstCCMPParam

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

Return Value

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

Remarks

- After enabling color correction and setting color correction matrix, the CCM parameters take effect when you call API *MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET* or

MvCamCtrl.NET::MyCamera::MV_CC_SaveImageEx2 to convert the format of Bayer8/10/12/16 into RGB24/48, RGBA32/64, BGR24/48, or BGRA32/64.

- This API is available for the device, which supports the function.

4.5.9 MvCamCtrl.NET::MyCamera::MV_CC_SetBayerCLUTParam_NET

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

API Definition

```
public Int32 MV_CC_SetBayerCLUTParam_NET(  
    MV_CC_CLUT_PARAM    pstCLUTParam  
);
```

Parameters

pstCLUTParam

CLUT parameters structure. See ***MV_CC_CLUT_PARAM*** for details.

Return Value

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

Remarks

- After enabling CLUT and setting CLUT parameters, the parameters will take effect when you call API ***MvCamCtrl.NET::MyCamera::MV_CC_ConvertPixelFormat_NET*** or ***MvCamCtrl.NET::MyCamera::MV_CC_SaveImageEx2*** to convert the format of Bayer8/10/12/16 into RGB24/48, RGBA32/64, BGR24/48, or BGRA32/64.
- This API is available for the device, which supports the function.

4.5.10 MvCamCtrl.NET::MyCamera::MV_CC_ImageContrast_NET

Adjust image contrast.

API Definition

```
public Int32 MV_CC_ImageContrast_NET(  
    MV_CC_CONTRAST_PARAM    pstContrastParam  
);
```

Parameters

pstContrastParam

[IN] [OUT] Contrast parameter structure. See ***MV_CC_CONTRAST_PARAM*** for details.

Return Value

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

Remarks

This API is available for the device, which supports adjusting image contrast.

4.5.11 MvCamCtrl.NET::MyCamera::MV_CC_ImageSharpen_NET

Adjust image sharpness.

API Definition

```
public Int32 MV_CC_ImageSharpen_NET(  
    MV_CC_SHARPEN_PARAM    pstSharpenParam  
);
```

Parameters

pstSharpenParam

[IN] [OUT] Sharpness parameter structure. See *MV_CC_SHARPEN_PARAM* for details.

Return Value

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

Remarks

This API is available for the device which supports the function of adjusting image sharpness.

4.5.12 MvCamCtrl.NET::MyCamera::MV_CC_ColorCorrect_NET

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

API Definition

```
public Int32 MV_CC_ColorCorrect_NET(  
    MV_CC_COLOR_CORRECT_PARAM    pstColorCorrectParam  
);
```

Parameters

pstColorCorrectParam

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

Return Value

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

Remarks

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

4.5.13 MvCamCtrl.NET::MyCamera::MV_CC_NoiseEstimate

Estimate the noise.

API Definition

```
public Int32 MV_CC_NoiseEstimate_NET(  
    ref MV_CC_NOISE_ESTIMATE_PARAM pstNoiseEstimateParam  
)
```

Parameters

pstNoiseEstimateParam

[IN][OUT] Noise estimation parameters, see **MV_CC_NOISE_ESTIMATE_PARAM** for details

Return Value

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

Remarks

This API is available for the device, which supports noise estimation.

4.5.14 MvCamCtrl.NET::MyCamera::MV_CC_SpatialDenoise_NET

This API is used for spatial denoising.

API Definition

```
public Int32 MV_CC_SpatialDenoise_NET(  
    ref MV_CC_SPATIAL_DENOISE_PARAM pstSpatialDenoiseParam  
)
```

Parameters

pstSpatialDenoiseParam

Spatial denoising parameters, see **MV_CC_SPATIAL_DENOISE_PARAM** for details.

Return Value

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

Remarks

This API is available for the device, which supports spatial denoising.

4.5.15 MvCamCtrl.NET::MyCamera::MV_CC_LSCCalib_NET

This API is used for LSC calibration.

API Definition

```
public Int32 MV_CC_LSCCalib_NET(  
    MV_CC_LSC_CALIB_PARAM    pstLSCCalibParam  
);
```

Parameters

pstLSCCalibParam

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

Return Value

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

4.5.16 MvCamCtrl.NET::MyCamera::MV_CC_LSCCorrect_NET

This API is used for LSC correction.

API Definition

```
public Int32 MV_CC_LSCCorrect_NET(  
    MV_CC_LSC_CORRECT_PARAM    pstLSCCorrectParam  
);
```

Parameters

pstLSCCorrectParam

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

Return Value

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

4.5.17 MvCamCtrl.NET::MyCamera::MV_CC_HB_Declare_NET

Decode lossless compression stream into raw data.

API Definition

```
Public Int32 MV_CC_HB_Declare_NET(  
    MV_CC_HB_DECLARE_PARAM      pstDeclareParam  
);
```

Parameters

pstDeclareParam

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

Return Value

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

Remarks

This API should be supported by cameras.

4.5.18 MvCamCtrl.NET::MyCamera::MV_CC_RotateImage_NET

Rotate images in MONO8/RGB24/BGR24 format.

API Definition

```
public Int32 MV_CC_RotateImage_NET(  
    MV_CC_ROTATE_IMAGE_PARAM    pstRotateParam  
);
```

Parameters

pstRotateParam

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

Return Value

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

4.5.19 MvCamCtrl.NET::MyCamera::MV_CC_FlipImage_NET

Flip images in MONO8/RGB24/BGR24 format.

API Definition

```
public Int32 MV_CC_FlipImage_NET(  
    MV_CC_FLIP_IMAGE_PARAM    pstFlipParam  
);
```

Parameters

pstFlipParam

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

Return Value

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

4.5.20 MvCamCtrl.NET::MyCamera::MV_CC_StartRecord_NET

Start recording.

API Definition

```
int MV_CC_StartRecord_NET(  
    ref MyCamera.MV_CC_RECORD_PARAM    pstRecordParam  
);
```

Parameters

pstRecordParam

[IN] Video parameters, see the structure ***MV_CC_RECORD_PARAM*** for details.

Return Value

Return ***MyCamera.MV_OK (0)*** on success; return ***Error Code*** on failure.

Example

C#

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
using System.Threading;  
  
namespace GrabImage  
{  
    class GrabImage  
    {
```



```
static bool g_bExit = false;
static uint g_nPayloadSize = 0;
public static void ReceiveImageWorkThread(object obj)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = obj as MyCamera;
    MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
    IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);
    if (pData == IntPtr.Zero)
    {
        return;
    }
    uint nDataSize = g_nPayloadSize;
    MyCamera.MV_CC_INPUT_FRAME_INFO stInputFrameInfo = new MyCamera.MV_CC_INPUT_FRAME_INFO();
    while (true)
    {
        nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
        if (nRet == MyCamera.MV_OK)
        {
            Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
+ "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);
            stInputFrameInfo.pData = pData;
            stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
            nRet = device.MV_CC_InputOneFrame_NET(ref stInputFrameInfo);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet);
            }
        }
        else
        {
            Console.WriteLine("No data:{0:x8}", nRet);
        }
        if (g_bExit)
        {
            break;
        }
    }
}
static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    do
    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo;           // General device information
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detect the optimal network packet size (It works for GigE camera only)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
}
else
{
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
}
}
```

```
//Set triggering mode to Off

if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Get packet size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
g_nPayloadSize = stParam.nCurValue;
MyCamera.MV_CC_RECORD_PARAM stRecordPar = new MyCamera.MV_CC_RECORD_PARAM();
nRet = device.MV_CC_GetIntValue_NET("Width", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.nWidth = (ushort)stParam.nCurValue;
nRet = device.MV_CC_GetIntValue_NET("Height", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Height failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.nHeight = (ushort)stParam.nCurValue;
MyCamera.MVCC_ENUMVALUE stEnumValue = new MyCamera.MVCC_ENUMVALUE();
nRet = device.MV_CC_GetEnumValue_NET("PixelFormat", ref stEnumValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.enPixelType = (MyCamera.MvGvspPixelType)stEnumValue.nCurValue;
MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", ref stFloatValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Float value failed: nRet {0:x8}", 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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI;
stRecordPar.strFilePath = "./Recording.avi";
nRet = device.MV_CC_StartRecord_NET(ref stRecordPar);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet);
    break;
}

//Start acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}
Thread hReceiveImageThreadHandle = new Thread(ReceiveImageWorkThread);
hReceiveImageThreadHandle.Start(device);
Console.WriteLine("Press enter to exit");
Console.ReadKey();
g_bExit = true;
Thread.Sleep(1000);

//Stop acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Stop recording
nRet = device.MV_CC_StopRecord_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop Record failed{0:x8}", nRet);
    break;
}

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
```

```
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resource
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module GrabImage
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Do While (True)

            'Enumerate device
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
            If MyCamera.MV_OK <> nRet Then
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)
                Exit Do
            End If
            If (0 = stDeviceInfoList.nDeviceNum) Then
                Console.WriteLine("No Find Gige | Usb Device !")
                Exit Do
            End If
            Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
            stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
            If (0 = m_stDeviceInfoList.nDeviceNum) Then
```

```
    MsgBox("No Find Gige | Usb Device !")
    Return
End If

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

'Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Detect the optimal network packet size (It works for GigE camera only)
If stdevInfo.nLayerType = MyCamera.MV_GIGE_DEVICE Then
    Dim nPacketSize As Int32
    nPacketSize = dev.MV_CC_GetOptimalPacketSize_NET()
    If nPacketSize > 0 Then
        nRet = dev.MV_CC_SetIntValue_NET("GevSCSPPacketSize", nPacketSize)
        If 0 <> nRet Then
            Console.WriteLine("Warning: Set Packet Size failed:{0:x8}", nRet)
        End If
    Else
        Console.WriteLine("Warning: Get Packet Size failed:{0:x8}", nPacketSize)
    End If
End If

'Get packet size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim stRecordPar As MyCamera.MV_CC_RECORD_PARAM = New MyCamera.MV_CC_RECORD_PARAM()
nRet = dev.MV_CC_GetIntValue_NET("Width", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.nWidth = stParam.nCurValue
nRet = dev.MV_CC_GetIntValue_NET("Height", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Height failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.nHeight = stParam.nCurValue
```

```
Dim stEnumValue As MyCamera.MVCC_ENUMVALUE = New MyCamera.MVCC_ENUMVALUE()
nRet = dev.MV_CC_GetEnumValue_NET("PixelFormat", stEnumValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.enPixelFormat = stEnumValue.nCurValue
Dim stFloatValue As MyCamera.MVCC_FLOATVALUE = New MyCamera.MVCC_FLOATVALUE()
nRet = dev.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", stFloatValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Float value failed: nRet {0:x8}", nRet)
    Exit Do
End If

'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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI
stRecordPar.strFilePath = "./Recording.avi"
nRet = dev.MV_CC_StartRecord_NET(stRecordPar)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet)
    Exit Do
End If

'Start acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
Dim stImageInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX()
Dim pData As IntPtr
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
pData = Marshal.AllocHGlobal(nPayloadSize)
Dim nDataSize As UInt32
nDataSize = nPayloadSize
Dim stInputFrameInfo As MyCamera.MV_CC_INPUT_FRAME_INFO = New MyCamera.MV_CC_INPUT_FRAME_INFO()

'Acquire image
Do While True
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufferForDriver, nPayloadSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
```

```
Dim stSaveParam As MyCamera.MV_SAVE_IMAGE_PARAM_EX = New MyCamera.MV_SAVE_IMAGE_PARAM_EX()
stInputFrameInfo.pData = pData
stInputFrameInfo.nDataLen = stImageInfo.nFrameLen
nRet = dev.MV_CC_InputOneFrame_NET(stInputFrameInfo)
If 0 <> nRet Then
    Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet)
    Exit Do
End If
Else
    Console.WriteLine("Get one frame failed:{0:x8}", nRet)
End If
Loop
Marshal.FreeHGlobal(pBufForDriver)

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

'Stop recording
nRet = dev.MV_CC_StopRecord_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop record failed:{0:x8}", nRet)
    If 0 <> nRet Then
        Console.WriteLine("Open device failed:{0:x8}", nRet)
        Exit Do
    End If
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle and release resource
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()

End Sub
End Module
```


4.5.21 MvCamCtrl.NET::MyCamera::MV_CC_InputOneFrame_NET

Transmit video parameters.

API Definition

```
public int MV_CC_InputOneFrame_NET(  
    ref MyCamera.MV_CC_INPUT_FRAME_INFO    pstInputFrameInfo  
);
```

Parameters

pstInputFrameInfo

[IN] Video parameters, see the structure **MV_CC_INPUT_FRAME_INFO** for details.

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
using System.Threading;  
  
namespace GrabImage  
{  
    class GrabImage  
    {  
        static bool g_bExit = false;  
        static uint g_nPayloadSize = 0;  
        public static void ReceiveImageWorkThread(object obj)  
        {  
            int nRet = MyCamera.MV_OK;  
            MyCamera device = obj as MyCamera;  
            MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();  
            IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);  
            if (pData == IntPtr.Zero)  
            {  
                return;  
            }  
            uint nDataSize = g_nPayloadSize;  
            MyCamera.MV_CC_INPUT_FRAME_INFO stInputFrameInfo = new MyCamera.MV_CC_INPUT_FRAME_INFO();  
            while (true)  
            {
```

```
nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
if (nRet == MyCamera.MV_OK)
{
    Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
    + "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);
    stInputFrameInfo.pData = pData;
    stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
    nRet = device.MV_CC_InputOneFrame_NET(ref stInputFrameInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet);
    }
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}
if (g_bExit)
{
    break;
}
}
}

// Enumerate device
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}
MyCamera.MV_CC_DEVICE_INFO stDevInfo; //General device information
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
if (m_stDevList.nDeviceNum == 0)
{
    Console.WriteLine("no camera found!\n");
    return;
}

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
```

```
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detect the optimal network packet size (It works for GigE camera only)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

//Set triggering mode to Off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Get packet size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
g_nPayloadSize = stParam.nCurValue;
MyCamera.MV_CC_RECORD_PARAM stRecordPar = new MyCamera.MV_CC_RECORD_PARAM();
nRet = device.MV_CC_GetIntValue_NET("Width", ref stParam);
if (MyCamera.MV_OK != nRet)
{

```

```
        Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
        break;
    }
    stRecordPar.nWidth = (ushort)stParam.nCurValue;
    nRet = device.MV_CC_GetIntValue_NET("Height", ref stParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get Height failed: nRet {0:x8}", nRet);
        break;
    }
    stRecordPar.nHeight = (ushort)stParam.nCurValue;
    MyCamera.MVCC_ENUMVALUE stEnumValue = new MyCamera.MVCC_ENUMVALUE();
    nRet = device.MV_CC_GetEnumValue_NET("PixelFormat", ref stEnumValue);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
        break;
    }
    stRecordPar.enPixelFormat = (MyCamera.MvGvspPixelFormat)stEnumValue.nCurValue;
    MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
    nRet = device.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", ref stFloatValue);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get Float value failed: nRet {0:x8}", 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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI;
stRecordPar.strFilePath = "./Recording.avi";
nRet = device.MV_CC_StartRecord_NET(ref stRecordPar);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet);
    break;
}

//Start image acquisition
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}
Thread hReceiveImageThreadHandle = new Thread(ReceiveImageWorkThread);
hReceiveImageThreadHandle.Start(device);
```

```
Console.WriteLine("Press enter to exit");
Console.ReadKey();
g_bExit = true;
Thread.Sleep(1000);

//Stop image acquisition
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Stop recording
nRet = device.MV_CC_StopRecord_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop Record failed{0:x8}", nRet);
    break;
}

//Shut device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resource
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module GrabImage

Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)
        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Exit Do
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        If (0 = m_stDeviceInfoList.nDeviceNum) Then
            MsgBox("No Find Gige | Usb Device !")
            Return
        End If

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed:{0:x8}", nRet)
            Exit Do
        End If

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed:{0:x8}", nRet)
            Exit Do
        End If

        'Detect the optimal network packet size (It works for GigE camera only)
        If stdevInfo.nTLayerType = MyCamera.MV_GIGE_DEVICE Then
            Dim nPacketSize As Int32
            nPacketSize = dev.MV_CC_GetOptimalPacketSize_NET()
            If nPacketSize > 0 Then
                nRet = dev.MV_CC_SetIntValue_NET("GevSCSPPacketSize", nPacketSize)
                If 0 <> nRet Then
```

```
        Console.WriteLine("Warning: Set Packet Size failed:{0:x8}", nRet)
    End If
    Else
        Console.WriteLine("Warning: Get Packet Size failed:{0:x8}", nPacketSize)
    End If
End If

'Get packet size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim stRecordPar As MyCamera.MV_CC_RECORD_PARAM = New MyCamera.MV_CC_RECORD_PARAM()
nRet = dev.MV_CC_GetIntValue_NET("Width", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.nWidth = stParam.nCurValue
nRet = dev.MV_CC_GetIntValue_NET("Height", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Height failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.nHeight = stParam.nCurValue
Dim stEnumValue As MyCamera.MVCC_ENUMVALUE = New MyCamera.MVCC_ENUMVALUE()
nRet = dev.MV_CC_GetEnumValue_NET("PixelFormat", stEnumValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
    Exit Do
End If
stRecordPar.enPixelFormat = stEnumValue.nCurValue
Dim stFloatValue As MyCamera.MVCC_FLOATVALUE = New MyCamera.MVCC_FLOATVALUE()
nRet = dev.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", stFloatValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Float value failed: nRet {0:x8}", nRet)
    Exit Do
End If

'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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI
stRecordPar.strFilePath = "./Recording.avi"
nRet = dev.MV_CC_StartRecord_NET(stRecordPar)
```

```
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet)
    Exit Do
End If

'Start acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If
Dim pBufferForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)
Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
Dim stImageInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX()
Dim pData As IntPtr
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
pData = Marshal.AllocHGlobal(nPayloadSize)
Dim nDataSize As UInt32
nDataSize = nPayloadSize
Dim stInputFrameInfo As MyCamera.MV_CC_INPUT_FRAME_INFO = New MyCamera.MV_CC_INPUT_FRAME_INFO()

'Acquire image
Do While True
    nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufferForDriver, nPayloadSize, FrameInfo, 1000)
    If MyCamera.MV_OK = nRet Then
        Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
        Dim stSaveParam As MyCamera.MV_SAVE_IMAGE_PARAM_EX = New MyCamera.MV_SAVE_IMAGE_PARAM_EX()
        stInputFrameInfo.pData = pData
        stInputFrameInfo.nDataLen = stImageInfo.nFrameLen
        nRet = dev.MV_CC_InputOneFrame_NET(stInputFrameInfo)
        If 0 <> nRet Then
            Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet)
            Exit Do
        End If
    Else
        Console.WriteLine("Get one frame failed:{0:x8}", nRet)
    End If
Loop
Marshal.FreeHGlobal(pBufferForDriver)

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

'Stop recording
nRet = dev.MV_CC_StopRecord_NET()
If 0 <> nRet Then
```



```
    Console.WriteLine("Stop record failed:{0:x8}", nRet)
    Exit Do
End If

'Shut device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle and release resource
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()

End Sub
End Module
```

4.5.22 MvCamCtrl.NET::MyCamera::MV_CC_StopRecord_NET

Stop recording.

API Definition

```
int MV_CC_StopRecord_NET(
);
```

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;
namespace GrblImage
{
    class GrblImage
    {
        static bool g_bExit = false;
        static uint g_nPayloadSize = 0;
        public static void ReceivelImageWorkThread(object obj)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = obj as MyCamera;
            MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
            IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);
            if (pData == IntPtr.Zero)
            {
                return;
            }
            uint nDataSize = g_nPayloadSize;
            MyCamera.MV_CC_INPUT_FRAME_INFO stInputFrameInfo = new MyCamera.MV_CC_INPUT_FRAME_INFO();
            while (true)
            {
                nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
                if (nRet == MyCamera.MV_OK)
                {
                    Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
                    + "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);
                    stInputFrameInfo.pData = pData;
                    stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
                    nRet = device.MV_CC_InputOneFrame_NET(ref stInputFrameInfo);
                    if (MyCamera.MV_OK != nRet)
                    {
                        Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet);
                    }
                }
                else
                {
                    Console.WriteLine("No data:{0:x8}", nRet);
                }
                if (g_bExit)
                {
                    break;
                }
            }
        }
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
```

```
MyCamera device = new MyCamera();
do
{
    //Enumerate device
    MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
    nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }
    MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device information
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
    if (m_stDevList.nDeviceNum == 0)
    {
        Console.WriteLine("no camera found!\n");
        return;
    }

    //Create device
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        break;
    }

    //Open device
    nRet = device.MV_CC_OpenDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        break;
    }

    //Detect the optimal network packet size (It works for GigE camera only)
    if (stDevInfo.nLayerType == MyCamera.MV_GIGE_DEVICE)
    {
        int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
        if (nPacketSize > 0)
        {
            nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
            if (nRet != MyCamera.MV_OK)
            {
                Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
            }
        }
    }
}
```

```
    }
  }
  else
  {
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
  }
}

//Set triggering mode to Off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
  Console.WriteLine("Set TriggerMode failed!");
  break;
}

//Get packet size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
  Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
  break;
}
g_nPayloadSize = stParam.nCurValue;
MyCamera.MV_CC_RECORD_PARAM stRecordPar = new MyCamera.MV_CC_RECORD_PARAM();
nRet = device.MV_CC_GetIntValue_NET("Width", ref stParam);
if (MyCamera.MV_OK != nRet)
{
  Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
  break;
}
stRecordPar.nWidth = (ushort)stParam.nCurValue;
nRet = device.MV_CC_GetIntValue_NET("Height", ref stParam);
if (MyCamera.MV_OK != nRet)
{
  Console.WriteLine("Get Height failed: nRet {0:x8}", nRet);
  break;
}
stRecordPar.nHeight = (ushort)stParam.nCurValue;
MyCamera.MVCC_ENUMVALUE stEnumValue = new MyCamera.MVCC_ENUMVALUE();
nRet = device.MV_CC_GetEnumValue_NET("PixelFormat", ref stEnumValue);
if (MyCamera.MV_OK != nRet)
{
  Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
  break;
}
stRecordPar.enPixelType = (MyCamera.MvGvspPixelType)stEnumValue.nCurValue;
MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", ref stFloatValue);
if (MyCamera.MV_OK != nRet)
{
  Console.WriteLine("Get Float value failed: nRet {0:x8}", 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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI;
    stRecordPar.strFilePath = "./Recording.avi";
    nRet = device.MV_CC_StartRecord_NET(ref stRecordPar);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start Record failed: nRet {0:x8}", nRet);
        break;
    }

    //Start acquisition
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }
    Thread hReceivImageThreadHandle = new Thread(ReceivImageWorkThread);
    hReceivImageThreadHandle.Start(device);
    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
    g_bExit = true;
    Thread.Sleep(1000);

    //Stop acquisition
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    //Stop recording
    nRet = device.MV_CC_StopRecord_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop Record failed{0:x8}", nRet);
        break;
    }

    //Shut device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy handle and release resource
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);
if (MyCamera.MV_OK != nRet)
{
    //Destroy handle and release resource
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module GrabImage

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Do While (True)

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Exit Do
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
```

```
        Console.WriteLine("No Find Gige | Usb Device !")
    Exit Do
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
If (0 = m_stDeviceInfoList.nDeviceNum) Then
    MsgBox("No Find Gige | Usb Device !")
    Return
End If

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

'Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Detect the optimal network packet size (It works for GigE camera only)
If stdevInfo.nTLayerType = MyCamera.MV_GIGE_DEVICE Then
    Dim nPacketSize As Int32
    nPacketSize = dev.MV_CC_GetOptimalPacketSize_NET()
    If nPacketSize > 0 Then
        nRet = dev.MV_CC_SetIntValue_NET("GevSCSPPacketSize", nPacketSize)
        If 0 <> nRet Then
            Console.WriteLine("Warning: Set Packet Size failed:{0:x8}", nRet)
        End If
    Else
        Console.WriteLine("Warning: Get Packet Size failed:{0:x8}", nPacketSize)
    End If
End If

'Get packet size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
Dim nPayloadSize As Int32 = stParam.nCurValue
Dim stRecordPar As MyCamera.MV_CC_RECORD_PARAM = New MyCamera.MV_CC_RECORD_PARAM()
nRet = dev.MV_CC_GetIntValue_NET("Width", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
```

```
Exit Do
End If
stRecordPar.nWidth = stParam.nCurValue
nRet = dev.MV_CC_GetIntValue_NET("Height", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Height failed: nRet {0:x8}", nRet)
Exit Do
End If
stRecordPar.nHeight = stParam.nCurValue
Dim stEnumValue As MyCamera.MVCC_ENUMVALUE = New MyCamera.MVCC_ENUMVALUE()
nRet = dev.MV_CC_GetEnumValue_NET("PixelFormat", stEnumValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet)
Exit Do
End If
stRecordPar.enPixelFormat = stEnumValue.nCurValue
Dim stFloatValue As MyCamera.MVCC_FLOATVALUE = New MyCamera.MVCC_FLOATVALUE()
nRet = dev.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", stFloatValue)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get Float value failed: nRet {0:x8}", nRet)
Exit Do
End If

'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 = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI
stRecordPar.strFilePath = "./Recording.avi"
nRet = dev.MV_CC_StartRecord_NET(stRecordPar)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet)
Exit Do
End If

'Start acquisition
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
Exit Do
End If
Dim pBufForDriver As IntPtr = Marshal.AllocHGlobal(nPayloadSize)

Dim FrameInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX
Dim stImageInfo As MyCamera.MV_FRAME_OUT_INFO_EX = New MyCamera.MV_FRAME_OUT_INFO_EX()
Dim pData As IntPtr
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
pData = Marshal.AllocHGlobal(nPayloadSize)
```



```
Dim nDataSize As UInt32
nDataSize = nPayloadSize
Dim stInputFrameInfo As MyCamera.MV_CC_INPUT_FRAME_INFO = New
MyCamera.MV_CC_INPUT_FRAME_INFO()

'Acquire image
Do While True
nRet = dev.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, FrameInfo, 1000)
If MyCamera.MV_OK = nRet Then
    Console.WriteLine("Width:" + Convert.ToString(FrameInfo.nWidth) + " Height:" +
Convert.ToString(FrameInfo.nHeight) + " FrameNum:" + Convert.ToString(FrameInfo.nFrameNum))
    Dim stSaveParam As MyCamera.MV_SAVE_IMAGE_PARAM_EX = New
MyCamera.MV_SAVE_IMAGE_PARAM_EX()
    stInputFrameInfo.pData = pData
    stInputFrameInfo.nDataLen = stImageInfo.nFrameLen
    nRet = dev.MV_CC_InputOneFrame_NET(stInputFrameInfo)
    If 0 <> nRet Then
        Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet)
        Exit Do
    End If
Else
    Console.WriteLine("Get one frame failed:{0:x8}", nRet)
End If
Loop
Marshal.FreeHGlobal(pBufForDriver)

'Stop acquisition
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

'Stop recording
nRet = dev.MV_CC_StopRecord_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop record failed:{0:x8}", nRet)
    Exit Do
End If

'Shut device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

'Destroy handle and release resource
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
End If
```

```
Exit Do
Loop
If 0 <> nRet Then

    'Destroy handle and release resource
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If
Console.WriteLine("Press enter to exit")
System.Console.ReadLine()

End Sub
End Module
```

4.5.23 MvCamCtrl.NET::MyCamera::MV_CC_SaveImageToFile_NET

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

API Definition

```
Int32 MV_CC_SaveImageToFile_NET(
    ref MV_SAVE_IMG_TO_FILE_PARAM pstSaveFileParam
)
```

Parameters

pstSaveFileParam

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM stSaveFileParam = new MyCamera.MV_SAVE_IMG_TO_FILE_PARAM();
stSaveFileParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Tif;
stSaveFileParam.enPixelType = m_stFrameInfo.enPixelType;
stSaveFileParam.pData = m_BufForDriver;
stSaveFileParam.nDataLen = m_stFrameInfo.nFrameLen;
stSaveFileParam.nHeight = m_stFrameInfo.nHeight;
stSaveFileParam.nWidth = m_stFrameInfo.nWidth;
stSaveFileParam.iMethodValue = 0;
stSaveFileParam.plImagePath = "Image_w" + stSaveFileParam.nWidth.ToString() + "_h" +
stSaveFileParam.nHeight.ToString() + "_fn" + m_stFrameInfo.nFrameNum.ToString() + ".tif";
int nRet = m_MyCamera.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
```

```
if (MyCamera.MV_OK != nRet)
{
    ShowErrorMsg("Save Tiff Fail!", nRet);
    return;
}
```

Example

VB

```
Dim stSavePoCloudPar As MyCamera.MV_SAVE_IMG_TO_FILE_PARAM = New
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM
enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Tif;
stSaveFileParam.enPixelType = m_stFrameInfo.enPixelType;
stSaveFileParam.pData = Marshal.UnsafeAddrOfPinnedArrayElement(m_BufForDriver, 0)
stSaveFileParam.nDataLen = m_stFrameInfo.nFrameLen;
stSaveFileParam.nHeight = m_stFrameInfo.nHeight;
stSaveFileParam.nWidth = m_stFrameInfo.nWidth;
stSaveFileParam.iMethodValue = 0;
stSaveFileParam.plImagePath = "save.tif";
nRet = dev.MV_CC_SaveImageToFile_NET(stSaveFileParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Save Tiff Fail:{0:x8}", nRet)
End If
```

4.5.24 MvCamCtrl.NET::MyCamera::MV_CC_SavePointCloudData_NET

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

API Definition

```
Int32 MV_CC_SavePointCloudData_NET(
    ref MV_SAVE_POINT_CLOUD_PARAM pstPointDataParam
)
```

Parameters

pstPointDataParam

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

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
MyCamera.MV_SAVE_POINT_CLOUD_PARAM stSavePoCloudPar = new
MyCamera.MV_SAVE_POINT_CLOUD_PARAM();

stSavePoCloudPar.nLineNum = stOutFrame.stFrameInfo.nWidth * nImageNum;
```

```
stSavePoCloudPar.nLinePntNum = stOutFrame.stFrameInfo.nHeight;

byte[] bDstImageBuf = new byte[stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048];
uint nDstImageSize = stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048;

stSavePoCloudPar.enPointCloudFileType = MyCamera.MV_SAVE_POINT_CLOUD_FILE_TYPE.MV_PointCloudFile_PLY;
stSavePoCloudPar.enSrcPixelFormat = stOutFrame.stFrameInfo.enPixelFormat;
stSavePoCloudPar.pSrcData = Marshal.UnsafeAddrOfPinnedArrayElement(bSaveImageBuf, 0);
stSavePoCloudPar.nSrcDataLen = nSaveDataLen;
stSavePoCloudPar.pDstBuf = Marshal.UnsafeAddrOfPinnedArrayElement(bDstImageBuf, 0);
stSavePoCloudPar.nDstBufSize = nDstImageSize;

//Save point cloud data
nRet = device.MV_CC_SavePointCloudData_NET(ref stSavePoCloudPar);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Save point cloud data fail:{0:x8}", nRet);
    break;
}
```

Example

VB

```
Dim stSavePoCloudPar As MyCamera.MV_SAVE_POINT_CLOUD_PARAM = New
MyCamera.MV_SAVE_POINT_CLOUD_PARAM
stSavePoCloudPar.nLineNum = stOutFrame.stFrameInfo.nWidth * nImageNum
stSavePoCloudPar.nLinePntNum = stOutFrame.stFrameInfo.nHeight
Dim bDstImageBuf(stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048) As Byte
Dim nDstImageSize As Int32 = stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048
stSavePoCloudPar.enPointCloudFileType = MyCamera.MV_SAVE_POINT_CLOUD_FILE_TYPE.MV_PointCloudFile_PLY
stSavePoCloudPar.enSrcPixelFormat = stOutFrame.stFrameInfo.enPixelFormat
stSavePoCloudPar.pSrcData = Marshal.UnsafeAddrOfPinnedArrayElement(bSaveImageBuf, 0)
stSavePoCloudPar.nSrcDataLen = nSaveDataLen
stSavePoCloudPar.pDstBuf = Marshal.UnsafeAddrOfPinnedArrayElement(bDstImageBuf, 0)
stSavePoCloudPar.nDstBufSize = nDstImageSize

//Save point cloud data
nRet = device.MV_CC_SavePointCloudData_NET(stSavePoCloudPar)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Save point cloud data fail:{0:x8}", nRet)
End If
```

4.6 Advanced Settings

4.6.1 MvCamCtrl.NET::MyCamera::MV_GIGE_SetGvcPTimeout_NET

Set timeout for GVCp command.

API Definition

```
int MV_GIGE_SetGvcpTimeout_NET(
    uint    nMillisec
);
```

Parameters

nMillisec

[IN] Heartbeat time, unit: ms, range: [0,10000].

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

This API should be called after connecting to device.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace SetGvcpTimeout
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE;
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Enum device failed:{0:x8}", nRet);
                return;
            }
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
            if (0 == stDevList.nDeviceNum)
            {
                return;
            }

            MyCamera.MV_CC_DEVICE_INFO stDevInfo;

            //Change the device information structure pointer to device information structure
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
                typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
MyCamera device = new MyCamera();

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    return;
}

uint nMillisec = 100;//millisecond
nRet = device.MV_GIGE_SetGvcpTimeout_NET(nMillisec);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Gvcp Timeout failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
```

```
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Open device failed!")
        End If
        Console.WriteLine("Open device succeed!")

        Dim nMillisec As Int32 = 100 'Millisecond
        nRet = dev.MV_GIGE_SetGevSCSP_NET(nMillisec)
        If 0 <> nRet Then
            Console.WriteLine("Set Gvcp Timeout failed")
        End If

        '//Other process...

        'Close camera
        nRet = dev.MV_CC_CloseDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Close device failed!")
        End If
    End Sub
End Module
```

```
Console.WriteLine("Close device succeed!")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")
```

```
End Sub
```

```
End Module
```

4.6.2 MvCamCtrl.NET::MyCamera::MV_GIGE_GetGvcptimeout_NET

Get the GVCP command timeout.

API Definition

```
int MV_GIGE_GetGvcptimeout_NET(
    unit    pMillisec
);
```

Parameters

pMillisec

[IN] Timeout pointer, unit: millisecond, it is 500ms by default.

Return Value

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Grab_Callback
{
    class Grab_Callback
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
```



```
//Enumerate device
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

//Print device information
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{

```

```
nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Get the GVCP command timeout
UInt32 nTimeOut = 0;
nRet = MV_GIGE_GetGvcptimeout_NET(ref nTimeOut);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get gvcptimeout fail:{0:x8}", nRet);
    break;
}

//Get the number of retransmission GVCP commands
UInt32 nRetryGvcptime = 0;
nRet = device.MV_GIGE_GetRetryGvcptime_NET(ref nRetryGvcptime);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get retry gvcptime failed:{0:x8}", nRet);
    break;
}

//Close device
```

```
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
VB
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Grab_Callback
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim pBufForSaveImage As IntPtr
        Dim nBufForSaveImage As Int32
        Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte
        Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1

        Sub Main()
            Dim Info As String
            Dim nRet As Int32 = MyCamera.MV_OK
```

```
Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)

Do While (True)
    ' Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enum Device failed:{0:x8}", nRet)
        Return
    End If

    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device !")
        Return
    End If

    ' Print device information
    Dim i As Int32
    For i = 0 To stDeviceInfoList.nDeviceNum - 1
        Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
        stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
            Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
            Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
            Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
            stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
            Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
            Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
            Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
            Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

            Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
            Console.WriteLine(Info)
        Else
            Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
            Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
            Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
            stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
            Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
            Console.WriteLine(Info)
        End If
    Next

    Console.WriteLine("please select a device")
    Dim nIndex As Int32
```

```
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get the number of retransmission GVCP commands
Dim nRetryGvcptimes As UInt32
nRetryGvcptimes = 0;
nRet = dev.MV_GIGE_GetGvcptimeout_NET(nRetryGvcptimes)
If 0 <> nRet Then
    Console.WriteLine("Get retry gvcptimes failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.6.3 MvCamCtrl.NET::MyCamera::MV_GIGE_GetRetryGvcTimes_NET

Get the number of GVCP retransmission commands.

API Definition

```
int MV_GIGE_GetRetryGvcTimes_NET(
    unit pRetryGvcTimes
);
```

Parameters

pRetryGvcTimes

[IN] Retransmission times pointer, it values 3 by default.

Return Value

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Grab_Callback
{
    class Grab_Callback
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

                //Print device information
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
                {
                    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

                    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
                    {
                        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
                    }
                    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
```

```
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
    Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
}
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Get the number of retransmission GVCP commands
UInt32 nRetryGvcpTimes = 0;
nRet = device.MV_GIGE_GetRetryGvcpTimes_NET(ref nRetryGvcpTimes);
if (MyCamera.MV_OK != nRet)
{
```



```
        Console.WriteLine("Get retry gvcv times failed:{0:x8}", nRet);
        break;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Grab_Callback

```
Dim dev As MyCamera = New MyCamera
Dim pBufForSavImage As IntPtr
Dim nBufForSavImage As Int32
Dim m_bytImageBuffer(1024 * 1024 * 1) As Byte
Dim m_bytImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)

    Do While (True)
        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        ' Print device information
        Dim i As Int32
        For i = 0 To stDeviceInfoList.nDeviceNum - 1
            Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
            stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
            If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
                Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
                stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
                Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
                Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
                Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
                Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

                Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
                Console.WriteLine(Info)
            Else
                Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
                Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
                stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
                Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
                Console.WriteLine(Info)
            End If
        Next i
    End While
End Sub
```

```
        End If
    Next

    Console.WriteLine("please select a device")
    Dim nIndex As Int32
    Try
        nIndex = Console.ReadLine()
    Catch ex As Exception
        Console.WriteLine("Invalid input!")
        Console.WriteLine("push enter to exit")
        System.Console.ReadLine()
    End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get the number of retransmission GVCP commands
Dim nRetryGvcptimes As UInt32
nRetryGvcptimes = 0;
nRet = dev.MV_GIGE_GetRetryGvcptimes_NET(nRetryGvcptimes)
If 0 <> nRet Then
    Console.WriteLine("Get retry gvcptimes failed:{0:x8}", nRet)
```

```
Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.6.4 MvCamCtrl.NET::MyCamera::MV_GIGE_SetRetryGvcPtimes_NET

Set the GVCP command retransmission times.

API Definition

```
int MV_GIGE_SetRetryGvcPtimes_NET(
    unit  nRetryGvcPtimes
);
```

Parameters

nRetryGvcPtimes

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

Return Value

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

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET; namespace Grab_Callback
{
    class Grab_Callback
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

                //Print device information
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
                {
                    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

                    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
                    {
                        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
```

```
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
    uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
    uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
    uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
    uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
    Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
    Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Set the number of retry GVCP commands
nRet = device.MV_GIGE_SetRetryGvcptimes_NET(3);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set retry gvcptimes failed:{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed:{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

VB

Imports System.Runtime.InteropServices

```
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Grab_Callback
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim pBufForSaveImage As IntPtr
        Dim nBufForSaveImage As Int32
        Dim m_bytelImageBuffer(1024 * 1024 * 1) As Byte
        Dim m_bytelImageBufferLen As Int32 = 1024 * 1024 * 1

        Sub Main()
            Dim Info As String
            Dim nRet As Int32 = MyCamera.MV_OK
            Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
            Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)

            Do While (True)
                'Enumerate device
                nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
                If MyCamera.MV_OK <> nRet Then
                    Console.WriteLine("Enum Device failed:{0:x8}", nRet)
                    Return
                End If

                If (0 = stDeviceInfoList.nDeviceNum) Then
                    Console.WriteLine("No Find Gige | Usb Device !")
                    Return
                End If

                ' Print device information
                Dim i As Int32
                For i = 0 To stDeviceInfoList.nDeviceNum - 1
                    Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
                    stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
                    If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                        Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
                        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
                        Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
                        stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
                        Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
                        Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
                        Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
                        Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

                        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
                        Console.WriteLine(Info)
                    End If
                Next i
            End While
        End Sub
    End Sub
End Module
```



```
Else
    Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
    Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
    stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
    Console.WriteLine(Info)
End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
```

```
Exit Do
End If

' Set the number of retry GVCP commands
nRet = dev.MV_GIGE_SetRetryGvcpTimes_NET(3)
If 0 <> nRet Then
    Console.WriteLine("Set retry gvcp times failed:{0:x8}", nRet)
Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.6.5 MvCamCtrl.NET::MyCamera::MV_GIGE_SetDiscoveryMode_NET

Set device ACK packet type.

API Definition

```
public Int32 MV_GIGE_SetDiscoveryMode_NET(
    UInt32  nMode
);
```

Parameters

nMode

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

Return Value

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

Remarks

The API is supported only by GigE cameras.

4.6.6 MvCamCtrl.NET::MyCamera::MV_USB_GetTransferSize_NET

Get the packet size of USB3 vision device.

API Definition

```
int MV_USB_GetTransferSize_NET(  
    UInt32    nValue  
);
```

Parameters

nValue

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

Return Value

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

Example

C#

```
using System;  
using System.Collections.Generic;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace Grab_Callback  
{  
  
    class Grab_Callback  
    {  
        public static MyCamera.cbOutputExdelegate ImageCallback;  
        public static MyCamera device = new MyCamera();  
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)  
        {  
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
```

```
Convert.ToString(pFrameInfo.nHeight)
    + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    do
    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

        //Print device information
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
                Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
            }
        }
    }
}
```

```
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

UInt32 nTransferSize = 0;
// Get transfer size
nRet = device.MV_USB_GetTransferSize_NET(ref nTransferSize);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Warning: Get TransferSize failed:{0:x8}", nRet);
    break;
}

// Set trigger mode as off
```

```
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Register image callback
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}

// Start grabbing image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

// Stop grabbing
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);
```

```
if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Grab_Callback

```
Dim dev As MyCamera = New MyCamera
Dim pBufForSavelImage As IntPtr
Dim nBufForSavelImage As Int32
Dim m_bytelImageBuffer(1024 * 1024 * 1) As Byte
Dim m_bytelImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
Private Sub cbOutputdelegateFunc(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO,
ByVal pUser As IntPtr)
    Console.WriteLine("Width:" + Convert.ToString(pFrameInfo.nWidth) + " Height:" +
Convert.ToString(pFrameInfo.nHeight) + " FrameNum:" + Convert.ToString(pFrameInfo.nFrameNum))
End Sub
```

```
Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)
```

```
Do While (True)
    'Enumerate device
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enum Device failed:{0:x8}", nRet)
        Return
```

```
End If

If (0 = stDeviceInfoList.nDeviceNum) Then
    Console.WriteLine("No Find Gige | Usb Device !")
    Return
End If

' Print device information
Dim i As Int32
For i = 0 To stDeviceInfoList.nDeviceNum - 1
    Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
    stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
        GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
        Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
        Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
        stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
            MyCamera.MV_GIGE_DEVICE_INFO)
        Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
        Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
        Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
        Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
            nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
        Console.WriteLine(Info)
    Else
        Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
        Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
        stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
            MyCamera.MV_USB3_DEVICE_INFO)
        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
            stUsbInfo.chSerialNumber + "]"
        Console.WriteLine(Info)
    End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
```



```
        Console.WriteLine("push enter to exit")
        System.Console.ReadLine()
    End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
    End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

Dim nTransferSize As UInt32
nTransferSize = 0
' Get transfer size
nRet = dev.MV_USB_GetTransferSize_NET(nTransferSize)
If 0 <> nRet Then
    Console.WriteLine("Warning: Get TransferSize failed:{0:x8}", nRet)
    Exit Do
End If

nBufForSaveImage = stParam.nCurValue * 3 + 2048
pBufForSaveImage = Marshal.AllocHGlobal(nBufForSaveImage)

' Register image callback
nRet = dev.MV_CC_RegisterImageCallBack_NET(cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
    Exit Do
End If

' Start grabbing
```

```
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If

Console.WriteLine("push enter to exit")
System.Console.ReadLine()

' Stop grabbing
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing fauled:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device fauled:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device fauled:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device fauled:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module

End Sub
End Module
```

4.6.7 MvCamCtrl.NET::MyCamera::MV_USB_SetTransferSize_NET

Set the packet size of USB3 vision device.

API Definition

```
int MV_USB_SetTransferSize_NET(  
    UInt32    nValue  
);
```

Parameters

nValue

[IN] Packet size, the value is larger than or equal to 0x10000, the default value is 1.

Return Value

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

Remarks

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

Example

C#

```
using System;  
using System.Collections.Generic;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace Grab_Callback  
{  
  
    class Grab_Callback  
    {  
        public static MyCamera.cbOutputExdelegate ImageCallback;  
        public static MyCamera device = new MyCamera();  
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)  
        {  
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +  
Convert.ToString(pFrameInfo.nHeight)  
                + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);  
        }  
  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;
```

```
do
{
    //Enumerate device
    MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
    nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

    //Print device information
    for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
    {
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
```

```
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Set transfer size
nRet = device.MV_USB_SetTransferSize_NET(1024);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Warning: Set TransferSize failed:{0:x8}", nRet);
    break;
}

// Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Register image callback
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
```

```
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}

// Start grabbing image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

// Stop grabbing
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
```

```
    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
  }
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Grab_Callback

```
    Dim dev As MyCamera = New MyCamera
    Dim pBufferForSaveImage As IntPtr
    Dim nBufForSaveImage As Int32
    Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
    Private Sub cbOutputdelegateFunc(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO,
    ByVal pUser As IntPtr)
        Console.WriteLine("Width:" + Convert.ToString(pFrameInfo.nWidth) + " Height:" +
    Convert.ToString(pFrameInfo.nHeight) + " FrameNum:" + Convert.ToString(pFrameInfo.nFrameNum))
    End Sub
```

```
    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
    cbOutputdelegateFunc)
```

```
        Do While (True)
            'Enumerate device
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
    stDeviceInfoList)
            If MyCamera.MV_OK <> nRet Then
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)
                Return
            End If
```

```
            If (0 = stDeviceInfoList.nDeviceNum) Then
                Console.WriteLine("No Find Gige | Usb Device !")
                Return
            End If
```

```
            ' Print device information
            Dim i As Int32
```

```
For i = 0 To stDeviceInfoList.nDeviceNum - 1
    Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
    stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
    If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
        Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
        Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
        stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
        Dim nIpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
        Dim nIpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
        Dim nIpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
        Dim nIpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nIpByte1.ToString() + "." + nIpByte2.ToString() + "." + nIpByte3.ToString() + "." + nIpByte4.ToString() + "]"
        Console.WriteLine(Info)
    Else
        Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
        Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
        stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
        Console.WriteLine(Info)
    End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
```



```
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get payload size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
nBufForSaveImage = stParam.nCurValue * 3 + 2048
pBufForSaveImage = Marshal.AllocHGlobal(nBufForSaveImage)

' Set transfer size
nRet = dev.MV_USB_SetTransferSize_NET(1024)
If 0 <> nRet Then
    Console.WriteLine("Warning: Set TransferSize failed:{0:x8}", nRet)
    Exit Do
End If

' Register image callback
nRet = dev.MV_CC_RegisterImageCallBack_NET(cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
    Exit Do
End If

' Start grabbing
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If
```

```
Console.WriteLine("push enter to exit")
System.Console.ReadLine()

' Stop grabbing
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module

End Sub
End Module
```

4.6.8 MvCamCtrl.NET::MyCamera::MV_USB_GetTransferWays_NET

Get the number of transmission channels for USB3 vision device.

API Definition

```
int MV_USB_GetTransferWays_NET(  
    ref UInt32  nTransferWays  
);
```

Parameters

nTransferWays

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

Return Value

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

Remarks

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

Example

C#

```
using System;  
using System.Collections.Generic;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace Grab_Callback  
{  
  
    class Grab_Callback  
    {  
        public static MyCamera.cbOutputExdelegate ImageCallback;  
        public static MyCamera device = new MyCamera();  
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)  
        {  
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +  
Convert.ToString(pFrameInfo.nHeight)  
                + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);  
        }  
  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;  
            do  
            {  
                //Enumerate device  
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
```

```
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

    //Print device information
    for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
    {
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
            (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
            typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
            stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
            (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
            typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
            stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
    try
    {
        nDevIndex = Convert.ToInt32(Console.ReadLine());
    }
    catch
```

```
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

UInt32 nTransferWays = 0;
//Get the USB transfer ways
nRet = device.MV_USB_GetTransferWays_NET(ref nTransferWays);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Warning: Get transfer Ways failed:{0:x8}", nRet);
    break;
}

//Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Register image callback
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
}
```

```
        break;
    }

    //Start grabbing image
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadLine();

    //Stop grabbing image
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
```

```
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading.Thread  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET
```

Module Grab_Callback

```
    Dim dev As MyCamera = New MyCamera  
    Dim pBufForSaveImage As IntPtr  
    Dim nBufForSaveImage As Int32  
    Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte  
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
    Private Sub cbOutputdelegateFunc(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO,  
ByVal pUser As IntPtr)  
        Console.WriteLine("Width:" + Convert.ToString(pFrameInfo.nWidth) + " Height:" +  
Convert.ToString(pFrameInfo.nHeight) + " FrameNum:" + Convert.ToString(pFrameInfo.nFrameNum))  
    End Sub
```

```
    Sub Main()  
        Dim Info As String  
        Dim nRet As Int32 = MyCamera.MV_OK  
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST  
        Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf  
cbOutputdelegateFunc)
```

```
        Do While (True)  
            'Enumerate device  
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),  
stDeviceInfoList)  
            If MyCamera.MV_OK <> nRet Then  
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)  
                Return  
            End If
```

```
            If (0 = stDeviceInfoList.nDeviceNum) Then  
                Console.WriteLine("No Find Gige | Usb Device !")  
                Return  
            End If
```

```
            ' Print device information  
            Dim i As Int32  
            For i = 0 To stDeviceInfoList.nDeviceNum - 1  
                Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO  
                stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),  
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
    Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
    Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
    stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
    Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
    Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
    Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
    Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
    Console.WriteLine(Info)
Else
    Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
    Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
    stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
    Console.WriteLine(Info)
End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
```



```
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get Payload Size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
nBufForSavImage = stParam.nCurValue * 3 + 2048
pBufForSavImage = Marshal.AllocHGlobal(nBufForSavImage)

nBufForSavImage = stParam.nCurValue * 3 + 2048
pBufForSavImage = Marshal.AllocHGlobal(nBufForSavImage)

Dim nTransferWays As UInt32
nTransferWays = 0
' Set the USB transfer ways
nRet = dev.MV_USB_GetTransferWays_NET(nTransferWays)
If 0 <> nRet Then
    Console.WriteLine("Warning: Get Transfer ways failed:{0:x8}", nRet)
    Exit Do
End If

' Register image callback
nRet = dev.MV_CC_RegisterImageCallBack_NET(cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
    Exit Do
End If

' Start grabbing image
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If
```

```
Console.WriteLine("push enter to exit")
System.Console.ReadLine()

' Stop grabbing image
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module

End Sub

End Module
```

4.6.9 MvCamCtrl.NET::MyCamera::MV_USB_SetTransferWays_NET

Set the number of transmission channels for USB3 vision device.

API Definition

```
int MV_USB_FreelImageBuffer_NET(
    UInt32    nValue
);
```

Parameters

nValue

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

Return Value

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

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace Grab_Callback
{
    class Grab_Callback
    {
        public static MyCamera.cbOutputExdelegate ImageCallback;
        public static MyCamera device = new MyCamera();
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
                Convert.ToString(pFrameInfo.nHeight) + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
        }

        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
```

```
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

    //Print device information
    for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
    {
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
        typeof(MyCamera.MV_CC_DEVICE_INFO));

        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
            (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
            typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
            stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
            (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
            typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
            stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
    try
    {
        nDevIndex = Convert.ToInt32(Console.ReadLine());
    }
    catch
```

```
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

UInt32 nTransferWays = 3;
// Set the USB transfer ways
nRet = device.MV_USB_SetTransferWays_NET(nTransferWays);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Warning: Set Transfer ways failed:{0:x8}", nRet);
    break;
}

// Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Register image callback
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
}
```

```
        break;
    }

    // Start grabbing image
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadLine();

    // Stop grabbing
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    // Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
```

```
}  
}
```

Example

VB

```
Imports System.Runtime.InteropServices  
Imports System.Threading.Thread  
Imports System.Net.IPAddress  
Imports MvCamCtrl.NET
```

Module Grab_Callback

```
    Dim dev As MyCamera = New MyCamera  
    Dim pBufForSaveImage As IntPtr  
    Dim nBufForSaveImage As Int32  
    Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte  
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1
```

```
    Private Sub cbOutputdelegateFunc(ByVal pData As IntPtr, ByRef pFrameInfo As MyCamera.MV_FRAME_OUT_INFO,  
ByVal pUser As IntPtr)  
        Console.WriteLine("Width:" + Convert.ToString(pFrameInfo.nWidth) + " Height:" +  
Convert.ToString(pFrameInfo.nHeight) + " FrameNum:" + Convert.ToString(pFrameInfo.nFrameNum))  
    End Sub
```

```
    Sub Main()  
        Dim Info As String  
        Dim nRet As Int32 = MyCamera.MV_OK  
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST  
        Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf  
cbOutputdelegateFunc)
```

```
        Do While (True)  
            'Enumerate device  
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),  
stDeviceInfoList)  
            If MyCamera.MV_OK <> nRet Then  
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)  
                Return  
            End If
```

```
            If (0 = stDeviceInfoList.nDeviceNum) Then  
                Console.WriteLine("No Find Gige | Usb Device !")  
                Return  
            End If
```

```
            ' Print device information  
            Dim i As Int32  
            For i = 0 To stDeviceInfoList.nDeviceNum - 1  
                Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO  
                stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),  
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
    Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stGigeInfo, 0, stGigeInfoPtr, 216)
    Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
    stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
    Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
    Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
    Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
    Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)

    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
    Console.WriteLine(Info)
Else
    Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
    Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
    stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
    Console.WriteLine(Info)
End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
```



```
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get payload size
Dim stParam As MyCamera.MVCC_INTVALUE = New MyCamera.MVCC_INTVALUE()
nRet = dev.MV_CC_GetIntValue_NET("PayloadSize", stParam)
If (MyCamera.MV_OK <> nRet) Then
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet)
    Exit Do
End If
nBufForSaveImage = stParam.nCurValue * 3 + 2048
pBufForSaveImage = Marshal.AllocHGlobal(nBufForSaveImage)

Dim nTransferWays As UInt32
nTransferWays = 3
' Set the usb Transfer ways
nRet = dev.MV_USB_SetTransferWays_NET(nTransferWays)
If 0 <> nRet Then
    Console.WriteLine("Warning: Set Transfer ways failed:{0:x8}", nRet)
    Exit Do
End If

' Register image callback
nRet = dev.MV_CC_RegisterImageCallBack_NET(cbCallback, 0)
If MyCamera.MV_OK <> nRet Then
    Console.WriteLine("Register image callback failed:{0:x8}", nRet)
    Exit Do
End If

' Start grabbing
nRet = dev.MV_CC_StartGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Start grabbing fauled:{0:x8}", nRet)
    Exit Do
End If

Console.WriteLine("push enter to exit")
System.Console.ReadLine()
```

```
' Stop grabbing
nRet = dev.MV_CC_StopGrabbing_NET()
If 0 <> nRet Then
    Console.WriteLine("Stop Grabbing failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module

End Sub
End Module
```

4.7 Camera Internal APIs

4.7.1 MvCamCtrl.NET::MyCamera::MV_CC_LocalUpgrade_NET

Device local upgrade.

API Definition

```
int MV_CC_LocalUpgrade_NET(  
    string    pFilePathName  
);
```

Parameters

pFilePathName

[IN] Upgrade packet path, including the absolute path or relative path.

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Remarks

- Call this API to send the upgrade firmware to the device for upgrade. This API will wait 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.

Example

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
  
namespace Upgrade  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;  
            MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
  
            int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);  
            if (MyCamera.MV_OK != nRet)  
            {  
                Console.WriteLine("Enumerating device failed:{0:x8}", nRet);  
                return;  
            }  
  
            Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));  
            if (0 == stDevList.nDeviceNum)  
            {  
                return;  
            }  
  
            MyCamera.MV_CC_DEVICE_INFO stDevInfo;
```

```
//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_LocalUpgrade_NET("C://mv_digicap.dav");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Local Upgrade failed:{0:x8}", nRet);
    return;
}

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Module1

    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

        'Enumerate devices
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device.")
            Return
        End If

        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

        'Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Creating device handle failed.")
        End If
        Console.WriteLine("The device handle is created.")

        'Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
        If 0 <> nRet Then
            Console.WriteLine("Opening camera failed.")
        End If
        Console.WriteLine("The camera is open.")

        nRet = dev.MV_CC_LocalUpgrade_NET("C://mv_digicap.dav")
        If 0 <> nRet Then
            Console.WriteLine("Local Upgrade failed")
        End If

        '//Other process...

        'Close camera
    End Sub
End Module
```

```
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")
```

End Sub

End Module

4.7.2 MvCamCtrl.NET::MyCamera::MV_CC_GetUpgradeProcess_NET

Get the current upgrade progress.

API Definition

```
int MV_CC_GetUpgradeProcess_NET(
    ref uint    pnProcess
);
```

Parameters

pnProcess

[OUT] Current upgrade progress, percentage: from 0 to 100

Return Value

Return *MyCamera.MV_OK (0)* on success; return **Error Code** on failure.

Example

C#

```
using System;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;

namespace Upgrade
{
    class Program
    {
        static void Main(string[] args)
        {
            uint nLayerType = MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE;
```

```
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();

int nRet = MyCamera.MV_CC_EnumDevices_NET(nTLayerType, ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enumerating device failed:{0:x8}", nRet);
    return;
}

Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    return;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

//Change the device information structure pointer to device information structure
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
typeof(MyCamera.MV_CC_DEVICE_INFO));

MyCamera device = new MyCamera();

//Create device handle
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Creating device failed:{0:x8}", nRet);
    return;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Opening device failed:{0:x8}", nRet);
    return;
}

nRet = device.MV_CC_LocalUpgrade_NET("C://mv_digicap.dav");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Local Upgrade failed:{0:x8}", nRet);
    return;
}

uint nProcess = 0;
while (true)
{
    nRet = device.MV_CC_GetUpgradeProcess_NET(ref nProcess);
    if (nProcess == 100)
    {

```

```
        Console.WriteLine("Upgrade finished!");
        break;
    }
    Thread.Sleep(1000);    }

//Other process...

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Closing device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroying device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
```

Module Module1

```
Sub Main()
    Dim dev As MyCamera = New MyCamera
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST

    'Enumerate devices
    nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
    If MyCamera.MV_OK <> nRet Then
        Console.WriteLine("Enumerating device failed." + Convert.ToString(nRet))
        Return
    End If

    If (0 = stDeviceInfoList.nDeviceNum) Then
        Console.WriteLine("No Find Gige | Usb Device.")
        Return
    End If
End Sub
```



```
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

'Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Creating device handle failed.")
End If
Console.WriteLine("The device handle is created.")

'Open camera
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Opening camera failed.")
End If
Console.WriteLine("The camera is open.")

nRet = dev.MV_CC_LocalUpgrade_NET("C://mv_digicap.dav")
If 0 <> nRet Then
    Console.WriteLine("Local Upgrade failed")
End If

Dim nProcess = 0
While True
    nRet = dev.MV_CC_GetUpgradeProcess_NET(nProcess)
    If 0 <> nRet Then
        Console.WriteLine("Upgrade finished")
    End If
    Sleep(1000)
End While

//Other process...

'Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Closing camera failed.")
End If
Console.WriteLine("The camera is closed.")

'Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroying handle failed.")
End If
Console.WriteLine("The handle is destroyed.")

End Sub
```

End Module

4.7.3 MvCamCtrl.NET::MyCamera::MV_XML_GetGenICamXML_NET

Get the camera description file in XML format.

API Definition

```
int MV_XML_GetGenICamXML_NET(  
    IntPtr      pData,  
    uint        nDataSize,  
    ref uint    pnDataLen  
);
```

Parameters

pData

[IN][OUT] The XML file buffer address

nDataSize

[IN] The XML file buffer size

pnDataLen

[OUT] The XML file length

Return Value

Return *MyCamera.MV_OK (0)* on success; 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

C#

```
using System;  
using System.Runtime.InteropServices;  
using System.IO;  
using MvCamCtrl.NET;  
namespace GetXML  
{  
    class Program  
    {  
  
        static void Main(string[] args)
```

```
{
    uint nLayerType = MyCamera.MV_GIGE_DEVICE;
    MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
    int nRet = MyCamera.MV_CC_EnumDevices_NET(nLayerType, ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        return;
    }
    Console.WriteLine("The number of devices found: " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        return;
    }
    MyCamera.MV_CC_DEVICE_INFO stDevInfo;
    //Change the device information structure pointer to device information structure
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[0],
    typeof(MyCamera.MV_CC_DEVICE_INFO));
    MyCamera device = new MyCamera();

    //Create device
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        return;
    }
    //Open device
    nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Exclusive,0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        return;
    }

    uint nXmlBufSize = (1024 * 1024 * 3);
    //User should allocate the memory size according to requirement
    IntPtr pXmlBuf = Marshal.AllocHGlobal((int)nXmlBufSize);
    uint nXmlLen = 0;
    nRet = device.MV_XML_GetGenICamXML_NET(pXmlBuf,nXmlBufSize,ref nXmlLen);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get GenICam XML failed:{0:x8}", nRet);
        return;
    }

    Marshal.FreeHGlobal(pXmlBuf);

    //Other process...

    //Close device
}
```

```
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    return;
}

//Destroy handle and release resources
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}
}
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET
Module Module1
    Sub Main()
        Dim dev As MyCamera = New MyCamera
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        ' Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine ("Enumerating device failed!" + Convert.ToString(nRet))
            Return
        End If
        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find ABC Gige | Usb Device !")
            Return
        End If
        Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
        stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(0),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
        ' Create handle
        nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
        If 0 <> nRet Then
            Console.WriteLine("Create device failed!")
        End If
        Console.WriteLine("Create device succeed")
        ' Open camera
        nRet = dev.MV_CC_OpenDevice_NET()
```

```
If 0 <> nRet Then
    Console.WriteLine("Open device failed!")
End If
Console.WriteLine("Open device succeed!")
Dim nXmlBufSize As Int32 = (1024 * 1024 * 3)
' User should allocate the memory size according to requirement
Dim pXmlBuf As IntPtr = Marshal.AllocHGlobal(nXmlBufSize)
Dim nXmlLen As Int32 = 0
nRet = dev.MV_XML_GetGenICamXML_NET(pXmlBuf, nXmlBufSize, nXmlLen)
If 0 <> nRet Then
    Console.WriteLine("Get GenICam XML failed")
    Marshal.FreeHGlobal(pXmlBuf)
End If
Marshal.FreeHGlobal(pXmlBuf)

//Other process...
' Close camera
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Close device failed!")
End If
Console.WriteLine("Close device succeed!")
' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed!")
End If
Console.WriteLine("Destroy device succeed!")

End Sub
End Module
```

4.7.4 MvCamCtrl.NET::MyCamera::MV_XML_GetNodeInterfaceType_NET

Get the current node type.

API Definition

```
int MV_XML_GetNodeInterfaceType(
    const char          *pstrName,
    MyCamera.MV_XML_InterfaceType *pInterfaceType
);
```

Parameters

pstrName

[IN] Node name

pInterfaceType

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

Return Value

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

Remarks

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

Example

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.IO;
using MvCamCtrl.NET;
namespace Grab_Callback
{
    class Grab_Callback
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

                //Print device information
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
                {
                    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

                    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
```

```
{
    MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
    uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
    uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
    uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
    uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
    Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
    Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
}
else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
    Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
}
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}
```

```
//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Get current node type
MyCamera.MV_XML_InterfaceType stXmlInterfaceType = new MyCamera.MV_XML_InterfaceType();
nRet = device.MV_XML_GetNodeInterfaceType_NET("Width", ref stXmlInterfaceType);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get node Interface type failed:{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```


Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NETModule Grab_Callback

Dim dev As MyCamera = New MyCamera
Dim pBufForSaveImage As IntPtr
Dim nBufForSaveImage As Int32
Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte
Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1

Sub Main()
    Dim Info As String
    Dim nRet As Int32 = MyCamera.MV_OK
    Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
    Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)

    Do While (True)
        'Enumerate device
        nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
        If MyCamera.MV_OK <> nRet Then
            Console.WriteLine("Enum Device failed:{0:x8}", nRet)
            Return
        End If

        If (0 = stDeviceInfoList.nDeviceNum) Then
            Console.WriteLine("No Find Gige | Usb Device !")
            Return
        End If

        ' Print device information
        Dim i As Int32
        For i = 0 To stDeviceInfoList.nDeviceNum - 1
            Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
            stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
            If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
                Marshal.Copy(stDeviceInfo.SpecialInfo.stGigEInfo, 0, stGigeInfoPtr, 216)
                Dim stGigeInfo As MyCamera.MV_GIGE_DEVICE_INFO
                stGigeInfo = CType(Marshal.PtrToStructure(stGigeInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
                Dim nlpByte1 As UInt32 = (stGigeInfo.nCurrentIp And &HFF000000) >> 24
                Dim nlpByte2 As UInt32 = (stGigeInfo.nCurrentIp And &HFF0000) >> 16
                Dim nlpByte3 As UInt32 = (stGigeInfo.nCurrentIp And &HFF00) >> 8
                Dim nlpByte4 As UInt32 = (stGigeInfo.nCurrentIp And &HFF)
```

```
Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigeInfo.chUserDefinedName + "]IP[" +
nlpByte1.ToString() + "." + nlpByte2.ToString() + "." + nlpByte3.ToString() + "." + nlpByte4.ToString() + "]"
Console.WriteLine(Info)
Else
    Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
    Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
    Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
    stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
    Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
    Console.WriteLine(Info)
End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)

' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
```

```
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Get current node type
Dim stXmlInterfaceType As MyCamera.MV_XML_InterfaceType = New MyCamera.MV_XML_InterfaceType
nRet = dev.MV_XML_GetNodeInterfaceType_NET("Width", stXmlInterfaceType)
If 0 <> nRet Then
    Console.WriteLine("Get node Interface type failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub

End Module
```

4.7.5 MvCamCtrl.NET::MyCamera::MV_XML_GetNodeAccessMode_NET

Get current node access mode.

API Definition

```
int MV_XML_GetNodeAccessMode_NET(  
    string                *strKey  
    ref MyCamera.MV_XML_AccessMode  *stAccessMode  
);
```

Parameters

strKey

[IN] Node name

stAccessMode

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

Return Value

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

Remarks

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

Example

C#

```
using System;  
using System.Collections.Generic;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace Grab_Callback  
{  
  
    class Grab_Callback  
    {  
        public static MyCamera.cbOutputExdelegate ImageCallback;  
        public static MyCamera device = new MyCamera();  
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)  
        {  
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +  
Convert.ToString(pFrameInfo.nHeight)  
                + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);  
        }  
  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;  
            do  
            {
```

```
//Enumerate device
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
if (0 == stDevList.nDeviceNum)
{
    break;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;                //General device information

//Print device information
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{

```

```
nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

MyCamera.MV_XML_AccessMode stAccessMode = new MyCamera.MV_XML_AccessMode();
//Get the node access mode
nRet = device.MV_XML_GetNodeAccessMode_NET("Width", ref stAccessMode);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Warning: Get node access mode failed:{0:x8}", nRet);
    break;
}

//Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Register image callback
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
```

```
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}

//Start grabbing image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

//Stop grabbing image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}
```

```
    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
  }
}
```

Example

VB

```
Imports System.Runtime.InteropServices
Imports System.Threading.Thread
Imports System.Net.IPAddress
Imports MvCamCtrl.NET

Module Grab_Callback

    Dim dev As MyCamera = New MyCamera
    Dim pBufForSaveImage As IntPtr
    Dim nBufForSaveImage As Int32
    Dim m_byteImageBuffer(1024 * 1024 * 1) As Byte
    Dim m_byteImageBufferLen As Int32 = 1024 * 1024 * 1

    Sub Main()
        Dim Info As String
        Dim nRet As Int32 = MyCamera.MV_OK
        Dim stDeviceInfoList As MyCamera.MV_CC_DEVICE_INFO_LIST = New MyCamera.MV_CC_DEVICE_INFO_LIST
        Dim cbCallback As MyCamera.cbOutputdelegate = New MyCamera.cbOutputdelegate(AddressOf
cbOutputdelegateFunc)

        Do While (True)
            'Enumerate device
            nRet = dev.MV_CC_EnumDevices_NET((MyCamera.MV_GIGE_DEVICE Or MyCamera.MV_USB_DEVICE),
stDeviceInfoList)
            If MyCamera.MV_OK <> nRet Then
                Console.WriteLine("Enum Device failed:{0:x8}", nRet)
                Return
            End If

            If (0 = stDeviceInfoList.nDeviceNum) Then
                Console.WriteLine("No Find Gige | Usb Device !")
                Return
            End If

            'Print device information
            Dim i As Int32
            For i = 0 To stDeviceInfoList.nDeviceNum - 1
                Dim stDeviceInfo As MyCamera.MV_CC_DEVICE_INFO = New MyCamera.MV_CC_DEVICE_INFO
                stDeviceInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(i),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
                If (MyCamera.MV_GIGE_DEVICE = stDeviceInfo.nTLayerType) Then
                    Dim stGigeInfoPtr As IntPtr = Marshal.AllocHGlobal(216)
```



```
        Marshal.Copy(stDeviceInfo.SpecialInfo.stGigEInfo, 0, stGigEInfoPtr, 216)
        Dim stGigEInfo As MyCamera.MV_GIGE_DEVICE_INFO
        stGigEInfo = CType(Marshal.PtrToStructure(stGigEInfoPtr, GetType(MyCamera.MV_GIGE_DEVICE_INFO)),
MyCamera.MV_GIGE_DEVICE_INFO)
        Dim nIpByte1 As UInt32 = (stGigEInfo.nCurrentIp And &HFF000000) >> 24
        Dim nIpByte2 As UInt32 = (stGigEInfo.nCurrentIp And &HFF0000) >> 16
        Dim nIpByte3 As UInt32 = (stGigEInfo.nCurrentIp And &HFF00) >> 8
        Dim nIpByte4 As UInt32 = (stGigEInfo.nCurrentIp And &HFF)

        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stGigEInfo.chUserDefinedName + "]IP[" +
nIpByte1.ToString() + "." + nIpByte2.ToString() + "." + nIpByte3.ToString() + "." + nIpByte4.ToString() + "]"
        Console.WriteLine(Info)
    Else
        Dim stUsbInfoPtr As IntPtr = Marshal.AllocHGlobal(540)
        Marshal.Copy(stDeviceInfo.SpecialInfo.stUsb3VInfo, 0, stUsbInfoPtr, 540)
        Dim stUsbInfo As MyCamera.MV_USB3_DEVICE_INFO
        stUsbInfo = CType(Marshal.PtrToStructure(stUsbInfoPtr, GetType(MyCamera.MV_USB3_DEVICE_INFO)),
MyCamera.MV_USB3_DEVICE_INFO)
        Info = "DEV[" + Convert.ToString(i) + "] NAME[" + stUsbInfo.chUserDefinedName + "]Model[" +
stUsbInfo.chSerialNumber + "]"
        Console.WriteLine(Info)
    End If
Next

Console.WriteLine("please select a device")
Dim nIndex As Int32
Try
    nIndex = Console.ReadLine()
Catch ex As Exception
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End Try

If nIndex > stDeviceInfoList.nDeviceNum - 1 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

If nIndex < 0 Then
    Console.WriteLine("Invalid input!")
    Console.WriteLine("push enter to exit")
    System.Console.ReadLine()
End
End If

Dim stdevInfo As MyCamera.MV_CC_DEVICE_INFO
stdevInfo = CType(Marshal.PtrToStructure(stDeviceInfoList.pDeviceInfo(nIndex),
GetType(MyCamera.MV_CC_DEVICE_INFO)), MyCamera.MV_CC_DEVICE_INFO)
```

```
' Create handle
nRet = dev.MV_CC_CreateDevice_NET(stdevInfo)
If 0 <> nRet Then
    Console.WriteLine("Create device failed:{0:x8}", nRet)
    Exit Do
End If

' Open device
nRet = dev.MV_CC_OpenDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

Dim stAccessMode As MyCamera.MV_XML_AccessMode = New MyCamera.MV_XML_AccessMode()
' Get the node access mode
nRet = dev.MV_XML_GetNodeAccessMode_NET("Width", stAccessMode)
If 0 <> nRet Then
    Console.WriteLine("Warning: Get node access mode failed:{0:x8}", nRet)
    Exit Do
End If

' Close device
nRet = dev.MV_CC_CloseDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Open device failed:{0:x8}", nRet)
    Exit Do
End If

' Destroy handle
nRet = dev.MV_CC_DestroyDevice_NET()
If 0 <> nRet Then
    Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    Exit Do
End If

Exit Do
Loop

If 0 <> nRet Then
    ' Destroy handle
    nRet = dev.MV_CC_DestroyDevice_NET()
    If 0 <> nRet Then
        Console.WriteLine("Destroy device failed:{0:x8}", nRet)
    End If
End If

Console.WriteLine("Press enter to exit")
System.Console.ReadLine()
End Sub
```

End Module

Chapter 5 Data Structure and Enumeration

5.1 Data Structure

5.1.1 MV_ACTION_CMD_INFO

Command information structure

Structure Definition

```
public struct MV_ACTION_CMD_INFO{  
    public uint    bActionTimeEnable;  
    public long    nActionTime;  
    public uint    nDeviceKey;  
    public uint    nGroupKey;  
    public uint    nGroupMask;  
    public uint    nReserved;  
    public uint    nTimeOut;  
    public string  pBroadcastAddress;  
};
```

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 "ActionTimeEnable" is "1", it is related to the clock rate

pBroadcastAddress

Broadcast address

nTimeOut

ACK timeout, 0 indicates no need for acknowledgement

nReserved

Reserved.

5.1.2 MV_ACTION_CMD_RESULT

Returned information list of command

Structure Definition

```
public struct MV_ACTION_CMD_RESULT{
    public string      strDeviceAddress;
    public int         nStatus;
    public uint[]      nReserved;
};
```

Members

trDeviceAddress

Device IP address

nStatus

Status code

nReserved

Reserved.

See Also

MV_ACTION_CMD_RESULT_LIST

5.1.3 MV_ACTION_CMD_RESULT_LIST

Returned information list

Structure Definition

```
public struct MV_ACTION_CMD_RESULT_LIST{
    public uint      nNumResults;
    public IntPtr    pResults;
}
```

Members

nNumResults

The number of return values

pResults

Returned information of command

5.1.4 MV_ALL_MATCH_INFO

Different matching types

Structure Definition

```
public struct MV_ALL_MATCH_INFO{  
    public uint      nInfoSize;  
    public uint      nType;  
    public IntPtr    pInfo;  
}
```

Members

nInfoSize

Information type should be outputted

nType

Outputted information buffer, which requires the memory allocated by application layer

pInfo

Information buffer size.

Remarks

The corresponding output structures of pInfo are different as the information types are different, see as the following table:

nType macro definition	Value	Description	pInfo Structure
MV_MATCH_TYPE_ NET_DETECT	0x00000001	Network flow and packet loss information	<i>MV_MATCH_INFO_NET_DETECT</i>
MV_MATCH_TYPE_ USB_DETECT	0x00000002	Total byte number of USB3Vision camera received by host	<i>MV_MATCH_INFO_USB_DETECT</i>

5.1.5 MV_CamL_DEV_INFO

Structure of CameraLink device information

Structure Definition

```
struct _MV_CamL_DEV_INFO_{  
    unsigned char  chPortID[INFO_MAX_BUFFER_SIZE];  
    unsigned char  chModelName[INFO_MAX_BUFFER_SIZE];  
    unsigned char  chFamilyName[INFO_MAX_BUFFER_SIZE];
```

```
unsigned char  chDeviceVersion[INFO_MAX_BUFFER_SIZE];
unsigned char  chManufacturerName[INFO_MAX_BUFFER_SIZE];
unsigned char  chSerialNumber[INFO_MAX_BUFFER_SIZE];
unsigned int   nReserved[38];
}MV_CamL_DEV_INFO;
```

Members

chPortID

Port No.

chModelName

Model name

chFamilyName

Device family name

chDeviceVersion

Version No.

chManufacturerName

Manufacturer name

chSerialNumber

Serial No.

nReserved

Reserved byte

See Also

MV_CC_DEVICE_INFO

5.1.6 MV_CC_CCM_PARAM

Structure about Color Correction Parameters

Member	Data Type	Description
bCCMEnable	Boolean	Whether to enable color correction.
nCCMat	Int32[]	Color correction matrix, range: (-8192, 8192)
nRes	UInt32[]	Reserved.

5.1.7 MV_CC_CCM_PARAM_EX

CCM Parameter Structure

Member	Data Type	Description
bCCMEnable	public Boolean	Whether to enable CCM.
nCCMat	public Int32[]	Color correction matrix, range: (-65536,65536). The maximum length is 9 bytes.
nCCMScale	public UInt32	Quantitative scale (integer power of 2), the maximum value: 65536
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.8 MV_CC_CLUT_PARAM**CLUT Parameter Structure**

Member	Data Type	Description
bCLUTEnable	public Boolean	Whether to enable CLUT.
nCLUTScale	public UInt32	Quantitative scale (integer power of 2). The maximum value: 65536, recommended value: 1024
nCLUTSize	public UInt32	CLUT size, range: [17,33]. Recommended value: 17.
pCLUTBuf	public IntPtr	CLUT buffer
nCLUTBufLen	public UInt32	CLUT buffer length (nCLUTSize*nCLUTSize*nCLUTSize*sizeof(int)*3)
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.9 MV_CC_COLOR_CORRECT_PARAM

Structure about Color Correction Parameters

Member	Data Type	Description
nWidth	public UInt32	Image width
nHeight	public UInt32	Image height
pSrcBuf	public IntPtr	Input data buffer
nSrcBufLen	public UInt32	Input data length
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pDstBuf	public IntPtr	Output data buffer
nDstBufSize	public UInt32	Size of output buffer
nDstBufLen	public UInt32	Output data length
nImageBit	public UInt32	Image bit depth: 8, 10,12, or 16
stGammaParam	public <i>MV_CC_GAMMA_PARAM</i>	Gamma parameters
stCCMParam	public <i>MV_CC_CCM_PARAM_EX</i>	CCM parameters
stCLUTParam	public <i>MV_CC_CLUT_PARAM</i>	CLUT parameters
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.10 MV_CC_CONTRAST_PARAM**Contrast Parameter Structure**

Member	Data Type	Description
nWidth	public UInt32	Image width. Minimum value: 8.
nHeight	public UInt32	Image height. Minimum value: 8.
pSrcBuf	public IntPtr	Input data buffer
nSrcBufLen	public UInt32	Length of input data
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pDstBuf	public IntPtr	Output data buffer

Member	Data Type	Description
nDstBufSize	public UInt32	Size of the provided output buffer
nDstBufLen	public UInt32	Length of output data
nContrastFactor	public UInt32	Contrast. Range: [1,10000].
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.11 MV_CC_DEVICE_INFO

Device information structure

Structure Definition

```
public struct MV_CC_DEVICE_INFO{
    public uint          nMacAddrHigh;
    public uint          nMacAddrLow;
    public ushort        nMajorVer;
    public ushort        nMinorVer;
    public uint          nReserved;
    public uint          nTLayerType;
    public MyCamera.MV_CC_DEVICE_INFO.SPECIAL_INFO  SpecialInfo;
}
```

Members

nMacAddrHigh

MAC address high-bit

nMacAddrLow

MAC address low-bit

nMajorVer

Main version No.

nMinorVer

Sub version No.

nTLayerType

Transport layer types, see the definitions 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 deice
MV_CAMERALINK_DEVICE	0x00000008	CameraLink device

SpecialInfo

GigE and Usb3 device information structure, see ***SPECIAL_INFO*** for details.

nReserved

Reserved bytes

5.1.12 MV_CC_DEVICE_INFO_LIST

Device information list

Structure Definition

```
public struct MV_CC_DEVICE_INFO_LIST{
    public uint    nDeviceNum;
    public IntPtr  pDeviceInfo;
};
```

Members

nDeviceNum

The number of online devices

pDeviceInfo

Online device information, each array indicates one device, and up to 256 devices are supported.

5.1.13 MV_CC_FILE_ACCESS

Structure of saving and getting file

Structure Definition

```
public struct MV_CC_FILE_ACCESS{
    public uint    nReserved;
    public string  pDevFileName;
```

```
public string  pUserName;  
}
```

Members

nReserved

Reserved.

pDevFileName

Device file name

pUserName

User file name

5.1.14 MV_CC_FILE_ACCESS_PROGRESS

Structure about parameters loading progress

Structure Definition

```
public struct{  
    public long    nCompleted;  
    public long    nTotal;  
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]  
    public unit    nRes;  
}MV_CC_FILE_ACCESS_PROGRESS;
```

Members

nCompleted

Completed size

nTotal

Total size

nRes

Reserved.

5.1.15 MV_CC_FLIP_IMAGE_PARAM

Structure about Image Flipping

Member	Data Type	Description
enPixelFormat	<i>MvGvspPixelFormat</i>	Pixel format
nWidth	UInt32	Image length

Member	Data Type	Description
nHeight	UInt32	Image height
pSrcData	IntPtr	Buffer of input data
nSrcDataLen	UInt32	Size of input data
pDstBuf	IntPtr	Buffer of output data
nDstBufLen	UInt32	Size of output data
nDstBufSize	UInt32	Size of the output buffer
enFlipType	<i>MV_IMG_FLIP_TYPE</i>	Flip type
nRes	UInt32[]	Reserved.

5.1.16 MV_CC_FRAME_SPEC_INFO

Structure about Watermark Information

Member	Data Type	Description
nSecondCount	UInt32	Seconds
nCycleCount	UInt32	The number of cycles
nCycleOffset	UInt32	Cycle offset
fGain	Single	Gain
fExposureTime	UInt32	Exposure Time
nAverageBrightness	UInt32	Average brightness
nRed	UInt32	Red
nGreen	UInt32	Green
nBlue	UInt32	Blue
nFrameCounter	UInt32	The total number of frames
nTriggerIndex	UInt32	Trigger index
nInput	UInt32	Input
nOutput	UInt32	Output
nOffsetX	UInt16	Horizontal offset
nOffsetY	UInt16	Vertical offset
nFrameWidth	UInt16	Watermark width

Member	Data Type	Description
nFrameHeight	UInt16	Watermark height
nReserved	UInt32[]	Reserved.

5.1.17 MV_CC_GAMMA_PARAM

Gamma Parameter Structure

Member	Data Type	Description
enGammaType	public <i>MV_CC_GAMMA_TYPE</i>	Gamma type
fGammaValue	public Single	Gamma value, range: [0.1,4.0]
pGammaCurveBuf	public IntPtr	Gamma curve buffer
nGammaCurveBufLen	public UInt32	Size of gamma curve
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.18 MV_CC_HB_DECODE_PARAM

Structure about Lossless Decoding Parameters

Member	Data Type	Description
pSrcBuf	IntPtr	Buffer of input data
nSrcLen	UInt32	Size of input data
nWidth	UInt32	Image width
nHeight	UInt32	Image height
pDstBuf	IntPtr	Buffer of output data
nDstBufLen	UInt32	Size of output data
nDstBufSize	UInt32	Size of the output buffer
enDstPixelFormat	<i>MvGvspPixelFormat</i>	Pixel format
stFrameSpecInfo	<i>MV_CC_FRAME_SPEC_INFO</i>	Watermark information
nRes	UInt32[]	Reserved.

5.1.19 MV_CC_INPUT_FRAME_INFO

Structure about Video Data

Member	Data Type	Description
pData	IntPtr	Image data pointer
nDataLen	UInt32	Image size
nRes	UInt32[]	Reserved.

5.1.20 MV_CC_LSC_CALIB_PARAM

Structure about LSC Calibration Parameters

Member	Data Type	Description
nWidth	public UInt32	Image width. Range: [16,65535].
nHeight	public UInt32	Image height. Range: [16,65535].
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pSrcBuf	public IntPtr	Input data buffer
nSrcBufLen	public UInt32	Length of input data
pCalibBuf	public IntPtr	Output calibration data file buffer
nCalibBufSize	public UInt32	Size of the provided calibration data file buffer
nCalibBufLen	public UInt32	Length of calibration data file buffer
nSecNumW	public UInt32	Number of width sections
nSecNumH	public UInt32	Number of height sections
nPadCoef	public UInt32	Padding coefficient. Range: [1,5].
nCalibMethod	public UInt32	Calibration method: 0 (use the center as the reference)

Member	Data Type	Description
		1 (use the brightest area as the reference) 2 (adjust to the target brightness)
nTargetGray	public UInt32	Target brightness: 8bit. Range: [0,255] 10bit. Range: [0,1023] 12bit. Range: [0,4095] 16bit. Range: [0,65535]
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.21 MV_CC_LSC_CORRECT_PARAM



Structure about LSC Correction Parameters



Member	Data Type	Description
nWidth	public UInt32	Image width. Range: [16,65536].
nHeight	public UInt32	Image height. Range: [16,65536].
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pSrcBuf	public IntPtr	Input data buffer
nSrcBufLen	public UInt32	Length of input data
pDstBuf	public IntPtr	Output data buffer
nDstBufSize	public UInt32	Size of the provided output buffer
nDstBufLen	public UInt32	Length of output data
pCalibBuf	public IntPtr	Input calibration data file buffer

Member	Data Type	Description
nCalibBufLen	public UInt32	Length of input calibration data file buffer
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.22 MV_CC_NOISE_ESTIMATE_PARAM

Structure about Noise Estimation Parameters






Member	Data Type	Description
nWidth	public UInt32	Image width
nHeight	public UInt32	Image height
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pSrcBuf	public IntPtr	Buffer of input data
nSrcBufLen	public UInt32	Input data size
pstROIRect	public IntPtr	ROI information, see <i>MV_CC_RECT_I</i> for details.
nROINum	public UInt32	The number of ROIs
nNoiseThreshold	public UInt32	Noise threshold, range: [0,4095]  Note This node is the Bayer noise estimation parameter, and it is invalid for MONO8/RGB.
pNoiseProfile	public IntPtr	Output noise feature  Note This node is the Bayer noise estimation parameter, and it is invalid for MONO8/RGB.
nNoiseProfileSize	public UInt32	Output buffer size




Member	Data Type	Description
		 Note This node is the Bayer noise estimation parameter, and it is invalid for MONO8/RGB.
nNoiseProfileLen	public UInt32	Output noise feature length  Note This node is the Bayer noise estimation parameter, and it is invalid for MONO8/RGB.
nRes	public UInt32[]	Reserved. Maximum: 8 bytes.

5.1.23 MV_CC_SPATIAL_DENOISE_PARAM

Structure about Spatial Denoising Parameters

Member	Data Type	Description
nWidth	public UInt32	Image width
nHeight	public UInt32	Image height
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pSrcBuf	public IntPtr	Buffer of input data
nSrcBufLen	public UInt32	Input data size
pDstBuf	public IntPtr	Output denoised data
nDstBufSize	public UInt32	Buffer size of output data
nDstBufLen	public UInt32	Output denoised data length
pNoiseProfile	public IntPtr	Input noise features
nNoiseProfileLen	public UInt32	Input noise features length
nBayerDenoiseStrength	public UInt32	Denoising strength, range: [0,100].

Member	Data Type	Description
		 Note This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nBayerSharpenStrength	public UInt32	Sharpening strength, range: [0,32].  Note This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nBayerNoiseCorrect	public UInt32	Noise correction factor, range: [0,1280].  Note This node is the Bayer spatial denosing parameter, and it is invalid for MONO8/RGB.
nNoiseCorrectLum	public UInt32	Luminance correction factor, range: [1,2000].  Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nNoiseCorrectChrom	public UInt32	Hue correction factor, range: [1,2000].  Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthLum	public UInt32	Luminance denoising strength, range: [0,100].

Member	Data Type	Description
		 Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthChrom	public UInt32	Hue denoising strength, range: [0,100].  Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nStrengthSharpen	public UInt32	Sharpening strength, range: [1,1000].  Note This node is the MONO8/RGB spatial denosing parameter, and it is invalid for Bayer.
nRes	public UInt32[]	Reserved. Maximum: 8 bytes.

5.1.24 MV_CC_PIXEL_CONVERT_PARAM

Image transformation parameter structure

Structure Definition

```

public struct MV_PIXEL_CONVERT_PARAM{
    public MyCamera.MvGvspPixelType    enDstPixelType;
    public MyCamera.MvGvspPixelType    enSrcPixelType;
    public uint                        nDstBufferSize;
    public uint                        nDstLen;
    public ushort                      nHeight;
    public uint                        nRes;
    public uint                        nSrcDataLen;
    public ushort                      nWidth;
    public IntPtr                      pDstBuffer;
    public IntPtr                      pSrcData
}

```

Members

nWidth

Image width

nHeight

Image height

enSrcPixelFormat

Source pixel format, see definition in ***MvGvspPixelFormat*** .

pSrcData

Original image data

nSrcDataLen

Original image data size

enDstPixelFormat

Target pixel format, see definition in ***MvGvspPixelFormat*** .

pDstBuffer

Buffer for output data, save the transformed target data.

nDstLen

Transformed target data length

nDstBufferSize

Buffer size of output data

nRes

Reserved

Remarks

The supported input and output pixel formats after transforming 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	✓	✓	✓	✓	×	×

5.1.25 MV_CC_RECORD_PARAM

Video parameters

Structure Definition

```
public struct MV_CC_RECORD_PARAM{
    public MyCamera.MvGvspPixelType    enPixelFormat;
    public MyCamera.MV_RECORD_FORMAT_TYPE enRecordFmtType;
```

```
public float      fFrameRate;
public uint      nBitRate;
public ushort    nHeight;
public uint[]    nRes;
public ushort    nWidth;
public string    strFilePath;
};
```

Members

enPixelType

Pixel format

enRecordFmtType

Video format

fFrameRate

Frame rate

nBitRate

Bit rate

nHeight

Image height

nWidth

Image width

strFilePath

Video storage file

res

Reserved

5.1.26 MV_CC_RECT_I

Structure about ROI Rectangle Information

Member	Data Type	Description
nX	public UInt32	X-coordinate of rectangle upper left corner
nY	public UInt32	Y-coordinate of rectangle upper left corner
nWidth	public UInt32	Rectangle width
nHeight	public UInt32	Rectangle height

5.1.27 MV_CC_ROTATE_IMAGE_PARAM

Structure about Image Rotation

Member	Data Type	Description
enPixelFormat	<i>MvGvspPixelFormat</i>	Pixel format
nWidth	UInt32	Image width
nHeight	UInt32	Image height
pSrcData	IntPtr	Buffer of input data
nSrcDataLen	UInt32	Size of input data
pDstBuf	IntPtr	Buffer of output data
nDstBufLen	UInt32	Size of output data
nDstBufSize	UInt32	Size of the output buffer
enRotationAngle	<i>MV_IMG_ROTATION_ANGLE</i>	Rotation angle
nRes	UInt32[]	Reserved.

5.1.28 MV_CC_SHARPEN_PARAM

Sharpness Parameter Structure

Member	Data Type	Description
nWidth	public UInt32	Image width. Minimum value: 8.
nHeight	public UInt32	Image height. Minimum value: 8.
pSrcBuf	public IntPtr	Input data buffer
nSrcBufLen	public UInt32	Length of input data
enPixelFormat	public <i>MvGvspPixelFormat</i>	Pixel format
pDstBuf	public IntPtr	Output data buffer
nDstBufSize	public UInt32	Size of the provided output buffer
nDstBufLen	public UInt32	Length of output data

Member	Data Type	Description
nSharpenAmount	public UInt32	Sharpness. Range: [0,500].
nSharpenRadius	public UInt32	Radius of the adjustment area. Range: [1,21].
nSharpenThreshold	public UInt32	Sharpness threshold. Range: [0,255].
nRes	public UInt32[]	Reserved. The maximum length is 8 bytes.

5.1.29 MV_DISPLAY_FRAME_INFO

Image display structure

Structure Definition

```
public struct MV_DISPLAY_FRAME_INFO{  
    public IntPtr      hWnd;  
    public IntPtr      pData;  
    public UInt32      nDataLen;  
    public UInt16      nWidth;  
    public UInt16      nHeight;  
    public MvGvspPixelType enPixelFormat;  
    public UInt32[]    nReserved;  
}
```

Members

hWnd

Window handle

pData

Image data

nDataLen

Image data size

nWidth

Image width

nHeight

Image height

enPixelFormat

Image data pixel format, see definition in the enumeration type: ***MvGvspPixelFormat***

nReserved

Reserved

5.1.30 MV_EVENT_OUT_INFO

Event callback information

Structure Definition

```
public struct MV_EVENT_OUT_INFO{
    public string      EventName;
    public uint        nBlockIdHigh;
    public uint        nBlockIdLow;
    public uint        nEventDataSize;
    public ushort<     nEventID;
    public uint        nReserved;
    public ushort      nStreamChannel;
    public uint        nTimestampHigh;
    public uint        nTimestampLow;
    public IntPtr      pEventData;
}
```

Members

EventName

Event name

nEventID

Event No.

nStreamChannel

Stream channel No.

nBlockIdHigh

Frame No. high byte

nBlockIdLow

Frame No. low byte

nTimestampHigh

Timestamp high byte

nTimestampLow

Timestamp low byte

pEventData

Event data

nEventDataSize

Event data size

5.1.31 MV_FRAME_OUT

Structure of picture data and picture information

Structure Definition

```
public struct MV_FRAME_OUT{
    public IntPtr          pBufAddr;
    public MV_FRAME_OUT_INFO_EX  stFrameInfo;
    public UInt32[]        nReserved;
}
```

Members

pBufAddr

Picture data

stFrameInfo

Picture information, refer to *MV_FRAME_OUT_INFO_EX* for details.

nReserved

Reserved

5.1.32 MV_FRAME_OUT_INFO

Outputted frame information structure

Structure Definition

```
public struct MV_FRAME_OUT_INFO{
    public MyCamera.MvGvspPixelType  enPixelType;
    public uint                       nDevTimeStampHigh;
    public uint                       nDevTimeStampLow;
    public uint                       nFrameLen;
    public uint                       nFrameNum;
    public ushort                    nHeight;
    public long                      nHostTimeStamp;
    public uint                       nReserved;
    public uint                       nReserved0;
    public ushort                    nWidth;
}
```

Members

nWidth

Image width

nHeight

Image height

enPixelFormat

Pixel format, refer to *MvGvspPixelFormat* for details.

nFrameNum

Frame No.

nDevTimeStampHigh

Timestamp generated by camera, High 32 bits

nDevTimeStampLow

Timestamp generated by camera, low 32 bits

nReserved0

Reserved (8 bytes)

nHostTimeStamp

Timestamp generated by host

nFrameLen

Frame size

nReserved

Reserved

5.1.33 MV_FRAME_OUT_INFO_EX

Outputted frame information structure

Structure Definition

```
public struct MV_FRAME_OUT_INFO_EX{
    public MyCamera.MvGvspPixelFormat    enPixelFormat;
    public ushort                        fExposureTime;
    public ushort                        fGain;
    public uint                          nAverageBrightness;
    public uint                          nBlue;
    public uint                          nCycleCount;
    public uint                          nCycleOffset;
    public uint                          nDevTimeStampHigh;
    public uint                          nDevTimeStampLow;
    public uint                          nFrameCounter;
    public uint                          nFrameLen;
    public uint                          nFrameNum;
    public uint                          nGreen;
    public ushort                        nHeight;
    public long                          nHostTimeStamp;
    public uint                          nInput;
    public uint                          nLostPacket;
```

```
public ushort      nOffsetX;
public ushort      nOffsetY;
public uint         nOutput;
public uint         nRed;
public uint         nReserved;
public uint         nReserved0;
public uint         nSecondCount;
public uint         nTriggerIndex;
public ushort      nWidth;
}
```

Members

nWidth

Image width

nHeight

Image height

enPixelFormat

Pixel format, refer to *MvGvspPixelFormat* for details.

nFrameNum

Frame No

.

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 125us, reset in every 1 second

nCycleOffset

Clock period offset, reset in every 125us

fGain

Gain

fExposureTime

Exposure time

nAverageBrightness

Average brightness

nRed

WB red

nGreen

WB green

nBlue

WB blue

nFrameCounter

The number of frames

nTriggerIndex

Trigger counting

nInput

Line input

nOutput

Line output

nLostPacket

The number of lost packets

nOffsetX

X value of ROI area offset

nOffsetY

Y value of ROI area offset

nReserved

Reserved

5.1.34 MV_GENTL_DEV_INFO

Structure about information of device enumerated via GenTL

Structure Definition

```
public struct{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
    public string    chInterfaceID;
```

```
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chDeviceID;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chVendorName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chModelName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chTLType;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chDisplayName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chUserDefinedName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chSerialNumber;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
public string    chDeviceVersion;
public UInt32    nCtlIndex;
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
public UInt32[]  nReserved;
}MV_GENTL_DEV_INFO;
```

Members

chInterfaceID

Interface ID

chDeviceID

Device ID

chVendorName

Vendor name

chModelName

Model name

chTLType

Transport layer type

chDisplayName

Device name

chUserDefinedName

User defined name

chSerialNumber

Serial number

chDeviceVersion

Device version

nCtlIndex

CTI file index

nReserved

Reserved.

5.1.35 MV_GENTL_DEV_INFO_LIST

Structure about list of devices enumerated via GenTL

Structure Definition

```
public struct{
    public UInt32    nDeviceNum;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = MV_MAX_GENTL_DEV_NUM)]
    public IntPtr[]  pDeviceInfo;
}MV_GENTL_DEV_INFO_LIST;
```

Members

nDeviceNum

The number of online devices

pDeviceInfo

Device information, up to 256 devices are supported.

5.1.36 MV_GENTL_IF_INFO

Structure about information of interface enumerated via GenTL

Structure Definition

```
public struct{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
    public string    chInterfaceID;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
    public string    chTLType;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = INFO_MAX_BUFFER_SIZE)]
    public string    chDisplayName;
    public UInt32    nCtlIndex;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
    public UInt32    nReserved;
}MV_GENTL_IF_INFO;
```

Members

chInterfaceID

Interface ID

chTLType

Transport layer type

chDisplayName

Device name

nCtlIndex

CTI file index

nReserved

Reserved.

5.1.37 MV_GENTL_IF_INFO_LIST

Structure about list of interfaces enumerated via GenTL

Structure Definition

```
public struct{
    public UInt32      nInterfaceNum;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = MV_MAX_GENTL_IF_NUM)]
    public IntPtr[]    plFInfo;
}MV_GENTL_IF_INFO_LIST;
```

Members

nInterfaceNum

The number of interfaces.

plFInfo

The interface information, up to 256 interfaces are supported.

5.1.38 MV_GIGE_DEVICE_INFO

GIGE device information structure

Structure Definition

```
public struct MV_GIGE_DEVICE_INFO{
    public string  chDeviceVersion;
    public string  chManufacturerName;
    public string  chManufacturerSpecificInfo;
    public string  chModelName;
    public string  chSerialNumber;
    public string  chUserDefinedName;
    public uint    nCurrentIp;
    public uint    nCurrentSubNetMask;
    public uint    nDefaultGateWay;
```

```
public uint nIpCfgCurrent;  
public uint nIpCfgOption;  
public uint nNetExport;  
public uint nReserved;  
}
```

Members

nIpCfgOption

IP configuration options

nIpCfgCurrent

Current IP configuration

nCurrentIp

Current device IP

nCurrentSubNetMask

Current subnet mask

nDefaultGateWay

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 bytes

5.1.39 MV_IMAGE_BASIC_INFO

Basic image information

Structure Definition

```
public struct MV_IMAGE_BASIC_INFO{
    public uint    enPixelList;
    public uint    enPixelFormat;
    public float   fFrameRateMax;
    public float   fFrameRateMin;
    public float   fFrameRateValue;
    public uint    nHeightInc;
    public uint    nHeightMax;
    public uint    nHeightMin;
    public uint    nHeightValue;
    public uint    nReserved;
    public uint    nSupportedPixelFormatNum;
    public uint    nWidthInc;
    public uint    nWidthMax;
    public ushort  nWidthMin;
    public ushort  nWidthValue;
}
```

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

enPixelFormat

Pixel format, see detailed definitions in *MvGvspPixelFormat*

nSupportedPixelFormatNum

Supported pixel format type

enPixelFormatList

Supported pixel format list, see detailed definition in *MvGvspPixelFormat*

nReserved

Reserved, set to 0

5.1.40 MV_MATCH_INFO_NET_DETECT

Network flow and packet loss information

API Definition

```
public struct MV_MATCH_INFO_NET_DETECT{
    public int64      nReviceDataSize;
    public int64      nLostPacketCount;
    public uint32     nLostFrameCount;
    public uint32     nNetRecvFrameCount;
    public int64      nRequestResendPacketCount;
    public int64      nResendPacketCount;
}
```

Parameters**nReviceDataSize**

Received data size[data statistics between StartGrabbing and StopGrabbing]

nLostPacketCount

Packet loss number

nLostFrameCount

Number of frame loss

nNetRecvFrameCount

Received frame number

nRequestResendPacketCount

The number of packets, which are requested to resend

nResendPacketCount

The number of resent packets

5.1.41 MV_MATCH_INFO_USB_DETECT

Total number of bytes host received from USB3 vision camera

Structure Definition

```
public struct MV_MATCH_INFO_USB_DETECT{
    public uint    nErrorFrameCount;
    public uint    nReserved;
    public uint    nReviceDataSize;
    public uint    nRevicedFrameCount;
}
```

Members

nErrorFrameCount

The number of error frames

nReserved

Reserved

nReviceDataSize

Received data size

nRevicedFrameCount

The number of received frames

5.1.42 MV_NETTRANS_INFO

Network transport information structure

Structure Definition

```
public struct MV_NETTRANS_INFO{
    public int64    nReviceDataSize;
    public int32    nThrowFrameCount;
    public uint32    nNetRecvFrameCount;
    public int64    nRequestResendPacketCount;
    public int64    nResendPacketCount;
}
```

Members

nReviceDataSize

Received data size

nThrowFrameCount

The number of lost frames

nNetRecvFrameCount

Received frame number

nRequestResendPacketCount

The number of packets, which are requested to resend

nResendPacketCount

The number of resent packets

5.1.43 MV_RECORD_FORMAT_TYPE

Video format definition

Structure Definition

```
public enum MV_RECORD_FORMAT_TYPE{
    MV_FormatType_Undefined  = 0,
    MV_FormatType_AVI        = 1,
};
```

Members

MV_FormatType_Undefined

Undefined format

MV_FormatType_AVI

AVI format

5.1.44 MV_SAVE_IMAGE_PARAM

Parameter structure for transforming picture format

Structure Definition

```
public struct MV_SAVE_IMAGE_PARAM{
    public MyCamera.MV_SAVE_IAMGE_TYPE  enImageType;
    public MyCamera.MvGvspPixelType     enPixelFormat;
    public uint                          nBufferSize;
    public uint                          nDataLen;
    public ushort                        nHeight;
    public uint                          nImageLen;
    public ushort                        nWidth;
    public IntPtr                        pData;
    public IntPtr                        pImageBuffer;
}
```

Members

enImageType

Output picture format, see value definition in *MV_SAVE_IMAGE_TYPE* .

enPixelType

Pixel format of original image data, see value definition in *MvGvspPixelFormat* .

nBufferSize

Output data buffer area size

nDataLen

Original image data length

nHeight

Resolution of image height

nImageLen

Transformed picture data length

nWidth

Resolution of image width

pData

Original image data

pImageBuffer

Output data buffer area, used for storing transformed picture data

5.1.45 MV_SAVE_IMAGE_PARAM_EX

Structure of picture format transformation parameters

Structure Definition

```
public struct MV_SAVE_IMAGE_PARAM_EX{
    public MyCamera.MV_SAVE_IMAGE_TYPE    enImageType;
    public MyCamera.MvGvspPixelFormat      enPixelType;
    public uint                            iMethodValue;
    public uint                            nBufferSize;
    public uint                            nDataLen;
    public ushort                          nHeight;
    public uint                            nImageLen;
    public uint                            nJpgQuality;
    public uint                            nReserved;
    public ushort                          nWidth;
    public IntPtr                          pData;
    public IntPtr                          pImageBuffer;
}
```

Members

enImageType

Output picture format, see value definition in ***MV_SAVE_IMAGE_TYPE*** .

enPixelType

Pixel format of original image data, see value definition in ***MvGvspPixelType*** .

iMethodValue

Transmission mode

nBufferSize

Output data buffer area size

nDataLen

Original image data length

nHeight

Resolution of image height

nImageLen

Transformed picture data length

nJpgQuality

Encoding quality, range from 50 to 99

nReserved

Reserved

nWidth

Resolution of image width

pData

Original image data

pImageBuffer

Output data buffer area, used for storing transformed picture data

5.1.46 MV_SAVE_IMG_TO_FILE_PARAM

Structure about image saving parameters.

Structure Definition

```
public struct{
    public MvGvspPixelType    enPixelType;
    public IntPtr            pData;
    public UInt32            nDataLen;
    public UInt16            nWidth;
    public UInt16            nHeight;
```



```
public MV_SAVE_IAMGE_TYPE  enImageType;
public UInt32              nQuality;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
public string              pImagePath;
public UInt32              iMethodValue;
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
public UInt32[]            nRes;
}MV_SAVE_IMG_TO_FILE_PARAM;
```

Members

enPixelFormat

The pixel format of the input data, see the enumeration definition *MvGvspPixelFormat* for details.

pData

Input data buffer

nDataLen

Input data size

nWidth

Image width

nHeigh

Image height

enImageType

Input image format, see the enumeration definition *MV_SAVE_IAMGE_TYPE* for details.

nQuality

Encoding quality: (0-100]

pImagePath

Input file path

iMethodValue

Interpolation method of converting Bayer to RGB24: 0-nearest neighbor 1-bilinearity 2-Hamilton

nRes

Reserved.

5.1.47 MV_SAVE_POINT_CLOUD_PARAM

Structure about parameters of saving 3D point cloud data

Structure Definition

```
public struct{
    public UInt32          nLinePntNum;
    public UInt32          nLineNum;
```

```
public MvGvspPixelType      enSrcPixelType;
public IntPtr              pSrcData;
public UInt32              nSrcDataLen;
public IntPtr              pDstBuf;
public UInt32              nDstBufSize;
public UInt32              nDstBufLen;
public MV_SAVE_POINT_CLOUD_FILE_TYPE enPointCloudFileType;
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 8)]
public UInt32[]            nRes;
}MV_SAVE_POINT_CLOUD_PARAM;
```

Members

nLinePntNum

The number of points in each row, which is the image width

nLineNum

The number of rows, which is the image height

enSrcPixelType

The pixel format of the input data, see the enumeration definition *MvGvspPixelFormat* for details.

pSrcData

Input data buffer

nSrcDataLen

Input data size

pDstBuf

Output pixel data buffer

nDstBufSize

Provided output buffer size, the value is (nLinePntNum * nLineNum * (16*3 + 4) + 2048)

nDstBufLen

Buffer size of output pixel data

enPointCloudFileType

Provided file type of output point data, see the enumeration definition *MV_SAVE_POINT_CLOUD_FILE_TYPE* for details.

nRes

Reserved.

5.1.48 MV_TRANSMISSION_TYPE_NET

Transmission mode, including single cast mode, multicast mode, and so on.

Structure Definition

```
struct _MV_TRANSMISSION_TYPE_NET_T{
    MV_GIGE_TRANSMISSION_TYPE    enTransmissionType;
    unsigned int                  nDestIp;
    unsigned int                  nDestPort;
    unsigned int                  nReserved[32];
}MV_TRANSMISSION_TYPE;
```

Members

nDestIp

Target IP, it is valid under multicast

nDestPort

It is valid under multicast

nCurrentSubNetMask

Current subnet mask

nReserved

Reserved

enTransmissionType

Transmission mode, see the definitions below:

Macro Definition	Value	Description
MV_GIGE_TRANSTYPE_UNICAST	0	Unicast
MV_GIGE_TRANSTYPE_MULTICAST	1	Multicast
MV_GIGE_TRANSTYPE_LIMITEDBROADCAST	2	LAN broadcast
MV_GIGE_TRANSTYPE_SUBNETBROADCAST	3	Subnet broadcast
MV_GIGE_TRANSTYPE_CAMERADEFINED	4	Get from camera
MV_GIGE_TRANSTYPE_UNICAST_DEFINED_PORT	5	Port No. of getting image data
MV_GIGE_TRANSTYPE_UNICAST_WITHOUT_RECV	00010000	Unicast mode, but not receive image data
MV_GIGE_TRANSTYPE_MULTICAST_WITHOUT_RECV	00010001	Multiple mode, but not receive image data

5.1.49 MV_USB3_DEVICE_INFO

USB device information structure

Structure Definition

```
public struct MV_USB3_DEVICE_INFO{
    public string      chDeviceGUID;
    public string      chDeviceVersion;
    public string      chFamilyName;
    public string      chManufacturerName;
    public string      chModelName;
    public string      chSerialNumber;
    public string      chUserDefinedName;
    public string      chVendorName;
    public byte        CrtlInEndPoint;
    public byte        CrtlOutEndPoint;
    public byte        EventEndPoint;
    public ushort      idProduct;
    public ushort      idVendor;
    public uint         nbcdUSB;
    public uint         nDeviceNumber;
    public uint         nReserved;
    public byte        StreamEndPoint;
}
```

Members

CrtlInEndPoint

Control input port

CrtlOutEndPoint

Control output port

StreamEndPoint

Stream port

EventEndPoint

Event port

idVendor

Supplier ID

idProduct

Product 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

5.1.50 MV_XML_FEATURE_Boolean

IBoolean node information structure

Structure Definition

```
public struct MV_XML_FEATURE_Boolean{
    public int          bIsLocked;
    public bool         bValue;
    public MyCamera.MV_XML_AccessMode enAccessMode;
    public MyCamera.MV_XML_Visibility enVisivility;
    public uint         nReserved;
    public string       strDescription;
    public string       strDisplayName;
    public string       strName;
    public string       strToolTip;
}
```

Members

bIsLocked

Lock or not (not support, reserved): 0- unlock, 1- lock

bValue

Current value

enAccessMode

Accessing mode, see value definitions in Enumeration Type: ***MV_XML_AccessMode***

enVisivility

Visible or not, see value definitions in Enumeration Type: ***MV_XML_Visibility***

nReserved

Reserved.

strDescription

Node description, it is not supported, reserved

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

5.1.51 MV_XML_FEATURE_Integer

Integer node information structure

Structure Definition

```
public struct MV_XML_FEATURE_Integer{
    public int          bIsLocked;
    public MyCamera.MV_XML_AccessMode  enAccessMode;
    public MyCamera.MV_XML_Visibility  enVisivility;
    public long         nIncrement;
    public long         nMaxValue;
    public long         nMinValue;
    public uint         nReserved;
    public long         nValue;
    public string       strDescription;
    public string       strDisplayName;
    public string       strName;
    public string       strToolTip;
}
```

Members

strName

Node name

strDisplayName

Display name

strDescription

Node description, it is not supported, reserved

strToolTip

Prompt

enVisivility

Visible or not, see value definitions in Enumeration Type: ***MV_XML_Visibility***

enAccessMode

Accessing mode, see value definitions in Enumeration Type: ***MV_XML_AccessMode***

blsLocked

Lock or not (not supported, reserved): 0- unlock, 1- lock

nValue

Current value

nMinValue

Minimum value

nMaxValue

Maximum value

nIncrement

Increment

5.1.52 MV_XML_NODE_FEATURE

Single node basic attribute

Structure Definition

```
public struct MV_XML_NODE_FEATURE{
    public MyCamera.MV_XML_InterfaceType  enType;
    public MyCamera.MV_XML_Visibility    enVisivility;
    public uint                          nReserved;
    public string                        strDescription;
    public string                        strDisplayName;
    public string                        strName;
    public string                        strToolTip;
}
```

Members

enType

Node type, see value definitions in Enumeration Type: ***MV_XML_InterfaceType***

enVisivility

Visible or not see value definitions in Enumeration Type: ***MV_XML_Visibility***

nReserved

Reserved

strDescription

Node description, it is not supported, reserved

strDisplayName

Display name

strName

Node name

strToolTip

Prompt

5.1.53 MV_XML_NODES_LIST

Node list

Structure Definition

```
public struct MV_XML_NODES_LIST{
    public uint          nNodeNum;
    public MyCamera.MV_XML_NODE_FEATURE  stNodes;
}
```

Members

nNodeNum

The number of nodes

stNodes

Single node information, refer to ***MV_XML_NODE_FEATURE*** for details.

5.1.54 MVCC_ENUMVALUE

Parameters of enumeration type

Structure Definition

```
public struct MVCC_ENUMVALUE{
    public uint  nCurValue;
    public uint  nReserved;
    public uint  nSupportedNum;
    public uint  nSupportValue;
}
```


Members

nCurValue

Current value

nReserved

The number of supported valid data

nSupportedNum

Supported enumeration type. Each array indicates one type, and up to nSupportValue types are supported

nSupportValue

Reserved, and set as 0

5.1.55 MVCC_FLOATVALUE

Parameters of float type

Structure Definition

```
public struct MVCC_FLOATVALUE{  
    public float    fCurValue;  
    public float    fMax;  
    public float    fMin;  
    public uint     nReserved;  
}
```

Members

fCurValue

Current value

fMax

Maximum value

fMin

Minimum value

nReserved

Reserved

5.1.56 MVCC_INTVALUE

Parameters of integer type

Structure Definition

```
public struct MVCC_FLOATVALUE{  
    public unsigned int    nCurValue;  
    public unsigned int    nMax;  
    public unsigned int    nMin;  
    public unsigned int    nInc;  
    public unsigned int    nReserved;  
}
```

Members

nCurValue

Current value

nMax

Maximum value

nMin

Minimum value

nInc

Increment

nReserved

Reserved

5.1.57 MVCC_FLOATVALUE_EX

Integer type parameter value

Structure Definition

```
public struct MVCC_FLOATVALUE_EX{  
    public Int64    nCurValue;  
    public Int64    nMax;  
    public Int64    nMin;  
    public Int64    nInc;  
    public uint[]   nReserved;  
}
```

Members

nCurValue

Current value

nMax

Maximum value

nMin

Minimum value

nInc

Increment

nReserved

Reserved, set as 0

5.1.58 MVCC_STRINGVALUE

Parameters of string type

Structure Definition

```
public struct MVCC_STRINGVALU{  
    public string  chCurValue;  
    public uint   nReserved;  
}
```

Members

chCurValue

Current value

nReserved

Reserved, and set as 0

5.1.59 SPECIAL_INFO

GigE and Usb3 device information structure

Structure Definition

```
public struct SPECIAL_INFO{  
    public byte   stGigEInfo;  
    public byte   stUsb3VInfo;  
    public byte   stCamLInfo;  
}
```

Members

stGigEInfo

GIGE device information, it is valid when nLayerType in structure of **MV_CC_DEVICE_INFO** is MV_GIGE_DEVICE.

stUsb3VInfo

USB device information, it is valid when nLayerType in structure of **MV_CC_DEVICE_INFO** is MV_USB_DEVICE.

stCamLInfo

CameraLink device information, it is valid when nLayerType in structure of **MV_CC_DEVICE_INFO** is MV_CAMERALINK_DEVICE.

5.2 Enumeration

5.2.1 EPixelFormat

Pixel type enumeration

Enumeration Definition

```
public enum EPixelFormat{
    PixelType_Gvsp_Undefined           = -1,
    PixelType_Gvsp_Mono1p              = 16842807,
    PixelType_Gvsp_Mono2p              = 16908344,
    PixelType_Gvsp_Mono4p              = 17039417,
    PixelType_Gvsp_Mono8               = 17301505,
    PixelType_Gvsp_Mono8_Signed        = 17301506,
    PixelType_Gvsp_BayerGR8            = 17301512,
    PixelType_Gvsp_BayerRG8            = 17301513,
    PixelType_Gvsp_BayerGB8            = 17301514,
    PixelType_Gvsp_BayerBG8            = 17301515,
    PixelType_Gvsp_Mono10_Packed       = 17563652,
    PixelType_Gvsp_Mono12_Packed       = 17563654,
    PixelType_Gvsp_BayerGR10_Packed    = 17563686,
    PixelType_Gvsp_BayerRG10_Packed    = 17563687,
    PixelType_Gvsp_BayerGB10_Packed    = 17563688,
    PixelType_Gvsp_BayerBG10_Packed    = 17563689,
    PixelType_Gvsp_BayerGR12_Packed    = 17563690,
    PixelType_Gvsp_BayerRG12_Packed    = 17563691,
    PixelType_Gvsp_BayerGB12_Packed    = 17563692,
    PixelType_Gvsp_BayerBG12_Packed    = 17563693,
    PixelType_Gvsp_Mono10              = 17825795,
    PixelType_Gvsp_Mono12              = 17825797,
    PixelType_Gvsp_Mono16              = 17825799,
    PixelType_Gvsp_BayerGR10           = 17825804,
    PixelType_Gvsp_BayerRG10           = 17825805,
    PixelType_Gvsp_BayerGB10           = 17825806,
    PixelType_Gvsp_BayerBG10           = 17825807,
    PixelType_Gvsp_BayerGR12           = 17825808,
    PixelType_Gvsp_BayerRG12           = 17825809,
    PixelType_Gvsp_BayerGB12           = 17825810,
    PixelType_Gvsp_BayerBG12           = 17825811,
    PixelType_Gvsp_Mono14              = 17825829,
```

```
PixelType_Gvsp_BayerGR16      = 17825838,
PixelType_Gvsp_BayerRG16      = 17825839,
PixelType_Gvsp_BayerGB16      = 17825840,
PixelType_Gvsp_BayerBG16      = 17825841,
PixelType_Gvsp_YUV411_Packed   = 34340894,
PixelType_Gvsp_YCBCR411_8_CBYCRY = 34340924,
PixelType_Gvsp_YCBCR601_411_8_CBYCRY = 34340927,
PixelType_Gvsp_YCBCR709_411_8_CBYCRY = 34340930,
PixelType_Gvsp_YUV422_Packed   = 34603039,
PixelType_Gvsp_YUV422_YUYV_Packed = 34603058,
PixelType_Gvsp_RGB565_Packed    = 34603061,
PixelType_Gvsp_BGR565_Packed    = 34603062,
PixelType_Gvsp_YCBCR422_8      = 34603067,
PixelType_Gvsp_YCBCR601_422_8   = 34603070,
PixelType_Gvsp_YCBCR709_422_8   = 34603073,
PixelType_Gvsp_YCBCR422_8_CBYCRY = 34603075,
PixelType_Gvsp_YCBCR601_422_8_CBYCRY = 34603076,
PixelType_Gvsp_YCBCR709_422_8_CBYCRY = 34603077,
PixelType_Gvsp_RGB8_Packed      = 35127316,
PixelType_Gvsp_BGR8_Packed      = 35127317,
PixelType_Gvsp_YUV444_Packed    = 35127328,
PixelType_Gvsp_RGB8_Planar      = 35127329,
PixelType_Gvsp_YCBCR8_CBYCRY    = 35127354,
PixelType_Gvsp_YCBCR601_8_CBYCRY = 35127357,
PixelType_Gvsp_YCBCR709_8_CBYCRY = 35127360,
PixelType_Gvsp_RGBA8_Packed     = 35651606,
PixelType_Gvsp_BGRA8_Packed     = 35651607,
PixelType_Gvsp_RGB10V1_Packed   = 35651612,
PixelType_Gvsp_RGB10V2_Packed   = 35651613,
PixelType_Gvsp_RGB12V1_Packed   = 35913780,
PixelType_Gvsp_RGB10_Packed     = 36700184,
PixelType_Gvsp_BGR10_Packed     = 36700185,
PixelType_Gvsp_RGB12_Packed     = 36700186,
PixelType_Gvsp_BGR12_Packed     = 36700187,
PixelType_Gvsp_RGB10_Planar     = 36700194,
PixelType_Gvsp_RGB12_Planar     = 36700195,
PixelType_Gvsp_RGB16_Planar     = 36700196,
PixelType_Gvsp_RGB16_Packed     = 36700211,
}
```

5.2.2 MV_CAM_ACQUISITION_MODE

Acquisition mode enumeration

Enumeration Definition

```
public enum{
    MV_ACQ_MODE_SINGLE    = 0,
    MV_ACQ_MODE_MUTLI     = 1,
```

```
MV_ACQ_MODE_CONTINUOUS = 2
}MV_CAM_ACQUISITION_MODE
```

Members

MV_ACQ_MODE_SINGLE

Single frame mode

MV_ACQ_MODE_MUTLI

Multi-frame mode

MV_ACQ_MODE_CONTINUOUS

Continuous acquisition mode

5.2.3 MV_CAM_BALANCEWHITE_AUTO

Auto white balance mode

Enumeration Definition

```
public enum{
    MV_BALANCEWHITE_AUTO_OFF      = 0,
    MV_BALANCEWHITE_AUTO_CONTINUOUS = 1,
    MV_BALANCEWHITE_AUTO_ONCE     = 2
}MV_CAM_BALANCEWHITE_AUTO
```

Members

MV_BALANCEWHITE_AUTO_OFF

Disable

MV_BALANCEWHITE_AUTO_CONTINUOUS

Continuous

MV_BALANCEWHITE_AUTO_ONCE

Once

5.2.4 MV_CAM_EXPOSURE_AUTO_MODE

Enumeration of auto exposure mode enumeration

Enumeration Definition

```
public enum{
    MV_EXPOSURE_AUTO_MODE_OFF      = 0,
    MV_EXPOSURE_AUTO_MODE_ONCE     = 1,
    MV_EXPOSURE_AUTO_MODE_CONTINUOUS = 2
}MV_CAM_EXPOSURE_AUTO_MODE
```

Members

MV_EXPOSURE_AUTO_MODE_OFF

Disable

MV_EXPOSURE_AUTO_MODE_ONCE

Once

MV_EXPOSURE_AUTO_MODE_CONTINUOUS

Continuous

5.2.5 MV_CAM_GAIN_MODE

Gain mode enumeration

Enumeration Definition

```
public enum{
    MV_GAIN_MODE_OFF      = 0,
    MV_GAIN_MODE_ONCE     = 1,
    MV_GAIN_MODE_CONTINUOUS = 2
}MV_CAM_GAIN_MODE
```

Members

MV_GAIN_MODE_OFF

Disable

MV_GAIN_MODE_ONCE

Once

MV_GAIN_MODE_CONTINUOUS

Continuous

5.2.6 MV_CAM_GAMMA_SELECTOR

Gamma type enumeration

Enumeration Definition

```
public enum{
    MV_GAMMA_SELECTOR_USER   = 1,
    MV_GAMMA_SELECTOR_SRGB  = 2
}MV_CAM_GAMMA_SELECTOR
```

Members

MV_GAMMA_SELECTOR_USER

Custom

MV_GAMMA_SELECTOR_SRGB

sRGB type

5.2.7 MV_CAM_TRIGGER_MODE

Trigger mode enumeration

Enumeration Definition

```
public enum{
    MV_TRIGGER_MODE_OFF    = 0,
    MV_TRIGGER_MODE_ON     = 1
}MV_CAM_TRIGGER_MODE
```

Members

MV_TRIGGER_MODE_OFF

Disable

MV_TRIGGER_MODE_ON

Enable

5.2.8 MV_CAM_TRIGGER_SOURCE

Trigger source enumeration

Enumeration Definition

```
public enum{
    MV_TRIGGER_SOURCE_LINE0    = 0,
    MV_TRIGGER_SOURCE_LINE1    = 1,
    MV_TRIGGER_SOURCE_LINE2    = 2,
    MV_TRIGGER_SOURCE_LINE3    = 3,
    MV_TRIGGER_SOURCE_COUNTER0  = 4,
    MV_TRIGGER_SOURCE_SOFTWARE = 7
}MV_CAM_TRIGGER_SOURCE
```

Members

MV_TRIGGER_SOURCE_LINE0

LINE0 hardware trigger

MV_TRIGGER_SOURCE_LINE1

LINE1 hardware trigger

MV_TRIGGER_SOURCE_LINE2

LINE2 hardware trigger

MV_TRIGGER_SOURCE_LINE3

LINE3 hardware trigger

MV_TRIGGER_SOURCE_COUNTER0

COUNTER0 hardware trigger

MV_TRIGGER_SOURCE_SOFTWARE

Software trigger

5.2.9 MV_CC_BAYER_NOISE_FEATURE_TYPE**Enumeration about Noise Characteristics**

Member	Marco Definition Value	Description
MV_CC_BAYER_NOISE_FEATURE_TYPE_INVALID	0	Invalid
MV_CC_BAYER_NOISE_FEATURE_TYPE_PROFILE	1	Noise curve
MV_CC_BAYER_NOISE_FEATURE_TYPE_LEVEL	2	Noise level
MV_CC_BAYER_NOISE_FEATURE_TYPE_DEFAULT	3	Default value

5.2.10 MV_GIGE_EVENT

Event enumeration type

Enumeration Definition

```
public enum{
    MV_EVENT_ExposureEnd          = 1,
    MV_EVENT_FrameStartOvertrigger = 2,
    MV_EVENT_AcquisitionStartOvertrigger = 3,
    MV_EVENT_FrameStart           = 4,
    MV_EVENT_AcquisitionStart      = 5,
    MV_EVENT_EventOverrun          = 6
}MV_GIGE_EVENT
```

Members

MV_EVENT_ExposureEnd

The end of each frame exposure, not support

MV_EVENT_FrameStartOvertrigger

Frame starts over-trigger (the next frame is triggered before the end of the previous frame trigger), not support

MV_EVENT_AcquisitionStartOvertrigger

Streaming start over-trigger (the streaming signal is sent too often), not support

MV_EVENT_FrameStart

Start each frame, not support

MV_EVENT_AcquisitionStart

Start streaming (continuous or single frame mode), not support

MV_EVENT_EventOverrun

Event over-trigger (the event is sent too often), not support

5.2.11 MV_GIGE_TRANSMISSION_TYPE

Transmission mode, including single cast mode, multicast mode, and so on.

Enumeration Definition

```
public enum{
    MV_GIGE_TRANSTYPE_UNICAST          = 0x0,
    MV_GIGE_TRANSTYPE_MULTICAST        = 0x1,
    MV_GIGE_TRANSTYPE_LIMITEDBROADCAST = 0x2,
    MV_GIGE_TRANSTYPE_SUBNETBROADCAST  = 0x3,
    MV_GIGE_TRANSTYPE_CAMERADEFINED    = 0x4,
    MV_GIGE_TRANSTYPE_UNICAST_DEFINED_PORT = 0x5,
    MV_GIGE_TRANSTYPE_UNICAST_WITHOUT_RECV = 0x00010000,
    MV_GIGE_TRANSTYPE_MULTICAST_WITHOUT_RECV = 0x00010001,
}MV_GIGE_TRANSMISSION_TYPE;
```

Members

MV_GIGE_TRANSTYPE_UNICAST

Unicast

MV_GIGE_TRANSTYPE_MULTICAST

Multicast

MV_GIGE_TRANSTYPE_LIMITEDBROADCAST

LAN broadcast

MV_GIGE_TRANSTYPE_SUBNETBROADCAST

Subnet broadcast

MV_GIGE_TRANSTYPE_CAMERADEFINED

Get from camera

MV_GIGE_TRANSTYPE_UNICAST_DEFINED_PORT

Port No. of getting image data

MV_GIGE_TRANSTYPE_UNICAST_WITHOUT_RECV


Unicast mode, but not receive image data

MV_GIGE_TRANSTYPE_MULTICAST_WITHOUT_RECV

Multiple mode, but not receive image data

5.2.12 MV_CC_GAMMA_TYPE

Enumeration about Gamma Type

Enumeration Type	Macro Definition Value	Description
MV_CC_GAMMA_TYPE_NONE	0	Disable.
MV_CC_GAMMA_TYPE_VALUE	1	Gamma value
MV_CC_GAMMA_TYPE_USER_CURVE	2	Gamma curve: 8bit: required length: 256*sizeof(unsigned char) 10bit: required length: 1024*sizeof(unsigned short) 12bit: required length: 4096*sizeof(unsigned short) 16bit: required length: 65536*sizeof(unsigned short)
MV_CC_GAMMA_TYPE_LRGB2SRGB	3	Linear RGB to sRGB.
MV_CC_GAMMA_TYPE_SRGB2LRGB	4	sRGB to linear RGB.  Note This parameter is valid for color interpolation only, it is invalid for color correction.

5.2.13 MV_GRAB_STRATEGY

Strategy enumeration definition

Enumeration Definition

```
public enum{
    MV_GrabStrategy_OneByOne      = 0,
    MV_GrabStrategy_LatestImagesOnly = 1,
    MV_GrabStrategy_LatestImages   = 2,
    MV_GrabStrategy_UpcomingImage  = 3,
}MV_GRAB_STRATEGY;
```

Members

MV_GrabStrategy_OneByOne

Get image frames one by one in the chronological order, it is the default strategy.

MV_GrabStrategy_LatestImagesOnly

Only get the latest one frame in the list, and clear the rest images in the list.

MV_GrabStrategy_LatestImages

Get the latest image in the list, and the quantity of frames depends on the parameter **OutputQueueSize**, value range: [1,ImageNodeNum]. If the **OutputQueueSize** values 1, the strategy is same to "LatestImagesOnly", and if the **OutputQueueSize** values "ImageNodeNum", the strategy is same to "OneByOne".

MV_GrabStrategy_UpcomingImage

Wait for the upcoming frame.

5.2.14 MV_IMG_FLIP_TYPE

Enumeration about Flip Types

Member	Marco Definition Value	Description
MV_FLIP_VERTICAL	1	Vertical
MV_FLIP_HORIZONTAL	2	Horizontal

5.2.15 MV_IMG_ROTATION_ANGLE

Enumeration about Rotation Angle

Member	Marco Definition Value	Description
MV_IMAGE_ROTATE_90	1	90°
MV_IMAGE_ROTATE_180	2	180°
MV_IMAGE_ROTATE_270	3	270°

5.2.16 MV_SAVE_IAMGE_TYPE

Picture format type enumeration

Enumeration Definition

```
public enum MV_SAVE_IAMGE_TYPE{
    MV_Image_Undefined    = 0,
    MV_Image_Bmp          = 1,
    MV_Image_Jpeg         = 2,
    MV_Image_Png          = 3,
    MV_Image_Tif          = 4,
}
```

Members

MV_Image_Undefined

Undefined

MV_Image_Bmp

BMP picture

MV_Image_Jpeg

JPEG picture

MV_Image_Png

PNG picture

MV_Image_Tif

TIF picture

5.2.17 MV_SAVE_POINT_CLOUD_FILE_TYPE

The saved 3D data formats

Enumeration Definition

```
public enum MV_SAVE_POINT_CLOUD_FILE_TYPE{
    MV_PointCloudFile_Undefined    = 0,
    MV_PointCloudFile_PLY          = 1,
    MV_PointCloudFile_CSV          = 2,
    MV_PointCloudFile_OBJ          = 3,
};
```

Members

MV_PointCloudFile_Undefined

Undefined point cloud format

MV_PointCloudFile_PLY

The point cloud format named PLY

MV_PointCloudFile_CSV

The point cloud format named CSV

MV_PointCloudFile_OBJ

The point cloud format named OBJ

5.2.18 MV_XML_AccessMode

Accessing mode enumeration type

Enumeration Definition

```
public enum MV_XML_AccessMode{
    AM_NI = 0,
    AM_NA = 1,
    AM_WO = 2,
    AM_RO = 3,
    AM_RW = 4,
    AM_Undefined = 5,
    AM_CycleDetect=6,
}
```

Members

AM_NI

Not implemented

AM_NA

Not available

AM_WO

Write only

AM_RO

Read only

AM_RW

Read and write

AM_Undefined

Object is not yet initialized

AM_CycleDetect

Used internally for AccessMode cycle detection

5.2.19 MV_XML_InterfaceType

Node interface type

Enumeration Definition

```
public enum MV_XML_InterfaceType{
    IFT_IValue = 0,
    IFT_IBase = 1,
    IFT_IInteger = 2,
    IFT_IBoolean = 3,
    IFT_ICommand = 4,
    IFT_IFloat = 5,
    IFT_IString = 6,
    IFT_IRegister = 7,
    IFT_ICategory = 8,
    IFT_IEnumeration = 9,
    IFT_IEnumEntry = 10,
    IFT_IPort = 11,
}
```

Members

IFT_IValue

IValue interface

IFT_IBase

IBase interface

IFT_IInteger

IInteger interface

IFT_IBoolean

IBoolean interface

IFT_ICommand

ICommand interface

IFT_IFloat

IFloat interface

IFT_IString

IString interface

IFT_IRegister

IRegister interface

IFT_ICategory

Integer interface

IFT_IEnumeration

IEnumeration interface

IFT_IEnumEntry

IEnumEntry interface

IFT_IPort

IPort interface

5.2.20 MV_XML_Visibility

Enumerate visible modes.

Enumeration Definition

```
public enum{
  V_Beginner    = 0,
  V_Expert      = 1,
  V_Guru        = 2,
  V_Invisible   = 3,
  V_Undefined   = 99
}MV_XML_Visibility
```

Members

V_Beginner

Always visible

V_Expert

Visible for experts or Gurus

V_Guru

Visible for Gurus

V_Invisible

Not Visible

V_Undefined

Object is not yet initialized

5.2.21 MvGvspPixelFormat

Enumerate GigE protocol pixel format

Enumeration Definition

```
public enum MvGvspPixelFormat{
    //Custom pixel type
    PixelType_Gvsp_Undefined = -1,

    // Mono buffer format defines
    PixelType_Gvsp_Mono1p      = 0x01010037,
    PixelType_Gvsp_Mono2p      = 0x01020038,
    PixelType_Gvsp_Mono4p      = 0x01040039,
    PixelType_Gvsp_Mono8       = 0x01080001,
    PixelType_Gvsp_Mono8_Signed = 0x01080002,
    PixelType_Gvsp_Mono10      = 0x01100003,
    PixelType_Gvsp_Mono10_Packed = 0x010c0004,
    PixelType_Gvsp_Mono12      = 0x01100005,
    PixelType_Gvsp_Mono12_Packed = 0x010c0006,
    PixelType_Gvsp_Mono14      = 0x01100025,
    PixelType_Gvsp_Mono16      = 0x01100007,

    // Bayer buffer format defines
    PixelType_Gvsp_BayerGR8     = 0x01080008,
    PixelType_Gvsp_BayerRG8     = 0x01080009,
    PixelType_Gvsp_BayerGB8     = 0x0108000a,
    PixelType_Gvsp_BayerBG8     = 0x0108000b,
    PixelType_Gvsp_BayerGR10    = 0x0110000c,
    PixelType_Gvsp_BayerRG10    = 0x0110000d,
    PixelType_Gvsp_BayerGB10    = 0x0110000e,
    PixelType_Gvsp_BayerBG10    = 0x0110000f,
    PixelType_Gvsp_BayerGR12    = 0x01100010,
    PixelType_Gvsp_BayerRG12    = 0x01100011,
    PixelType_Gvsp_BayerGB12    = 0x01100012,
    PixelType_Gvsp_BayerBG12    = 0x01100013,
    PixelType_Gvsp_BayerGR10_Packed = 0x010c0026,
    PixelType_Gvsp_BayerRG10_Packed = 0x010c0027,
    PixelType_Gvsp_BayerGB10_Packed = 0x010c0028,
    PixelType_Gvsp_BayerBG10_Packed = 0x010c0029,
    PixelType_Gvsp_BayerGR12_Packed = 0x010c002a,
    PixelType_Gvsp_BayerRG12_Packed = 0x010c002b,
    PixelType_Gvsp_BayerGB12_Packed = 0x010c002c,
    PixelType_Gvsp_BayerBG12_Packed = 0x010c002d,
    PixelType_Gvsp_BayerGR16     = 0x0110002e,
    PixelType_Gvsp_BayerRG16     = 0x0110002f,
    PixelType_Gvsp_BayerGB16     = 0x01100030,
```

```
PixelType_Gvsp_BayerBG16          = 0x01100031,

// RGB Packed buffer format defines
PixelType_Gvsp_RGB8_Packed        = 0x02180014,
PixelType_Gvsp_BGR8_Packed        = 0x02180015,
PixelType_Gvsp_RGBA8_Packed       = 0x02200016,
PixelType_Gvsp_BGRA8_Packed       = 0x02200017,
PixelType_Gvsp_RGB10_Packed       = 0x02300018,
PixelType_Gvsp_BGR10_Packed       = 0x02300019,
PixelType_Gvsp_RGB12_Packed       = 0x0230001a,
PixelType_Gvsp_BGR12_Packed       = 0x0230001b,
PixelType_Gvsp_RGB16_Packed       = 0x02300033,
PixelType_Gvsp_RGB10V1_Packed     = 0x0220001c,
PixelType_Gvsp_RGB10V2_Packed     = 0x0220001d,
PixelType_Gvsp_RGB12V1_Packed     = 0x02240034,
PixelType_Gvsp_RGB565_Packed      = 0x02100035,
PixelType_Gvsp_BGR565_Packed      = 0x02100036,

// YUV Packed buffer format defines
PixelType_Gvsp_YUV411_Packed      = 0x020c001e,
PixelType_Gvsp_YUV422_Packed      = 0x0210001f,
PixelType_Gvsp_YUV422_YUYV_Packed = 0x02100032,
PixelType_Gvsp_YUV444_Packed      = 0x02180020,
PixelType_Gvsp_YCBCR8_CBYCR       = 0x0218003a,
PixelType_Gvsp_YCBCR422_8         = 0x0210003b,
PixelType_Gvsp_YCBCR422_8_CBYCRY  = 0x02100043,
PixelType_Gvsp_YCBCR411_8_CBYCRY  = 0x020c003c,
PixelType_Gvsp_YCBCR601_8_CBYCR   = 0x0218003d,
PixelType_Gvsp_YCBCR601_422_8     = 0x0210003e,
PixelType_Gvsp_YCBCR601_422_8_CBYCRY = 0x02100044,
PixelType_Gvsp_YCBCR601_411_8_CBYCRY = 0x020c003f,
PixelType_Gvsp_YCBCR709_8_CBYCR   = 0x02180040,
PixelType_Gvsp_YCBCR709_422_8     = 0x02100041,
PixelType_Gvsp_YCBCR709_422_8_CBYCRY = 0x02100045,
PixelType_Gvsp_YCBCR709_411_8_CBYCRY = 0x020c0042,

// RGB Planar buffer format defines
PixelType_Gvsp_RGB8_Planar        = 0x02180021,
PixelType_Gvsp_RGB10_Planar       = 0x02300022,
PixelType_Gvsp_RGB12_Planar       = 0x02300023,
PixelType_Gvsp_RGB16_Planar       = 0x02300024,

// Custom picture format
PixelType_Gvsp_Jpeg                = unchecked((Int32)0x80180001),
PixelType_Gvsp_Coord3D_ABC32f      = 0x026000C0,
PixelType_Gvsp_Coord3D_ABC32f_Planar = 0x026000C1,
PixelType_Gvsp_Coord3D_AC32f       = 0x024000C2, //3D coordinate A-C 32-bit floating point
PixelType_Gvsp_COORD3D_DEPTH_PLUS_MASK = unchecked((Int32)0x821c0001),
PixelType_Gvsp_Coord3D_ABC32       = unchecked((Int32)0x82603001),
PixelType_Gvsp_Coord3D_AB32f       = unchecked((Int32)0x82403002),
PixelType_Gvsp_Coord3D_AB32        = unchecked((Int32)0x82403003),
PixelType_Gvsp_Coord3D_AC32f_Planar = 0x024000C3,
```

```

PixelType_Gvsp_Coord3D_AC32      = unchecked((Int32)0x82403004),
PixelType_Gvsp_Coord3D_A32f     = 0x012000BD,
PixelType_Gvsp_Coord3D_A32      = unchecked((Int32)0x81203005),
PixelType_Gvsp_Coord3D_C32f     = 0x012000BF,
PixelType_Gvsp_Coord3D_C32      = unchecked((Int32)0x81203006),
PixelType_Gvsp_Coord3D_ABC16     = 0x023000b9,
PixelType_Gvsp_Coord3D_C16      = 0x011000b8,

//Lossless decoding pixel format
PixelType_Gvsp_HB_Mono8         = unchecked((Int32)0x81080001),
PixelType_Gvsp_HB_Mono10        = unchecked((Int32)0x81100003),
PixelType_Gvsp_HB_Mono10_Packed = unchecked((Int32)0x810c0004),
PixelType_Gvsp_HB_Mono12        = unchecked((Int32)0x81100005),
PixelType_Gvsp_HB_Mono12_Packed = unchecked((Int32)0x810c0006),
PixelType_Gvsp_HB_Mono16        = unchecked((Int32)0x81100007),
PixelType_Gvsp_HB_BayerGR8      = unchecked((Int32)0x81080008),
PixelType_Gvsp_HB_BayerRG8      = unchecked((Int32)0x81080009),
PixelType_Gvsp_HB_BayerGB8      = unchecked((Int32)0x8108000a),
PixelType_Gvsp_HB_BayerBG8      = unchecked((Int32)0x8108000b),
PixelType_Gvsp_HB_BayerGR10     = unchecked((Int32)0x8110000c),
PixelType_Gvsp_HB_BayerRG10     = unchecked((Int32)0x8110000d),
PixelType_Gvsp_HB_BayerGB10     = unchecked((Int32)0x8110000e),
PixelType_Gvsp_HB_BayerBG10     = unchecked((Int32)0x8110000f),
PixelType_Gvsp_HB_BayerGR12     = unchecked((Int32)0x81100010),
PixelType_Gvsp_HB_BayerRG12     = unchecked((Int32)0x81100011),
PixelType_Gvsp_HB_BayerGB12     = unchecked((Int32)0x81100012),
PixelType_Gvsp_HB_BayerBG12     = unchecked((Int32)0x81100013),
PixelType_Gvsp_HB_BayerGR10_Packed = unchecked((Int32)0x810c0026),
PixelType_Gvsp_HB_BayerRG10_Packed = unchecked((Int32)0x810c0027),
PixelType_Gvsp_HB_BayerGB10_Packed = unchecked((Int32)0x810c0028),
PixelType_Gvsp_HB_BayerBG10_Packed = unchecked((Int32)0x810c0029),
PixelType_Gvsp_HB_BayerGR12_Packed = unchecked((Int32)0x810c002a),
PixelType_Gvsp_HB_BayerRG12_Packed = unchecked((Int32)0x810c002b),
PixelType_Gvsp_HB_BayerGB12_Packed = unchecked((Int32)0x810c002c),
PixelType_Gvsp_HB_BayerBG12_Packed = unchecked((Int32)0x810c002d),
PixelType_Gvsp_HB_YUV422_Packed  = unchecked((Int32)0x8210001f),
PixelType_Gvsp_HB_YUV422_YUYV_Packed = unchecked((Int32)0x82100032),
PixelType_Gvsp_HB_RGB8_Packed   = unchecked((Int32)0x82180014),
PixelType_Gvsp_HB_BGR8_Packed   = unchecked((Int32)0x82180015),
PixelType_Gvsp_HB_RGBA8_Packed  = unchecked((Int32)0x82200016),
PixelType_Gvsp_HB_BGRA8_Packed  = unchecked((Int32)0x82200017),
}

```

Remarks

The macro definitions of enumeration types are listed below:

Macro Definition	Value
MV_GVSP_PIX_MONO	0x01000000
MV_GVSP_PIX_COLOR	0x02000000
MV_PIXEL_BIT_COUNT(n)	(n) << 16

Chapter 6 FAQ (Frequently Asked 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.

6.1 GigE Vision Camera

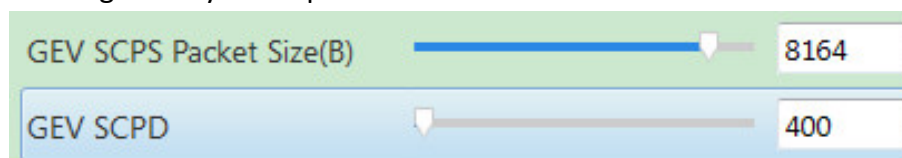
6.1.1 Why is there packet loss?

Cause

The abnormal network transmission environment causes the packet loss of data transmission.

Solution

1. Check if the bandwidth is sufficient.
2. Enable the NIC jumbo frame.
3. Disable firewall.
4. Increase the SCPD gradually till no packet loss.



6.1.2 Why does link error occur in the normal compiled Demo?

Cause

No administrator permission for Demo directory will make it unable to write the .exe file.

Solution

Change the Demo directory to the directory with administrator permission.

6.1.3 Why can't I set the static IP under DHCP?

Cause

The camera with unpublished version limit the gateway, the 0.0.0.0 will display failed.

Solution

Upgrade firmware again.

6.1.4 Why do I failed to perform the software trigger command when calling SDK?

Cause

The trigger source is not set to software trigger.

Solution

Before performing software trigger command, make sure the camera is in software trigger mode and the trigger source is set to software trigger.

6.1.5 Why does the camera often be offline?

Cause 1

The NIC card is in sleep status.

Solution 1

Set the power option of operating system to avoid the computer going to the sleep status.

Cause 2

The network port may be not plugged in.

Solution 2

Check the network port status.

6.1.6 Why is no permission returned when calling API MV_CC_OpenDevice_NET?

Cause 1

The camera is occupied.

Solution 1

Check if the camera is occupied or connected by other application.

Cause 2

The configured heartbeat timeout is too long, and the program exits abnormally without executing the API of shutting down device or destroying device handle. So the device remains occupied.

Solution 2

Wait till the heartbeat timed out or unplug the camera.

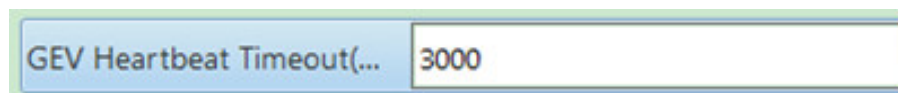
6.1.7 Why is there error code returned during debug process?

Cause

Debug will cause heartbeat sending timeout.

Solution

Lengthen the heartbeat time (example: 30s, and set the value to 3000). The default heartbeat time is 3s, see the picture below:



6.1.8 Why is no data error returned when calling API MV_CC_GetOneFrameTimeout_NET?

Cause

This API adopts active search method, and no data can be obtained when calling for only once.

Solution

Increase the timeout.

6.1.9 Why is there always no data when calling MV_CC_GetOneFrameTimeout_NET?

Cause

Image registration callback function has been called at the same time. These two functions cannot be called at the same time.

Solution

Stop calling the registration callback function.

6.1.10 Why can't open the camera after finishing debugging abnormally?

Cause

To avoid the heartbeat timeout under debug process, the default value of camera heartbeat timeout is 60000ms (60s). So sometimes the camera cannot be opened after finishing debugging abnormally.

Solution

Shut down camera before exiting debugging.

6.2 USB3 Vision Camera

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

Cause

The USB connected with camera is in Version 2.0, and the bandwidth is not enough.

Solution

Make sure the USB connected with camera is in Version 3.0. You can check the USB version information by the following methods:

1. Check the digit of the icon in front of camera name in the device list.



2. Check whether the value of **USB Speed Mode** in the device property is **Highspeed** (USB 2.0) or **SuperSpeed** (USB 3.0).

Appendix A. Error Code

The error may occurred during the MVC SDK integration are listed here for reference. You can search for the error description according to returned error codes or name.

Error Type	Error Code	Description
General Error Codes: From 0x80000000 to 0x800000FF		
MV_E_HANDLE	0x80000000	Error or invalid handle.
MV_E_SUPPORT	0x80000001	Not supported function.
MV_E_BUFOVER	0x80000002	Buffer is full.
MV_E_CALLORDER	0x80000003	Incorrect calling order
MV_E_PARAMETER	0x80000004	Incorrect parameter.
MV_E_RESOURCE	0x80000006	Applying resource failed.
MV_E_NODATA	0x80000007	No data.
MV_E_PRECONDITION	0x80000008	Precondition error, or the running environment changed.
MV_E_VERSION	0x80000009	Version mismatches.
MV_E_NOENOUGH_BUF	0x8000000A	Insufficient memory.
MV_E_ABNORMAL_IMAGE	0x8000000B	Abnormal image. Incomplete image caused by packet loss.
MV_E_LOAD_LIBRARY	0x8000000C	Importing DLL (Dynamic Link Library) failed.
MV_E_NOOUTBUF	0x8000000D	No buffer node can be outputted.
MV_E_ENCRYPT	0x8000000E	Encryption error.
MV_E_UNKNOW	0x800000FF	Unknown error.
GenICam Series Error Codes: RFrom 0x80000100 to 0x800001FF		
MV_E_GC_GENERIC	0x80000100	Generic error.
MV_E_GC_ARGUMENT	0x80000101	Illegal parameters.
MV_E_GC_RANGE	0x80000102	The value is out of range.
MV_E_GC_PROPERTY	0x80000103	Attribute error
MV_E_GC_RUNTIME	0x80000104	Running environment error.
MV_E_GC_LOGICAL	0x80000105	Incorrect logic

Error Type	Error Code	Description
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.
Exception Error Codes: From 0x00008001 to 0x00008002		

Error Type	Error Code	Description
MV_EXCEPTION_DEV_DISCONNECT	0x00008001	Device disconnected.
MV_EXCEPTION_VERSION_CHECK	0x00008002	SDK doesn't match the driver version.

Algorithm Error Codes

Error Type	Error Code	Description
General Error Codes		
MV_ALG_OK	0x00000000	OK
MV_ALG_ERR	0x00000000	Unknown error
Capability Related Error Codes		
MV_ALG_E_ABILITY_ARG	0x10000001	Invalid parameters of capabilities
Memory Related Error Codes (From 0x10000002 to 0x10000006)		
MV_ALG_E_MEM_NULL	0x10000002	The memory address is empty.
MV_ALG_E_MEM_ALIGN	0x10000003	The memory alignment is not satisfactory.
MV_ALG_E_MEM_LACK	0x10000004	No enough memory space.
MV_ALG_E_MEM_SIZE_ALIGN	0x10000005	The memory space does not meet the requirement of alignment.
MV_ALG_E_MEM_ADDR_ALIGN	0x10000006	The memory address does not meet the requirement of alignment.
Image Related Error Codes (From 0x10000007 to 0x1000000A)		
MV_ALG_E_IMG_FORMAT	0x10000007	Incorrect image format or the image format is not supported.
MV_ALG_E_IMG_SIZE	0x10000008	Invalid image width and height.
MV_ALG_E_IMG_STEP	0x10000009	The image width/height and step parameters mismatched.
MV_ALG_E_IMG_DATA_NULL	0x1000000A	The storage address of image is empty.
Input/Output Related Error Codes (From 0x1000000B to 0x10000010)		
MV_ALG_E_CFG_TYPE	0x1000000B	Incorrect type for setting/getting parameters.
MV_ALG_E_CFG_SIZE	0x1000000C	Incorrect size for setting/getting parameters.
MV_ALG_E_PRC_TYPE	0x1000000D	Incorrect processing type.

Error Type	Error Code	Description
MV_ALG_E_PRC_SIZE	0x1000000E	Incorrect parameter size for processing.
MV_ALG_E_FUNC_TYPE	0x1000000F	Incorrect sub-process type.
MV_ALG_E_FUNC_SIZE	0x10000010	Incorrect parameter size for sub-processing.
Operation Parameters Related Error Codes (From 0x10000011 to 0x10000013)		
MV_ALG_E_PARAM_INDEX	0x10000011	Incorrect index parameter.
MV_ALG_E_PARAM_VALUE	0x10000012	Incorrect or invalid value parameter.
MV_ALG_E_PARAM_NUM	0x10000013	Incorrect param_num parameter.
API Calling Related Error Codes (From 0x10000014 to 0x10000016)		
MV_ALG_E_NULL_PTR	0x10000014	Pointer to function is empty.
MV_ALG_E_OVER_MAX_MEM	0x10000015	The maximum memory reached.
MV_ALG_E_CALL_BACK	0x10000016	Callback function error.
Algorithm Library Encryption Related Error Codes (0x10000017 and 0x10000018)		
MV_ALG_E_ENCRYPT	0x10000017	Encryption error.
MV_ALG_E_EXPIRE	0x10000018	Incorrect algorithm library service life.
Basic Errors of Inner Module (From 0x10000019 and 0x1000001B)		
MV_ALG_E_BAD_ARG	0x10000019	Incorrect value range of the parameter.
MV_ALG_E_DATA_SIZE	0x1000001A	Incorrect data size.
MV_ALG_E_STEP	0x1000001B	Incorrect data step.
CPU Instruction Set Related Error Code		
MV_ALG_E_CPUID	0x1000001C	The instruction set of optimized code does not supported by the CPU.
MV_ALG_WARNING	0x1000001D	Warning.
MV_ALG_E_TIME_OUT	0x1000001E	Algorithm library timed out.
MV_ALG_E_LIB_VERSION	0x1000001F	Algorithm version No. error.
MV_ALG_E_MODEL_VERSION	0x10000020	Model version No. error.
MV_ALG_E_GPU_MEM_ALLOC	0x10000021	GUP memory allocation error.
MV_ALG_E_FILE_NON_EXIST	0x10000022	The file does not exist.
MV_ALG_E_NONE_STRING	0x10000023	The string is empty.

Error Type	Error Code	Description
MV_ALG_E_IMAGE_CODEC	0x10000024	Image decoder error.
MV_ALG_E_FILE_OPEN	0x10000025	Opening file failed.
MV_ALG_E_FILE_READ	0x10000026	Reading file failed.
MV_ALG_E_FILE_WRITE	0x10000027	Writing to file failed.
MV_ALG_E_FILE_READ_SIZE	0x10000028	Incorrect file read size.
MV_ALG_E_FILE_TYPE	0x10000029	Incorrect file type.
MV_ALG_E_MODEL_TYPE	0x1000002A	Incorrect model type.
MV_ALG_E_MALLOC_MEM	0x1000002B	Memory allocation error.
MV_ALG_E_BIND_CORE_FAILED	0x1000002C	Binding thread to core failed.
Denoising Related Error Codes (From 0x10402001 to 0x1040200f)		
MV_ALG_E_DENOISE_NE_IMG_FORMAT	0x10402001	Incorrect image format of noise characteristics.
MV_ALG_E_DENOISE_NE_FEATURE_TYPE	0x10402002	Incorrect noise characteristics type.
MV_ALG_E_DENOISE_NE_PROFILE_NUM	0x10402003	Incorrect number of noise characteristics.
MV_ALG_E_DENOISE_NE_GAIN_NUM	0x10402004	Incorrect number of noise characteristics gain.
MV_ALG_E_DENOISE_NE_GAIN_VAL	0x10402005	Incorrect noise curve gain value.
MV_ALG_E_DENOISE_NE_BIN_NUM	0x10402006	Incorrect number of noise curves.
MV_ALG_E_DENOISE_NE_INIT_GAIN	0x10402007	Incorrect settings of noise initial gain.
MV_ALG_E_DENOISE_NE_NOT_INIT	0x10402008	The noise is uninitialized.
MV_ALG_E_DENOISE_COLOR_MODE	0x10402009	Incorrect color mode.
MV_ALG_E_DENOISE_ROI_NUM	0x1040200a	Incorrect number of ROIs.

Error Type	Error Code	Description
MV_ALG_E_DENOISE_ROI_ORI_PT	0x1040200b	Incorrect ROI origin.
MV_ALG_E_DENOISE_ROI_SIZE	0x1040200c	Incorrect ROI size.
MV_ALG_E_DENOISE_GAIN_NOT_EXIST	0x1040200d	The camera gain does not exist (The maximum number of gains reached).
MV_ALG_E_DENOISE_GAIN_BEYOND_RANGE	0x1040200e	Invalid camera gain.
MV_ALG_E_DENOISE_NP_BUF_SIZE	0x1040200f	Incorrect noise characteristics memory size.

Appendix B. Sample Code

B.1 Connect to Cameras via IP Address

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ConnectSpecCamera
{
    class ConnectSpecCamera
    {
        static void Main(string[] args)
        {
            MyCamera.MV_CC_DEVICE_INFO stDevInfo = new MyCamera.MV_CC_DEVICE_INFO();
            stDevInfo.nTLayerType = MyCamera.MV_GIGE_DEVICE;
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDev = new MyCamera.MV_GIGE_DEVICE_INFO();
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();

            do
            {
                Console.WriteLine("Please input Device Ip : ");
                string strCurrentIp = Convert.ToString(Console.ReadLine()); // The IP address of the camera to connect to
                // en:The camera IP that needs to be connected (based on actual padding)
                Console.WriteLine("Please input Net Export Ip : ");
                string strNetExport = Convert.ToString(Console.ReadLine()); // The IP address of the network card to which
the camera is connected

                var parts = strCurrentIp.Split('.');
                try
                {
                    int nlp1 = Convert.ToInt32(parts[0]);
                    int nlp2 = Convert.ToInt32(parts[1]);
                    int nlp3 = Convert.ToInt32(parts[2]);
                    int nlp4 = Convert.ToInt32(parts[3]);
                    stGigEDev.nCurrentIp = (uint)((nlp1 << 24) | (nlp2 << 16) | (nlp3 << 8) | nlp4);

                    parts = strNetExport.Split('.');
                    nlp1 = Convert.ToInt32(parts[0]);
                    nlp2 = Convert.ToInt32(parts[1]);
                    nlp3 = Convert.ToInt32(parts[2]);
                    nlp4 = Convert.ToInt32(parts[3]);
```

```
stGigEDev.nNetExport = (uint)((nlp1 << 24) | (nlp2 << 16) | (nlp3 << 8) | nlp4);
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

// stGigEDev 结构体 stDevInfo.SpecialInfo.stGigEInfo(Byte[])
IntPtr stGigEInfoPtr = Marshal.AllocHGlobal(Marshal.SizeOf(stGigEDev));
Marshal.StructureToPtr(stGigEDev, stGigEInfoPtr, false);
stDevInfo.SpecialInfo.stGigEInfo = new Byte[Marshal.SizeOf(stDevInfo.SpecialInfo)];
Marshal.Copy(stGigEInfoPtr, stDevInfo.SpecialInfo.stGigEInfo, 0, Marshal.SizeOf(stDevInfo.SpecialInfo));
Marshal.Release(stGigEInfoPtr);

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// ch:打开 | en:Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

//Set the trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
```



```
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Start acquiring image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

// Get package size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
UInt32 nPayloadSize = stParam.nCurValue;

int nCount = 0;
IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nPayloadSize); //Raw data buff
IntPtr pBufForSaveImage = IntPtr.Zero; //Image data buff
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, ref FrameInfo, 1000);
    // ch:获取一帧图像 | en:Get one image
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get One Frame: Width[{0}] , Height[{1}] , FrameNum[{2}]", FrameInfo.nWidth,
FrameInfo.nHeight, FrameInfo.nFrameNum);
        if (pBufForSaveImage == IntPtr.Zero)
        {
            pBufForSaveImage = Marshal.AllocHGlobal((int)(FrameInfo.nHeight * FrameInfo.nWidth * 3 + 2048));
        }
        MyCamera.MV_SAVE_IMAGE_PARAM_EX stSaveParam = new
MyCamera.MV_SAVE_IMAGE_PARAM_EX();
        stSaveParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Bmp;
        stSaveParam.enPixelFormat = FrameInfo.enPixelFormat;
        stSaveParam.pData = pBufForDriver;
        stSaveParam.nDataLen = FrameInfo.nFrameLen;
        stSaveParam.nHeight = FrameInfo.nHeight;
        stSaveParam.nWidth = FrameInfo.nWidth;
        stSaveParam.pImageBuffer = pBufForSaveImage;
        stSaveParam.nBufferSize = (uint)(FrameInfo.nHeight * FrameInfo.nWidth * 3 + 2048);
        stSaveParam.nJpgQuality = 80;
        nRet = device.MV_CC_SaveImageEx_NET(ref stSaveParam);
        if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Save Image failed:{0:x8}", nRet);
    continue;
}

//Save image data to local
byte[] data = new byte[stSaveParam.nImageLen];
Marshal.Copy(pBufForSaveImage, data, 0, (int)stSaveParam.nImageLen);
FileStream pFile = null;
try
{
    pFile = new FileStream("frame" + nCount.ToString() + ".bmp", FileMode.Create);
    pFile.Write(data, 0, data.Length);
}
catch
{
    Console.WriteLine("Save failed");
    continue;
}
finally
{
    pFile.Close();
}
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
    break;
}
}
}
Marshal.FreeHGlobal(pBufForDriver);
Marshal.FreeHGlobal(pBufForSaveImage);

//Stop acquiring images
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
```

```
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
            break;
        }
    } while (false);

    if (MyCamera.MV_OK != nRet)
    {
        //Destroy device
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.2 Correct Color

The sample code below shows how to correct the color of the image of a camera with gamma, CCM, and CLUT.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ColorCorrect
{
    class ColorCorrect
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            IntPtr pDstData = IntPtr.Zero;
            UInt32 nDstDataSize = 0;

            IntPtr pCLUTData = IntPtr.Zero;
            UInt32 nCLUTDataSize = 0;

            do
            {
                // Enumerate devices.
```

```
MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Enum device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Enum device count :{0} \n", stDevList.nDeviceNum);
if (0 == stDevList.nDeviceNum)
{
    break;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;

// Print camera information such as IP and user ID.
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("[device " + i.ToString() + "]:");
        Console.WriteLine("DevIP:" + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
        Console.WriteLine("UserDefineName:" + stGigEDeviceInfo.chUserDefinedName + "\n");
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("[device " + i.ToString() + "]:");
        Console.WriteLine("SerialNumber:" + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("UserDefineName:" + stUsb3DeviceInfo.chUserDefinedName + "\n");
    }
}

Int32 nDevIndex = 0;
Console.Write("Please input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
```

```
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device.
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open the device.
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Get the optimal package size (GigE camera only).
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
}
else
{
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
}
}

// Set trigger mode to Off.
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{

```

```
        Console.WriteLine("Set TriggerMode failed!");
        break;
    }

    // Start image acquisition.
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    MyCamera.MV_FRAME_OUT stFrameOut = new MyCamera.MV_FRAME_OUT();

    nRet = device.MV_CC_GetImageBuffer_NET(ref stFrameOut, 1000);
    // Get one frame.
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stFrameOut.stFrameInfo.nWidth) + "],
Height["
        + Convert.ToString(stFrameOut.stFrameInfo.nHeight) + "], FrameNum[" +
        Convert.ToString(stFrameOut.stFrameInfo.nFrameNum) + "]);

        if (pDstData == IntPtr.Zero || nDstDataSize < stFrameOut.stFrameInfo.nFrameLen)
        {
            if (pDstData != IntPtr.Zero)
            {
                Marshal.FreeHGlobal(pDstData);
                pDstData = IntPtr.Zero;
                nDstDataSize = 0;
            }

            pDstData = Marshal.AllocHGlobal((int)stFrameOut.stFrameInfo.nFrameLen);
            if (pDstData == IntPtr.Zero)
            {
                Console.WriteLine("malloc pDstData failed");
                nRet = MyCamera.MV_E_RESOURCE;
                break;
            }
            nDstDataSize = stFrameOut.stFrameInfo.nFrameLen;
        }

        MyCamera.MV_CC_COLOR_CORRECT_PARAM stColorCorrectParam = new
        MyCamera.MV_CC_COLOR_CORRECT_PARAM();

        // Color correction parameters
        stColorCorrectParam.nWidth = stFrameOut.stFrameInfo.nWidth;
        stColorCorrectParam.nHeight = stFrameOut.stFrameInfo.nHeight;
        stColorCorrectParam.pSrcBuf = stFrameOut.pBufAddr;
        stColorCorrectParam.nSrcBufLen = stFrameOut.stFrameInfo.nFrameLen;
        stColorCorrectParam.enPixelType = stFrameOut.stFrameInfo.enPixelType;
```

```
stColorCorrectParam.pDstBuf = pDstData;
stColorCorrectParam.nDstBufSize = nDstDataSize;
stColorCorrectParam.nImageBit = 8;

// Gamma parameters. Select gamma type as needed.
stColorCorrectParam.stGammaParam.enGammaType =
MyCamera.MV_CC_GAMMA_TYPE.MV_CC_GAMMA_TYPE_VALUE;
stColorCorrectParam.stGammaParam.fGammaValue = (float)0.6;

// Enter CCM parameters or/and CLUT parameters.
stColorCorrectParam.stCCMParm.bCCMEnable = false;

stColorCorrectParam.stCLUTParam.bCLUTEnable = true;
stColorCorrectParam.stCLUTParam.nCLUTScale = 1024;
stColorCorrectParam.stCLUTParam.nCLUTSize = 17;

if (false == File.Exists("./CLUTCalib.bin"))
{
    Console.WriteLine("Open file failed");
    nRet = MyCamera.MV_E_RESOURCE;
    break;
}

FileStream fs = new FileStream("./CLUTCalib.bin", FileMode.Open);
byte[] data = new byte[fs.Length];

if (pCLUTData == IntPtr.Zero || nCLUTDataSize < fs.Length)
{
    if (pCLUTData != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(pCLUTData);
        pCLUTData = IntPtr.Zero;
        nCLUTDataSize = 0;
    }

    pCLUTData = Marshal.AllocHGlobal((int)fs.Length);
    if (pCLUTData == IntPtr.Zero)
    {
        Console.WriteLine("malloc pCLUTData failed");
        break;
    }
    nCLUTDataSize = (uint)fs.Length;
}

fs.Read(data, 0, data.Length);
fs.Close();

Marshal.Copy(data, 0, pCLUTData, (Int32)nCLUTDataSize);

stColorCorrectParam.stCLUTParam.pCLUTBuf = pCLUTData;
stColorCorrectParam.stCLUTParam.nCLUTBufLen = nCLUTDataSize;
```

```
nRet = device.MV_CC_ColorCorrect_NET(ref stColorCorrectParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Color Correct Failed:{0:x8}", nRet);
    break;
}

// Save image to file.
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM stSaveFileParam = new
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM();
stSaveFileParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Bmp;
stSaveFileParam.enPixelType = stFrameOut.stFrameInfo.enPixelType;
stSaveFileParam.nWidth = stFrameOut.stFrameInfo.nWidth;
stSaveFileParam.nHeight = stFrameOut.stFrameInfo.nHeight;
stSaveFileParam.nDataLen = stFrameOut.stFrameInfo.nFrameLen;
stSaveFileParam.pData = stFrameOut.pBufAddr;
stSaveFileParam.pImagePath = "BeforeImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + stFrameOut.stFrameInfo.nFrameNum.ToString() + ".bmp";
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    break;
}

stSaveFileParam.pData = pDstData;
stSaveFileParam.pImagePath = "AfterImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + stFrameOut.stFrameInfo.nFrameNum.ToString() + ".bmp";
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    break;
}

device.MV_CC_FreelImageBuffer_NET(ref stFrameOut);
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}

// Stop image acquisition.
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close the device.
nRet = device.MV_CC_CloseDevice_NET();
```



```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Close device failed{0:x8}", nRet);
            break;
        }

        // Destroy the device.
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
            break;
        }
    } while (false);

    if (pDstData != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(pDstData);
        pDstData = IntPtr.Zero;
    }

    if (pCLUTData != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(pCLUTData);
        pCLUTData = IntPtr.Zero;
    }

    if (MyCamera.MV_OK != nRet)
    {
        // Destroy the device.
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.3 Correct Lens Shading

The sample code shows how to correct lens shading.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
```

```
using System.Runtime.InteropServices;
using System.IO;

namespace LensShadingCorrection
{
    class LensShadingCorrection
    {
        public static MyCamera.cbOutputExdelegate ImageCallback;
        public static MyCamera device = new MyCamera();
        static bool g_IsNeedCalib = true;
        static IntPtr g_pCalibBuf = IntPtr.Zero;
        static IntPtr g_pDstData = IntPtr.Zero;
        static uint g_nCalibBufSize = 0;
        static uint g_nDstDataSize = 0;

        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            int nRet = MyCamera.MV_OK;
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
                Convert.ToString(pFrameInfo.nHeight) + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
            // Judge whether the camera needs calibration.
            if (true == g_IsNeedCalib)
            {
                // LSC calibration
                MyCamera.MV_CC_LSC_CALIB_PARAM stLSCCalib = new MyCamera.MV_CC_LSC_CALIB_PARAM();
                stLSCCalib.nWidth = pFrameInfo.nWidth;
                stLSCCalib.nHeight = pFrameInfo.nHeight;
                stLSCCalib.enPixelType = pFrameInfo.enPixelType;
                stLSCCalib.pSrcBuf = pData;
                stLSCCalib.nSrcBufLen = pFrameInfo.nFrameLen;

                if (g_pCalibBuf == IntPtr.Zero || g_nCalibBufSize < pFrameInfo.nWidth * pFrameInfo.nHeight * 2)
                {
                    if (g_pCalibBuf != IntPtr.Zero)
                    {
                        Marshal.FreeHGlobal(g_pCalibBuf);
                        g_pCalibBuf = IntPtr.Zero;
                        g_nCalibBufSize = 0;
                    }

                    g_pCalibBuf = Marshal.AllocHGlobal((int)(pFrameInfo.nWidth * pFrameInfo.nHeight * 2));
                    if (g_pCalibBuf == IntPtr.Zero)
                    {
                        Console.WriteLine("malloc pCalibBuf failed");
                        return;
                    }
                    g_nCalibBufSize = (uint)(pFrameInfo.nWidth * pFrameInfo.nHeight * 2);
                }

                stLSCCalib.pCalibBuf = g_pCalibBuf;
                stLSCCalib.nCalibBufSize = g_nCalibBufSize;
            }
        }
    }
}
```

```
stLSCCalib.nSecNumW = 689;
stLSCCalib.nSecNumH = 249;
stLSCCalib.nPadCoef = 2;
stLSCCalib.nCalibMethod = 2;
stLSCCalib.nTargetGray = 100;
nRet = device.MV_CC_LSCCalib_NET(ref stLSCCalib);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("LSC Calib failed:{0:x8}", nRet);
    return;
}

// Save calibration data file.
byte[] LSCCalibData = new byte[stLSCCalib.nCalibBufLen];
Marshal.Copy(stLSCCalib.pCalibBuf, LSCCalibData, 0, (int)stLSCCalib.nCalibBufLen);
FileStream pFile = null;
try
{
    pFile = new FileStream("./LSCCalib.bin", FileMode.Create);
    pFile.Write(LSCCalibData, 0, LSCCalibData.Length);
}
catch
{
    Console.WriteLine("Saving failed");
}
finally
{
    pFile.Close();
}

g_IsNeedCalib = false;
}

// LSC correction
if (g_pDstData == IntPtr.Zero || g_nDstDataSize < pFrameInfo.nFrameLen)
{
    if (g_pDstData != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(g_pDstData);
        g_pDstData = IntPtr.Zero;
        g_nDstDataSize = 0;
    }

    g_pDstData = Marshal.AllocHGlobal((int)pFrameInfo.nFrameLen);
    if (g_pDstData == IntPtr.Zero)
    {
        Console.WriteLine("malloc pDstData failed");
        return;
    }
    g_nDstDataSize = pFrameInfo.nFrameLen;
}
```

```
MyCamera.MV_CC_LSC_CORRECT_PARAM stLSCCorrectParam = new
MyCamera.MV_CC_LSC_CORRECT_PARAM();
stLSCCorrectParam.nWidth = pFrameInfo.nWidth;
stLSCCorrectParam.nHeight = pFrameInfo.nHeight;
stLSCCorrectParam.enPixelType = pFrameInfo.enPixelType;
stLSCCorrectParam.pSrcBuf = pData;
stLSCCorrectParam.nSrcBufLen = pFrameInfo.nFrameLen;

stLSCCorrectParam.pDstBuf = g_pDstData;
stLSCCorrectParam.nDstBufSize = g_nDstDataSize;

stLSCCorrectParam.pCalibBuf = g_pCalibBuf;
stLSCCorrectParam.nCalibBufLen = g_nCalibBufSize;
nRet = device.MV_CC_LSCCorrect_NET(ref stLSCCorrectParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("LSC Correct failed:{0:x8}", nRet);
    return;
}

if (pFrameInfo.nFrameNum < 10)
{
    // Save image to file.
    MyCamera.MV_SAVE_IMG_TO_FILE_PARAM stSaveFileParam = new
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM();
stSaveFileParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Bmp;
stSaveFileParam.enPixelType = pFrameInfo.enPixelType;
stSaveFileParam.nWidth = pFrameInfo.nWidth;
stSaveFileParam.nHeight = pFrameInfo.nHeight;
stSaveFileParam.nDataLen = pFrameInfo.nFrameLen;
stSaveFileParam.pData = pData;
stSaveFileParam.plImagePath = "BeforeImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + pFrameInfo.nFrameNum.ToString() + ".bmp";
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    return;
}

stSaveFileParam.pData = g_pDstData;
stSaveFileParam.plImagePath = "AfterImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + pFrameInfo.nFrameNum.ToString() + ".bmp";
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    return;
}
}
}
```

```
static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    do
    {
        // Enumerate devices.
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device information

        // Print device information
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("[device " + i.ToString() + "]:");
                Console.WriteLine("DevIP:" + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
                Console.WriteLine("UserDefineName:" + stGigEDeviceInfo.chUserDefinedName + "\n");
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
                Console.WriteLine("[device " + i.ToString() + "]:");
                Console.WriteLine("SerialNumber:" + stUsb3DeviceInfo.chSerialNumber);
                Console.WriteLine("UserDefineName:" + stUsb3DeviceInfo.chUserDefinedName + "\n");
            }
        }
    }
}
```

```
Int32 nDevIndex = 0;
Console.WriteLine("Please input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device.
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open the device.
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Get the optimal package size (GigE camera only).
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
}
else
{
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
}
```

```
    }
}

// Set trigger mode to Off.
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Judge whether it can be imported locally.
if (true == File.Exists("./LSCCalib.bin"))
{
    FileStream fs = new FileStream("./LSCCalib.bin", FileMode.Open);
    byte[] data = new byte[fs.Length];

    if (g_pCalibBuf == IntPtr.Zero || g_nCalibBufSize < fs.Length)
    {
        if (g_pCalibBuf != IntPtr.Zero)
        {
            Marshal.FreeHGlobal(g_pCalibBuf);
            g_pCalibBuf = IntPtr.Zero;
            g_nCalibBufSize = 0;
        }

        g_pCalibBuf = Marshal.AllocHGlobal((int)fs.Length);
        if (g_pCalibBuf == IntPtr.Zero)
        {
            Console.WriteLine("malloc pCalibBuf failed");
            break;
        }
        g_nCalibBufSize = (uint)fs.Length;
    }

    fs.Read(data, 0, data.Length);
    fs.Close();

    Marshal.Copy(data, 0, g_pCalibBuf, (Int32)g_nCalibBufSize);

    g_IsNeedCalib = false;
}

// Register image callback.
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}
```

```
// ch:开启抓图 || en: start grab image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

// Start image acquisition.
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close the device.
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy the device.
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (g_pCalibBuf != IntPtr.Zero)
{
    Marshal.FreeHGlobal(g_pCalibBuf);
    g_pCalibBuf = IntPtr.Zero;
    g_nCalibBufSize = 0;
}

if (g_pDstData != IntPtr.Zero)
{
    Marshal.FreeHGlobal(g_pDstData);
    g_pDstData = IntPtr.Zero;
    g_nDstDataSize = 0;
}

if (MyCamera.MV_OK != nRet)
{

```



```
// Destroy the device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
}
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.4 Convert Pixel Format

Convert images to the desired pixel format, such as Mono, Bayer.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ConvertPixelType
{
    class ConvertPixelType
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();

            do
            {
                //Enumerate devices
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count :{0} ", stDevList.nDeviceNum);
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }
            }
        }
    }
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDevInfo;

// Print device info
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

// Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Start acquiring images
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

// Get package size
```

```
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
UInt32 nPayloadSize = stParam.nCurValue;

// Dynamically create memory space based on the camera's real-time resolution
IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nPayloadSize);
IntPtr pBufForSaveImage = IntPtr.Zero;
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();

nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, ref FrameInfo, 1000);
// Get one frame
if (MyCamera.MV_OK == nRet)
{
    Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(FrameInfo.nWidth) + "], Height[" +
Convert.ToString(FrameInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(FrameInfo.nFrameNum) + "]);
    if (pBufForSaveImage == IntPtr.Zero)
    {
        pBufForSaveImage = Marshal.AllocHGlobal((int)(FrameInfo.nWidth * FrameInfo.nHeight * 3 + 2048));
    }
    MyCamera.MV_PIXEL_CONVERT_PARAM stConverPixelParam = new
MyCamera.MV_PIXEL_CONVERT_PARAM();
    stConverPixelParam.nWidth = FrameInfo.nWidth;
    stConverPixelParam.nHeight = FrameInfo.nHeight;
    stConverPixelParam.pSrcData = pBufForDriver;
    stConverPixelParam.nSrcDataLen = FrameInfo.nFrameLen;
    stConverPixelParam.enSrcPixelFormat = FrameInfo.enPixelFormat;
    stConverPixelParam.enDstPixelFormat = MyCamera.MvGvspPixelFormat.PixelType_Gvsp_RGB8_Packed;
    stConverPixelParam.pDstBuffer = pBufForSaveImage;
    stConverPixelParam.nDstBufferSize = (uint)(FrameInfo.nWidth * FrameInfo.nHeight * 3 + 2048);

    nRet = device.MV_CC_ConvertPixelFormat_NET(ref stConverPixelParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Convert pixel type Failed:{0:x8}", nRet);
        break;
    }

    // Save image data to local
    byte[] data = new byte[stConverPixelParam.nDstLen];
    Marshal.Copy(pBufForSaveImage, data, 0, (int)stConverPixelParam.nDstLen);
    FileStream pFile = null;
    try
    {
        pFile = new FileStream("AfterConvert_RGB.raw", FileMode.Create);
        pFile.Write(data, 0, data.Length);
    }
}
```

```
        catch
        {
            Console.WriteLine("Save failed");
        }
        finally
        {
            pFile.Close();
        }
    }
    Marshal.FreeHGlobal(pBufForDriver);
    Marshal.FreeHGlobal(pBufForSaveImage);

    //Stop acquiring images
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

B.5 Enhance Image

The sample code below shows how to enhance the image of a camera by configuring contrast and sharpness.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ImageEnhance
{
    class ImageEnhance
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            IntPtr pDstData = IntPtr.Zero;

            do
            {
                // Enumerate devices.
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count :{0} \n", stDevList.nDeviceNum);
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;

                // Print camera information such as IP and user ID.
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
                {
                    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

                    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
                    {
                        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
```

```
uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
Console.WriteLine("[device " + i.ToString() + "]:");
Console.WriteLine("DevIP:" + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
Console.WriteLine("UserDefineName:" + stGigEDeviceInfo.chUserDefinedName + "\n");
}
else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("[device " + i.ToString() + "]:");
    Console.WriteLine("SerialNumber:" + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("UserDefineName:" + stUsb3DeviceInfo.chUserDefinedName + "\n");
}
}

Int32 nDevIndex = 0;
Console.Write("Please input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device.
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open the device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
}
```

```
        break;
    }

    // Get the optimal package size (GigE camera only).
    if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
    {
        int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
        if (nPacketSize > 0)
        {
            nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
            if (nRet != MyCamera.MV_OK)
            {
                Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
            }
        }
        else
        {
            Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
        }
    }

    // Set trigger mode to Off.
    nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set TriggerMode failed!");
        break;
    }

    Console.WriteLine("*****\n");
    Console.WriteLine("* 0.Contrast; 1.Sharpen;                *\n");
    Console.WriteLine("*****\n");
    Console.WriteLine("Select enhance type:");
    Int32 nType = Convert.ToInt32(Console.ReadLine());
    if (nType != 0 && nType != 1)
    {
        Console.WriteLine("Input error", nRet);
        break;
    }

    // Start image acquisition.
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    MyCamera.MV_FRAME_OUT stFrameOut = new MyCamera.MV_FRAME_OUT();

    nRet = device.MV_CC_GetImageBuffer_NET(ref stFrameOut, 1000);
    // Get one frame.
```



```
if (MyCamera.MV_OK == nRet)
{
    Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stFrameOut.stFrameInfo.nWidth) + "],
Height["
        + Convert.ToString(stFrameOut.stFrameInfo.nHeight) + "], FrameNum[" +
Convert.ToString(stFrameOut.stFrameInfo.nFrameNum) + "]);

    if (pDstData == IntPtr.Zero)
    {
        pDstData = Marshal.AllocHGlobal((int)(stFrameOut.stFrameInfo.nFrameLen));
    }

    if (0 == nType)// Contrast parameters
    {
        MyCamera.MV_CC_CONTRAST_PARAM stContrastParam = new MyCamera.MV_CC_CONTRAST_PARAM();
        stContrastParam.nWidth = stFrameOut.stFrameInfo.nWidth;
        stContrastParam.nHeight = stFrameOut.stFrameInfo.nHeight;
        stContrastParam.enPixelFormat = stFrameOut.stFrameInfo.enPixelFormat;
        stContrastParam.pSrcBuf = stFrameOut.pBufAddr;
        stContrastParam.nSrcBufLen = stFrameOut.stFrameInfo.nFrameLen;

        stContrastParam.pDstBuf = pDstData;
        stContrastParam.nDstBufSize = stFrameOut.stFrameInfo.nFrameLen;
        stContrastParam.nContrastFactor = 1000;
        nRet = device.MV_CC_ImageContrast_NET(ref stContrastParam);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Adjust image contrast Failed:{0:x8}", nRet);
            break;
        }
    }
    else if (1 == nType)// Sharpness parameters.
    {
        MyCamera.MV_CC_SHARPEN_PARAM stSharpenParam = new MyCamera.MV_CC_SHARPEN_PARAM();
        stSharpenParam.nWidth = stFrameOut.stFrameInfo.nWidth;
        stSharpenParam.nHeight = stFrameOut.stFrameInfo.nHeight;
        stSharpenParam.enPixelFormat = stFrameOut.stFrameInfo.enPixelFormat;
        stSharpenParam.pSrcBuf = stFrameOut.pBufAddr;
        stSharpenParam.nSrcBufLen = stFrameOut.stFrameInfo.nFrameLen;

        stSharpenParam.pDstBuf = pDstData;
        stSharpenParam.nDstBufSize = stFrameOut.stFrameInfo.nFrameLen;

        stSharpenParam.nSharpenAmount = 350;
        stSharpenParam.nSharpenRadius = 10;
        stSharpenParam.nSharpenThreshold = 50;
        nRet = device.MV_CC_ImageSharpen_NET(ref stSharpenParam);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Image Sharpen Failed:{0:x8}", nRet);
            break;
        }
    }
}
```

```
}

// Save image data to local file.
byte[] data = new byte[stFrameOut.stFrameInfo.nFrameLen];
Marshal.Copy(stFrameOut.pBufAddr, data, 0, (int)stFrameOut.stFrameInfo.nFrameLen);
FileStream pFile = null;
try
{
    pFile = new FileStream("BeforeEnhance.raw", FileMode.Create);
    pFile.Write(data, 0, data.Length);
}
catch
{
    Console.WriteLine("Image enhance fail");
}
finally
{
    pFile.Close();
}

device.MV_CC_FreelImageBuffer_NET(ref stFrameOut);

Marshal.Copy(pDstData, data, 0, (int)stFrameOut.stFrameInfo.nFrameLen);
try
{
    pFile = new FileStream("AfterEnhance.raw", FileMode.Create);
    pFile.Write(data, 0, data.Length);
}
catch
{
    Console.WriteLine("Image enhance fail");
}
finally
{
    pFile.Close();
}

Console.WriteLine("Image enhance succeed");
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}

// Stop image acquisition.
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}
```

```
// Close the device.
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy the device.
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (pDstData != IntPtr.Zero)
{
    Marshal.FreeHGlobal(pDstData);
}

if (MyCamera.MV_OK != nRet)
{
    // Destroy the device.
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

B.6 File Access

Export the User Set or DPC (Defective Pixel Correction) file of a connected camera to the local PC as a binary file, or import a binary file from the local PC to a connected camera.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Threading;
using System.Runtime.InteropServices;
using System.IO;
```

```
namespace ParametrizeCamera_FileAccess
{
    class Program
    {
        public static MyCamera device;
        public static uint g_nMode = 0;
        public static int g_nRet = MyCamera.MV_OK;

        static void FileAccessProgress()
        {
            int nRet = MyCamera.MV_OK;
            MyCamera.MV_CC_FILE_ACCESS_PROGRESS stFileAccessProgress = new
MyCamera.MV_CC_FILE_ACCESS_PROGRESS();

            while (true)
            {
                //Get file access progress
                nRet = device.MV_CC_GetFileAccessProgress_NET(ref stFileAccessProgress);
                Console.WriteLine("State = {0:x8},Completed = {1},Total = {2}", nRet , stFileAccessProgress.nCompleted ,
stFileAccessProgress.nTotal);
                if (nRet != MyCamera.MV_OK || (stFileAccessProgress.nCompleted != 0 && stFileAccessProgress.nCompleted
== stFileAccessProgress.nTotal))
                {
                    break;
                }

                Thread.Sleep(50);
            }
        }

        static void FileAccessThread()
        {
            MyCamera.MV_CC_FILE_ACCESS stFileAccess = new MyCamera.MV_CC_FILE_ACCESS();

            stFileAccess.pUserFileName = "UserSet1.bin";
            stFileAccess.pDevFileName = "UserSet1";
            if (1 == g_nMode)
            {
                //Reading mode
                g_nRet = device.MV_CC_FileAccessRead_NET(ref stFileAccess);
                if (MyCamera.MV_OK != g_nRet)
                {
                    Console.WriteLine("File Access Read failed:{0:x8}", g_nRet);
                }
            }
            else if (2 == g_nMode)
            {
                //Writting mode
                g_nRet = device.MV_CC_FileAccessWrite_NET(ref stFileAccess);
                if (MyCamera.MV_OK != g_nRet)
                {
                    Console.WriteLine("File Access Write failed:{0:x8}", g_nRet);
                }
            }
        }
    }
}
```

```
    }
    }
}

static void Main(string[] args)
{
    device = new MyCamera();
    int nRet = MyCamera.MV_OK;

    do
    {
        // Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info

        //Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
```

```
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Reading mode
Console.WriteLine("Read to file");
g_nMode = 1;

Thread hReadHandle = new Thread(FileAccessThread);
hReadHandle.Start();

Thread.Sleep(5);
```

```
Thread hReadProgressHandle = new Thread(FileAccessProgress);
hReadProgressHandle.Start();

hReadProgressHandle.Join();
hReadHandle.Join();
if (MyCamera.MV_OK == g_nRet)
{
    Console.WriteLine("File Access Read Success");
}

Console.WriteLine("");

//Writting mode
Console.WriteLine("Write to file");
g_nMode = 2;

Thread hWriteHandle = new Thread(FileAccessThread);
hWriteHandle.Start();

Thread.Sleep(5);

Thread hWriteProgressHandle = new Thread(FileAccessProgress);
hWriteProgressHandle.Start();

hWriteProgressHandle.Join();
hWriteHandle.Join();
if (MyCamera.MV_OK == g_nRet)
{
    Console.WriteLine("File Access Write Success");
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
```

```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.7 Get Images Directly

The sample code below shows how to get images directly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace GrabImage
{
    class GrabImage
    {
        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = new MyCamera();
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
                {
                    break;
                }

                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info
```



```
// Print device info
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
    typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

// Set trigger mode to off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Get package size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
UInt32 nPayloadSize = stParam.nCurValue;

//Start acquiring images
```

```
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

int nCount = 0;
IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nPayloadSize);
IntPtr pBufForSaveImage = IntPtr.Zero;

MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (nCount++ != 10)
{
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, ref FrameInfo, 1000);
    //Get one image
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(FrameInfo.nWidth) + "], Height[" +
Convert.ToString(FrameInfo.nHeight)
            + "], FrameNum[" + Convert.ToString(FrameInfo.nFrameNum) + "]);

        if (pBufForSaveImage == IntPtr.Zero)
        {
            pBufForSaveImage = Marshal.AllocHGlobal((int)(FrameInfo.nHeight * FrameInfo.nWidth * 3 + 2048));
        }

        MyCamera.MV_SAVE_IMAGE_PARAM_EX stSaveParam = new
MyCamera.MV_SAVE_IMAGE_PARAM_EX();
        stSaveParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Bmp;
        stSaveParam.enPixelType = FrameInfo.enPixelType;
        stSaveParam.pData = pBufForDriver;
        stSaveParam.nDataLen = FrameInfo.nFrameLen;
        stSaveParam.nHeight = FrameInfo.nHeight;
        stSaveParam.nWidth = FrameInfo.nWidth;
        stSaveParam.plImageBuffer = pBufForSaveImage;
        stSaveParam.nBufferSize = (uint)(FrameInfo.nHeight * FrameInfo.nWidth * 3 + 2048);
        stSaveParam.nJpgQuality = 80;
        nRet = device.MV_CC_SaveImageEx_NET(ref stSaveParam);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Save Image failed:{0:x8}", nRet);
            continue;
        }

        //Save image data to local
        byte[] data = new byte[stSaveParam.nImageLen];
        Marshal.Copy(pBufForSaveImage, data, 0, (int)stSaveParam.nImageLen);
        FileStream pFile = null;
        try
        {
            pFile = new FileStream("frame" + nCount.ToString() + ".bmp", FileMode.Create);
```

```
        pFile.Write(data, 0, data.Length);
    }
    catch
    {
        Console.WriteLine("Saving image failed");
    }
    finally
    {
        pFile.Close();
    }
    continue;
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}
}
Marshal.FreeHGlobal(pBufForDriver);
Marshal.FreeHGlobal(pBufForSaveImage);

//Stop acquiring images
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
```

```
    }  
  }  
  
  Console.WriteLine("Press enter to exit");  
  Console.ReadKey();  
}  
}
```

B.8 Get Images Directly with High Performance

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

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace GrabImage_HighPerformance  
{  
    class GrabImage_HighPerformance  
    {  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;  
            MyCamera device = new MyCamera();  
            do  
            {  
                // Enumerate device  
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,  
ref stDevList);  
                if (MyCamera.MV_OK != nRet)  
                {  
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);  
                    break;  
                }  
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));  
                if (0 == stDevList.nDeviceNum)  
                {  
                    break;  
                }  
  
                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info  
  
                //Print device info  
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
```

```
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

//Set trigger mode to off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Start acquiring image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

int nCount = 0;
MyCamera.MV_FRAME_OUT FrameInfo = new MyCamera.MV_FRAME_OUT();
while (nCount++ != 100)
{
    nRet = device.MV_CC_GetImageBuffer_NET(ref FrameInfo, 1000);
    // Get one image
```

```
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(FrameInfo.stFrameInfo.nWidth) +
                "], Height[" + Convert.ToString(FrameInfo.stFrameInfo.nHeight)
                + "], FrameNum[" + Convert.ToString(FrameInfo.stFrameInfo.nFrameNum) + "]);

            if (FrameInfo.pBufAddr != IntPtr.Zero)
            {
                nRet = device.MV_CC_FreelImageBuffer_NET(ref FrameInfo);
                if (nRet != MyCamera.MV_OK)
                {
                    Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
                }
            }
        }
        else
        {
            Console.WriteLine("No data:{0:x8}", nRet);
        }
    }

    //Stop acquiring image
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    // Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
```



```
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.9 Get Images in Callback Function

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

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace Grab_Callback
{
    class Grab_Callback
    {
        public static MyCamera.cbOutputExdelegate ImageCallback;
        public static MyCamera device = new MyCamera();
        static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
        {
            Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
            Convert.ToString(pFrameInfo.nHeight)
                + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
        }

        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
            {
                //Enumerate device
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
                ref stDevList);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);
                    break;
                }
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
                if (0 == stDevList.nDeviceNum)
```

```
{
    break;
}

MyCamera.MV_CC_DEVICE_INFO stDevInfo;           // General device info

//Print device info
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.Write("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
}
```

```
        break;
    }
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    //Create device
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        break;
    }

    //Open device
    nRet = device.MV_CC_OpenDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Open device failed:{0:x8}", nRet);
        break;
    }

    //Detection network optimal package size(It only works for the GigE camera)
    if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
    {
        int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
        if (nPacketSize > 0)
        {
            nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
            if (nRet != MyCamera.MV_OK)
            {
                Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
            }
        }
        else
        {
            Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
        }
    }

    // Set trigger mode to off
    nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set TriggerMode failed!");
        break;
    }

    //Register image callback function
    ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
    nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
    if (MyCamera.MV_OK != nRet)
    {
```

```
        Console.WriteLine("Register image callback failed!");
        break;
    }

    //start acquiring images
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadLine();

    // Stop acquiring images
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
```

```
}  
}  
}
```

B.10 Get Images by Strategy

The sample code shows how to get image by different strategies.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
using System.Threading;  
  
namespace GrabImage  
{  
    class GrabStrategies  
    {  
        public static void UpcomingThread(object obj)  
        {  
            Thread.Sleep(3000);  
  
            MyCamera device = obj as MyCamera;  
            device.MV_CC_SetCommandValue_NET("TriggerSoftware");  
        }  
  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;  
            MyCamera device = new MyCamera();  
            do  
            {  
                // Enumerate device  
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,  
ref stDevList);  
                if (MyCamera.MV_OK != nRet)  
                {  
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);  
                    break;  
                }  
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));  
                if (0 == stDevList.nDeviceNum)  
                {  
                    break;  
                }  
            }  
        }  
    }  
}
```

```
MyCamera.MV_CC_DEVICE_INFO stDevInfo; // General device info

//Print device info
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("Device Number : " + stUsb3DeviceInfo.chModelName);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
// ch:创建 | en:Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

// :Set trigger mode and trigger source
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerMode", "On");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Mode failed:{0:x8}", nRet);
    break;
}
nRet = device.MV_CC_SetEnumValueByString_NET("TriggerSource", "Software");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Trigger Source failed:{0:x8}", nRet);
    break;
}

UInt32 nImageNodeNum = 5;
// Set number of image node
nRet = device.MV_CC_SetImageNodeNum_NET(nImageNodeNum);
```

```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set number of image node fail:{0:x8}", nRet);
            break;
        }

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

Console.WriteLine("*****");

        Console.Write("Please Input Grab Strategy:");
        UInt32 nGrabStrategy = 0;
        try
        {
            nGrabStrategy = (UInt32)Convert.ToInt32(Console.ReadLine());
        }
        catch
        {
            Console.Write("Invalid Input!\n");
            break;
        }

        //U3V device does not support UpcomingImage
        if (nGrabStrategy == (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage
            && MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
        {
            Console.Write("U3V device not support UpcomingImage\n");
            break;
        }

        switch(nGrabStrategy)
        {
        case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne:
            {
                Console.Write("Grab using the MV_GrabStrategy_OneByOne default strategy\n");
                nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_OneByOne);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
                    break;
                }
            }
            break;
        case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly:
            {
                Console.Write("Grab using strategy MV_GrabStrategy_LatestImagesOnly\n");
                nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImagesOnly);
```



```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
    }
    break;
case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_LatestImages\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_LatestImages);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }

        //Set the number of output buffers
        nRet = device.MV_CC_SetOutputQueueSize_NET(2);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }
    }
    break;
case (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage:
    {
        Console.WriteLine("Grab using strategy MV_GrabStrategy_UpcomingImage\n");
        nRet =
device.MV_CC_SetGrabStrategy_NET(MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Set Grab Strategy fail:{0:x8}", nRet);
            break;
        }

        Thread hUpcomingThread = new Thread(UpcomingThread);
        hUpcomingThread.Start(device);
    }
    break;
default:
    Console.WriteLine("Input error!Use default strategy: MV_GrabStrategy_OneByOne\n");
    break;
}

// Start acquiring image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
}
```

```
        break;
    }

    // Send software trigger command
    for (UInt32 i = 0; i < nImageNodeNum; i++)
    {
        nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Send Trigger Software command fail:{0:x8}", nRet);
            break;
        }
        Thread.Sleep(500);
    }

    MyCamera.MV_FRAME_OUT stOutFrame = new MyCamera.MV_FRAME_OUT();
    if (nGrabStrategy != (UInt32)MyCamera.MV_GRAB_STRATEGY.MV_GrabStrategy_UpcomingImage)
    {
        while(true)
        {
            nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 0);
            if (MyCamera.MV_OK == nRet)
            {
                Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth)
+ "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
+ "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);
            }
            else
            {
                Console.WriteLine("No data:{0:x8}", nRet);
                break;
            }

            nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
            }
        }
    }
    else//用于 upcoming
    {
        nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 5000);
        if (MyCamera.MV_OK == nRet)
        {
            Console.WriteLine("Get Image Buffer:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
" ], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
+ " ], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);

            nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
            if (MyCamera.MV_OK != nRet)
            {
```

```
        Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
    }
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}
}

//Stop acquiring image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.11 Get Images via Precision Time Protocol

The sample code below shows how to get images via precision time protocol.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;

namespace Grab_ActionCommand
{
    class Grab_ActionCommand
    {
        static bool g_bExit = false;
        static uint g_DeviceKey = 1;
        static uint g_GroupKey = 1;
        static uint g_GroupMask = 1;
        static uint g_nPayloadSize = 0;

        public static void ActionCommandWorkThread(object obj)
        {
            MyCamera device = obj as MyCamera;
            int nRet = MyCamera.MV_OK;
            MyCamera.MV_ACTION_CMD_INFO stActionCmdInfo = new MyCamera.MV_ACTION_CMD_INFO();
            MyCamera.MV_ACTION_CMD_RESULT_LIST stActionCmdResults = new
MyCamera.MV_ACTION_CMD_RESULT_LIST();

            stActionCmdInfo.nDeviceKey = g_DeviceKey;
            stActionCmdInfo.nGroupKey = g_GroupKey;
            stActionCmdInfo.nGroupMask = g_GroupMask;
            stActionCmdInfo.pBroadcastAddress = "255.255.255.255";
            stActionCmdInfo.nTimeOut = 100;
            stActionCmdInfo.bActionTimeEnable = 0;

            MyCamera.MV_ACTION_CMD_RESULT pResults = new MyCamera.MV_ACTION_CMD_RESULT();
            int size = Marshal.SizeOf(pResults);
            while (!g_bExit)
            {
                //Send the PTP clock photo command
                nRet = device.MV_GIGE_IssueActionCommand_NET(ref stActionCmdInfo, ref stActionCmdResults);
                if (MyCamera.MV_OK != nRet)
                {
                    Console.WriteLine("Issue Action Command failed! nRet {0:x8}", nRet);
                    continue;
                }
            }
        }
    }
}
```

```
MyCamera.MV_ACTION_CMD_RESULT stTempActionCmd = new MyCamera.MV_ACTION_CMD_RESULT();
var len = Marshal.SizeOf(stTempActionCmd) * stActionCmdResults.nNumResults;
var targetPtr = Marshal.AllocHGlobal((int)len);
unsafe
{
    byte* srcPtr = (byte*)stActionCmdResults.pResults.ToPointer();
    byte* tmpPtr = (byte*)targetPtr.ToPointer();

    for (int i = 0; i < len; i++)
    {
        *(tmpPtr + i) = *(srcPtr + i);
    }

    MyCamera.MV_ACTION_CMD_RESULT[] arrayMvActionCmdResult =
PtrToStructurs<MyCamera.MV_ACTION_CMD_RESULT>(targetPtr, (int)stActionCmdResults.nNumResults);

    for (uint i = 0; i < stActionCmdResults.nNumResults; i++)
    {
        //print the device infomation
        Console.WriteLine("Ip == " + arrayMvActionCmdResult[i].strDeviceAddress + ", Status == " +
Convert.ToInt32(arrayMvActionCmdResult[i].nStatus));
    }
    Marshal.FreeHGlobal(targetPtr);
}

public unsafe static T[] PtrToStructurs<T>(IntPtr pt, int lenth)
{
    T[] structur = new T[lenth];
    for (int i = 0; i < lenth; i++)
    {
        IntPtr ptr = new IntPtr((int)pt + (i * Marshal.SizeOf(typeof(T))));
        structur[i] = (T)Marshal.PtrToStructure(ptr, typeof(T));
    }
    return structur;
}

public static void ReceivelImageWorkThread(object obj)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = obj as MyCamera;
    MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
    IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);
    if (pData == IntPtr.Zero)
    {
        return;
    }
    uint nDataSize = g_nPayloadSize;

    while (true)
    {
```

```
nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
if (nRet == MyCamera.MV_OK)
{
    Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);
}
else
{
    Console.WriteLine("No data:{0:x8}", nRet);
}
if (g_bExit)
{
    break;
}
}
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    do
    {
        // ch:枚举 | en:Enum device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info

        //Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
```

```
uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
    Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
}
else
{
    Console.WriteLine("Not Support!\n");
    break;
}
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
```

```
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Setting Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Getting Packet Size failed {0:x8}", nPacketSize);
    }
}

// Set trigger mode to on
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 1);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Setting Trigger Mode failed! {0:x8}", nRet);
    break;
}

//Set the trigger source as "Action"
nRet = device.MV_CC_SetEnumValue_NET("TriggerSource", 9);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Setting Trigger Source failed! {0:x8}", nRet);
    break;
}

//Set device key
nRet = device.MV_CC_SetIntValue_NET("ActionDeviceKey", g_DeviceKey);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Setting Action Device Key failed! {0:x8}", nRet);
    break;
}

//Set the key of group address
nRet = device.MV_CC_SetIntValue_NET("ActionGroupKey", g_GroupKey);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Setting Action Group Key failed! {0:x8}", nRet);
    break;
}

//Set the mask of group address
nRet = device.MV_CC_SetIntValue_NET("ActionGroupMask", g_GroupMask);
if (MyCamera.MV_OK != nRet)
{

```



```
        Console.WriteLine("Setting Action Group Mask failed! {0:x8}", nRet);
        break;
    }

    //Get package size
    MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
    nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
        break;
    }
    g_nPayloadSize = stParam.nCurValue;

    // Start acquiring image
    nRet = device.MV_CC_StartGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
        break;
    }

    Thread hActionCommandThreadHandle = new Thread(ActionCommandWorkThread);
    hActionCommandThreadHandle.Start(device);

    Thread hReceiveImageThreadHandle = new Thread(ReceiveImageWorkThread);
    hReceiveImageThreadHandle.Start(device);

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();

    g_bExit = true;
    Thread.Sleep(1000);

    //Stop acquiring image
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    // Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
```

```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
            break;
        }
    } while (false);

    if (MyCamera.MV_OK != nRet)
    {
        //Destroy device
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.12 Get Camera Events

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

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace Events
{
    class Events
    {
        public static MyCamera.cbEventdelegateEx EventCallback;
        public static MyCamera device;

        static void EventCallbackFunc(ref MyCamera.MV_EVENT_OUT_INFO pEventInfo, IntPtr pUser)
        {
            Console.WriteLine("EventName[" + pEventInfo.EventName + "], EventID[" + pEventInfo.nEventID + "]);
        }

        static void Main(string[] args)
        {
            int nRet = MyCamera.MV_OK;
            do
```

```
{
    //Enumerate device
    MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
    nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Enum device failed:{0:x8}", nRet);
        break;
    }
    Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
    if (0 == stDevList.nDeviceNum)
    {
        break;
    }

    MyCamera.MV_CC_DEVICE_INFO stDevInfo;

    //Print device info
    for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
    {
        stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  : ", stDevList.nDeviceNum - 1);
    try
```

```
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
device = new MyCamera();
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}
```

```
//Set trigger mode to off
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// Enable ExposureEnd
nRet = device.MV_CC_SetEnumValueByString_NET("EventSelector", "ExposureEnd");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set EventSelector failed!");
    break;
}

nRet = device.MV_CC_SetEnumValueByString_NET("EventNotification", "On");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set EventNotification failed!");
    break;
}

//Register event callback function
EventCallback = new MyCamera.cbEventdelegateEx(EventCallbackFunc);
nRet = device.MV_CC_RegisterEventCallBackEx_NET("ExposureEnd", EventCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register event callback failed!");
    break;
}

// Start acquiring image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Push enter to exit");
Console.ReadLine();

//Stop acquiring image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
```

```
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.13 Get Chunk Information

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ChunkData
{
    class ChunkData
    {

```

```
public static MyCamera.cbOutputExdelegate ImageCallback;
static MyCamera.MV_CHUNK_DATA_CONTENT stChunkInfo;// Chunk structure info
static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
{
    //Print parse the timestamp information in the frame
    Console.WriteLine("ImageCallBack: ExposureTime[" + Convert.ToString(pFrameInfo.fExposureTime)
        + "], SecondCount[" + Convert.ToString(pFrameInfo.nSecondCount)
        + "], CycleCount[" + Convert.ToString(pFrameInfo.nCycleCount)
        + "], CycleOffset[" + Convert.ToString(pFrameInfo.nCycleOffset)
        + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);

    int nStrSize = Marshal.SizeOf(stChunkInfo);
    int nUnparsedChunkContent = (int)pFrameInfo.UnparsedChunkList.pUnparsedChunkContent;
    for (int i = 0; i < pFrameInfo.nUnparsedChunkNum; i++)
    {
        stChunkInfo = (MyCamera.MV_CHUNK_DATA_CONTENT)Marshal.PtrToStructure((IntPtr)
            (nUnparsedChunkContent + i * nStrSize), typeof(MyCamera.MV_CHUNK_DATA_CONTENT));

        Console.WriteLine("ChunkInfo:" + "ChunkID[0x{0:x8}],ChunkLen[" + Convert.ToString(stChunkInfo.nChunkLen)
            + "]",stChunkInfo.nChunkID);
    }
    Console.WriteLine("*****");
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    do
    {
        // ch:枚举 | en:Enum device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device information

        // ch:打印信息 en:Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
                typeof(MyCamera.MV_CC_DEVICE_INFO));
        }
    }
}
```

```
        if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
            uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
            uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
            uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
            uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
            Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
            Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
        }
        else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
        {
            MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
            Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
            Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
            Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
        }
    }

    Int32 nDevIndex = 0;
    Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
    try
    {
        nDevIndex = Convert.ToInt32(Console.ReadLine());
    }
    catch
    {
        Console.WriteLine("Invalid Input!\n");
        break;
    }

    if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
    {
        Console.WriteLine("Input Error!\n");
        break;
    }
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

    //Create device
    nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        break;
    }
}
```



```
//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

//Register image callback function
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}

// Enable Chunk Mode
nRet = device.MV_CC_SetBoolValue_NET("ChunkModeActive", true);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Chunk Mode failed:{0:x8}", nRet);
    break;
}

//Set Chunk Selector as Exposure
nRet = device.MV_CC_SetEnumValueByString_NET("ChunkSelector", "Exposure");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Exposure Chunk failed:{0:x8}", nRet);
    break;
}
```

```
// ch:开启 Chunk Enable | en:Open Chunk Enable
nRet = device.MV_CC_SetBoolValue_NET("ChunkEnable", true);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Chunk Enable failed:{0:x8}", nRet);
    break;
}

//Set Chunk Selector to Timestamp
nRet = device.MV_CC_SetEnumValueByString_NET("ChunkSelector", "Timestamp");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Timestamp Chunk failed:{0:x8}", nRet);
    break;
}

// ch:开启 Chunk Enable | en:Open Chunk Enable
nRet = device.MV_CC_SetBoolValue_NET("ChunkEnable", true);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set Chunk Enable failed:{0:x8}", nRet);
    break;
}

// Set the trigger mode as off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

// ch:开启抓图 | en:start grab
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

// Stop getting images
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
```

```
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Close device failed{0:x8}", nRet);
            break;
        }

        // ch:???? | en:Destroy device
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
            break;
        }
    } while (false);

    if (MyCamera.MV_OK != nRet)
    {
        //en:Destroy device
        nRet = device.MV_CC_DestroyDevice_NET();
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        }
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.14 Import/Export Camera Feature File

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

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace ParametrizeCamera_LoadAndSave
{
    class ParametrizeCamera_LoadAndSave
    {

```

```
static void Main(string[] args)
{
    MyCamera device = new MyCamera();
    int nRet = MyCamera.MV_OK;

    do{
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info

        // Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
                uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
                uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
                uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
                uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
                Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
                Console.WriteLine("device IP : " + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
            }
            else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
            {
                MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
                Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
                Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
                Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
            }
        }
    }
}
```

```
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Start export the camera properties to the file");
Console.WriteLine("Wait.....");

// en:Export the camera properties to the file
nRet = device.MV_CC_FeatureSave_NET("CameraFile");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("FeatureSave failed!");
    break;
}
Console.WriteLine("Finish export the camera properties to the file\n");

Console.WriteLine("Start import the camera properties from the file");
Console.WriteLine("Wait.....");
```

```
// en:Import the camera properties from the file
nRet = device.MV_CC_FeatureLoad_NET("CameraFile");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("FeatureLoad failed!");
    break;
}
Console.WriteLine("Finish import the camera properties from the file");

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.15 Perform Basic Functions of CamLink Cameras

Perform the basic functions of CamLink cameras, including connecting cameras, acquiring images, setting parameters, and so on.

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace CamLBasicDemo
{
    class CamLBasicDemo
    {
        static MyCamera.cbExceptiondelegate pCallBackFunc;
        // Callback function
        static void cbExceptiondelegate(uint nMsgType, IntPtr pUser)
        {
            if (nMsgType == MyCamera.MV_EXCEPTION_DEV_DISCONNECT)
            {
                Console.WriteLine("MV_EXCEPTION_DEV_DISCONNECT");
            }
        }

        // Get the value of various feature nodes
        static int GetParameters(ref MyCamera device)
        {
            if (null == device)
            {
                return MyCamera.MV_E_PARAMETER;
            }

            int nRet = MyCamera.MV_OK;

            // Get value of Integer nodes. Such as, 'width' etc.
            MyCamera.MVCC_INTVALUE stIntVal = new MyCamera.MVCC_INTVALUE();
            nRet = device.MV_CC_GetIntValue_NET("Width", ref stIntVal);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Get width failed:{0:x8}", nRet);
                return nRet;
            }
            Console.WriteLine("Current Width:{0:d}", stIntVal.nCurValue);

            // Get value of Enum nodes. Such as, 'TriggerMode' etc.
            MyCamera.MVCC_ENUMVALUE stEnumVal = new MyCamera.MVCC_ENUMVALUE();
            nRet = device.MV_CC_GetEnumValue_NET("TriggerMode", ref stEnumVal);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Get Trigger Mode failed:{0:x8}", nRet);
                return nRet;
            }
            Console.WriteLine("Current TriggerMode:{0:d}", stEnumVal.nCurValue);
        }
    }
}
```

```
// Get value of float nodes. Such as, 'AcquisitionFrameRate' etc.
MyCamera.MVCC_FLOATVALUE stFloatVal = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("AcquisitionFrameRate", ref stFloatVal);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get AcquisitionFrameRate failed:{0:x8}", nRet);
    return nRet;
}
Console.WriteLine("Current AcquisitionFrameRate:{0:f}Fps", stFloatVal.fCurValue);

// Get value of bool nodes. Such as, 'AcquisitionFrameRateEnable' etc.
bool bBoolVal = false;
nRet = device.MV_CC_GetBoolValue_NET("AcquisitionFrameRateEnable", ref bBoolVal);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get AcquisitionFrameRateEnable failed:{0:x8}", nRet);
    return nRet;
}
Console.WriteLine("Current AcquisitionFrameRateEnable:{0:d}", bBoolVal);

// Get value of String nodes. Such as, 'DeviceUserID' etc.
MyCamera.MVCC_STRINGVALUE stStrVal = new MyCamera.MVCC_STRINGVALUE();
nRet = device.MV_CC_GetStringValue_NET("DeviceUserID", ref stStrVal);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get DeviceUserID failed:{0:x8}", nRet);
    return nRet;
}
Console.WriteLine("Current DeviceUserID:{0:s}", stStrVal.chCurValue);

return MyCamera.MV_OK;
}

// Set the value of various feature nodes
static int SetParameters(ref MyCamera device)
{
    if (null == device)
    {
        return MyCamera.MV_E_PARAMETER;
    }

    int nRet = MyCamera.MV_OK;

    // Set value of Integer nodes. Such as, 'width' etc.
    nRet = device.MV_CC_SetIntValue_NET("Width", 200);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set Width failed:{0:x8}", nRet);
        return nRet;
    }
}
```



```
// Set value of float nodes. Such as, 'AcquisitionFrameRate' etc.
nRet = device.MV_CC_SetFloatValue_NET("AcquisitionFrameRate", 8.8f);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set AcquisitionFrameRate failed:{0:x8}", nRet);
    return nRet;
}

// Set value of bool nodes. Such as, 'AcquisitionFrameRateEnable' etc.
nRet = device.MV_CC_SetBoolValue_NET("AcquisitionFrameRateEnable", true);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set AcquisitionFrameRateEnable failed:{0:x8}", nRet);
    return nRet;
}

// Set value of String nodes. Such as, 'DeviceUserID' etc.
nRet = device.MV_CC_SetStringValue_NET("DeviceUserID", "UserIDChanged");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set DeviceUserID failed:{0:x8}", nRet);
    return nRet;
}

// Execute Command nodes. Such as, 'TriggerSoftware' etc.
// precondition
// Set value of Enum nodes. Such as, 'TriggerMode' etc.
nRet = device.MV_CC_SetEnumValue_NET("TriggerMode",
(uint)MyCamera.MV_CAM_TRIGGER_MODE.MV_TRIGGER_MODE_ON);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerMode failed:{0:x8}", nRet);
    return nRet;
}
nRet = device.MV_CC_SetEnumValue_NET("TriggerSource",
(uint)MyCamera.MV_CAM_TRIGGER_SOURCE.MV_TRIGGER_SOURCE_SOFTWARE);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set TriggerSource failed:{0:x8}", nRet);
    return nRet;
}
// execute command
nRet = device.MV_CC_SetCommandValue_NET("TriggerSoftware");
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Execute TriggerSoftware failed:{0:x8}", nRet);
    return nRet;
}

return MyCamera.MV_OK;
}
```

```
static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    bool bDevConnected = false; //whether a device is connected

    do
    {
        // Enum device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_CAMERALINK_DEVICE, ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;
        // Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
            typeof(MyCamera.MV_CC_DEVICE_INFO));
            if (MyCamera.MV_CAMERALINK_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_CamL_DEV_INFO stCamLDeviceInfo =
                (MyCamera.MV_CamL_DEV_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stCamLInfo,
                typeof(MyCamera.MV_CamL_DEV_INFO));
                Console.WriteLine(i.ToString() + ": [CamL] Serial Number : " + stCamLDeviceInfo.chSerialNumber);
                Console.WriteLine("PortID : " + stCamLDeviceInfo.chPortID);
                Console.WriteLine("chManufacturerName : " + stCamLDeviceInfo.chManufacturerName);
            }
            else
            {
                Console.WriteLine("Unknown Error.");
            }
        }
        Console.WriteLine("Enum finish.");

        Int32 nDevIndex = 0;
        Console.WriteLine("\nPlease input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
        try
        {
            nDevIndex = Convert.ToInt32(Console.ReadLine());
        }
        catch
```

```
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("Open finish.");
bDevConnected = true;

// Register Exception Callback
pCallbackFunc = new MyCamera.cbExceptiondelegate(cbExceptiondelegate);
nRet = device.MV_CC_RegisterExceptionCallBack_NET(pCallbackFunc, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register expection callback failed:{0:x8}", nRet);
    break;
}
GC.KeepAlive(pCallbackFunc);

/*****characteristic interfaces for CameraLink device*****/
// Get supported baudrates of the combined device and host interface
uint nBaudrateAblity = 0;
nRet = device.MV_CAML_GetSupportBauderates_NET(ref nBaudrateAblity);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get supported bauderate fail:{0:x8}", nRet);
    break;
}
Console.WriteLine("Current device supported bauderate:{0:x8}", nBaudrateAblity);
```

```
// Set device bauderate
nRet = device.MV_CAML_SetDeviceBauderate_NET((uint)MyCamera.MV_CAML_BAUDRATE_115200);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Set device bauderate fail:{0:x8}", nRet);
    break;
}

// Get the current device bauderate
uint nCurrentBaudrate = 0;
nRet = device.MV_CAML_GetDeviceBauderate_NET(ref nCurrentBaudrate);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get device bauderate fail:{0:x8}", nRet);
    break;
}
Console.WriteLine("Current device bauderate:{0:x8}", nCurrentBaudrate);

/*****properties configuration*****/
// Get the value of various feature nodes
nRet = GetParameters(ref device);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("GetParameters failed:{0:x8}", nRet);
    break;
}

// Set the value of various feature nodes
nRet = SetParameters(ref device);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SetParameters failed:{0:x8}", nRet);
    break;
}

// Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}
bDevConnected = false;

// Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
Console.WriteLine("\n Close finish.");
```

```
    } while (false);

    if (MyCamera.MV_OK != nRet)
    {
        // Ensure that the device is closed
        if ( bDevConnected )
        {
            device.MV_CC_CloseDevice_NET();
            bDevConnected = false;
        }
        // Destroy device
        device.MV_CC_DestroyDevice_NET();
    }

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();
}
}
```

B.16 Recording

Record video files.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;
using System.Threading;

namespace GrabImage
{
    class GrabImage
    {
        static bool g_bExit = false;
        static uint g_nPayloadSize = 0;
        public static void ReceiveImageWorkThread(object obj)
        {
            int nRet = MyCamera.MV_OK;
            MyCamera device = obj as MyCamera;
            MyCamera.MV_FRAME_OUT_INFO_EX stImageInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
            IntPtr pData = Marshal.AllocHGlobal((int)g_nPayloadSize);
            if (pData == IntPtr.Zero)
            {
                return;
            }
            uint nDataSize = g_nPayloadSize;
```

```
MyCamera.MV_CC_INPUT_FRAME_INFO stInputFrameInfo = new MyCamera.MV_CC_INPUT_FRAME_INFO();

while (true)
{
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pData, nDataSize, ref stImageInfo, 1000);
    if (nRet == MyCamera.MV_OK)
    {
        Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stImageInfo.nWidth) + "], Height[" +
Convert.ToString(stImageInfo.nHeight)
            + "], FrameNum[" + Convert.ToString(stImageInfo.nFrameNum) + "]);

        stInputFrameInfo.pData = pData;
        stInputFrameInfo.nDataLen = stImageInfo.nFrameLen;
        nRet = device.MV_CC_InputOneFrame_NET(ref stInputFrameInfo);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Input one frame failed: nRet {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
    if (g_bExit)
    {
        break;
    }
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    MyCamera device = new MyCamera();
    do
    {
        //Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info
```

```
//:Print device info
for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}

stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
    typeof(MyCamera.MV_CC_DEVICE_INFO));
```

```
//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
    if (nPacketSize > 0)
    {
        nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", (uint)nPacketSize);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

//Set trigger mode to off
if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
{
    Console.WriteLine("Set TriggerMode failed!");
    break;
}

//Get package size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
g_nPayloadSize = stParam.nCurValue;

MyCamera.MV_CC_RECORD_PARAM stRecordPar = new MyCamera.MV_CC_RECORD_PARAM();
```



```
nRet = device.MV_CC_GetIntValue_NET("Width", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.nWidth = (ushort)stParam.nCurValue;

nRet = device.MV_CC_GetIntValue_NET("Height", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Height failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.nHeight = (ushort)stParam.nCurValue;

MyCamera.MVCC_ENUMVALUE stEnumValue = new MyCamera.MVCC_ENUMVALUE();
nRet = device.MV_CC_GetEnumValue_NET("PixelFormat", ref stEnumValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Width failed: nRet {0:x8}", nRet);
    break;
}
stRecordPar.enPixelFormat = (MyCamera.MvGvspPixelFormat)stEnumValue.nCurValue;

MyCamera.MVCC_FLOATVALUE stFloatValue = new MyCamera.MVCC_FLOATVALUE();
nRet = device.MV_CC_GetFloatValue_NET("ResultingFrameRate", ref stFloatValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get Float value failed: nRet {0:x8}", nRet);
    break;
}
//Frame rate (1/16-120)fps
stRecordPar.fFrameRate = stFloatValue.fCurValue;

//Bit rate(128kbps-16Mbps)
stRecordPar.nBitRate = 1000;
//Recording format (currently, only support AVI)
stRecordPar.enRecordFmtType = MyCamera.MV_RECORD_FORMAT_TYPE.MV_FormatType_AVI;
stRecordPar.strFilePath = ".\\Recording.avi";
nRet = device.MV_CC_StartRecord_NET(ref stRecordPar);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start Record failed: nRet {0:x8}", nRet);
    break;
}

//Start acquiring
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
```

```
        break;
    }

    Thread hReceiveImageThreadHandle = new Thread(ReceiveImageWorkThread);
    hReceiveImageThreadHandle.Start(device);

    Console.WriteLine("Press enter to exit");
    Console.ReadKey();

    g_bExit = true;
    hReceiveImageThreadHandle.Join();

    // Stop acquiring image
    nRet = device.MV_CC_StopGrabbing_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
        break;
    }

    //Stop recording
    nRet = device.MV_CC_StopRecord_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Stop Record failed{0:x8}", nRet);
        break;
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
```

```
    }  
  }  
  
  Console.WriteLine("Press enter to exit");  
  Console.ReadKey();  
}  
}
```

B.17 Save Images of 3D Cameras in Point Cloud Format

The sample code shows how to save images of 3D cameras in point cloud format.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using MvCamCtrl.NET;  
using System.Runtime.InteropServices;  
using System.IO;  
  
namespace SavePonitCloudData_3D  
{  
    class SavePonitCloudData_3D  
    {  
        static void Main(string[] args)  
        {  
            int nRet = MyCamera.MV_OK;  
            MyCamera device = new MyCamera();  
            do  
            {  
                //Enumerate device  
                MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();  
                nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,  
ref stDevList);  
                if (MyCamera.MV_OK != nRet)  
                {  
                    Console.WriteLine("Enum device failed:{0:x8}", nRet);  
                    break;  
                }  
                Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));  
                if (0 == stDevList.nDeviceNum)  
                {  
                    break;  
                }  
  
                MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device info  
  
                //Print device info  
                for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
```

```
{
    stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
    typeof(MyCamera.MV_CC_DEVICE_INFO));

    if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
        (MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
        typeof(MyCamera.MV_GIGE_DEVICE_INFO));
        uint nlp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
        uint nlp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
        uint nlp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
        uint nlp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
        Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
        stGigEDeviceInfo.chUserDefinedName);
        Console.WriteLine("device IP : " + nlp1 + "." + nlp2 + "." + nlp3 + "." + nlp4);
    }
    else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
    {
        MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
        (MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
        typeof(MyCamera.MV_USB3_DEVICE_INFO));
        Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
        stUsb3DeviceInfo.chUserDefinedName);
        Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
        Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
    }
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

//Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
```

```
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

//Open device
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
    break;
}

// Check whether the device is set to 3D format
MyCamera.MVCC_ENUMVALUE EnumValue = new MyCamera.MVCC_ENUMVALUE();
nRet = device.MV_CC_GetEnumValue_NET("PixelFormat", ref EnumValue);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get the Camera format fail:{0:x8}", nRet);
    break;
}

MyCamera.MvGvspPixelFormat ePixelFormat = (MyCamera.MvGvspPixelFormat)EnumValue.nCurValue;
switch (ePixelFormat)
{
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_ABC32:
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_ABC32f:
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_AB32:
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_AB32f:
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_AC32:
    case MyCamera.MvGvspPixelFormat.PixelType_Gvsp_Coord3D_AC32f:
        {
            nRet = MyCamera.MV_OK;
            break;
        }

    default:
        {
            nRet = MyCamera.MV_E_SUPPORT;
            break;
        }
}
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("This is not a supported 3D format!");
    break;
}

//Detection network optimal package size(It only works for the GigE camera)
if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
{
    int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
```

```
if (nPacketSize > 0)
{
    nRet = device.MV_CC_SetIntValue_NET("GevSCPSPacketSize", Convert.ToInt32(nPacketSize));
    if (nRet != MyCamera.MV_OK)
    {
        Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
    }
}
else
{
    Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
}
}

//Get the access mode of the trigger mode
MyCamera.MV_XML_AccessMode pAccessMode = MyCamera.MV_XML_AccessMode.AM_NI;
if (MyCamera.MV_OK != device.MV_XML_GetNodeAccessMode_NET("TriggerMode", ref pAccessMode))
{
    Console.WriteLine("Get Access mode of trigger mode fail! nRet [0x%x]\n", nRet);
}
else
{
    // Set trigger mode to off
    if (MyCamera.MV_OK != device.MV_CC_SetEnumValue_NET("TriggerMode", 0))
    {
        Console.WriteLine("Set TriggerMode failed!");
        break;
    }
}

// Get package size
MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
    break;
}
UInt32 nPayloadSize = stParam.nCurValue;

// Start acquiring images
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

uint nImageNum = 100;
byte[] bSaveImageBuf = null;

try
```

```
{
    // Apply enough buffer to save the acquired images
    bSaveImageBuf = new byte[nPayloadSize * nImageNum];
}
catch (Exception ex)
{
    Console.WriteLine("Malloc Save buffer fail!\n");
    break;
}

uint nSaveImageSize = nPayloadSize * nImageNum;

// Total size of the images obtained
uint nSaveDataLen = 0;

MyCamera.MV_FRAME_OUT stOutFrame = new MyCamera.MV_FRAME_OUT();
for(uint i = 0; i < nImageNum; i++)
{
    nRet = device.MV_CC_GetImageBuffer_NET(ref stOutFrame, 1000);
    if (nRet == MyCamera.MV_OK)
    {
        Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(stOutFrame.stFrameInfo.nWidth) +
            "], Height[" + Convert.ToString(stOutFrame.stFrameInfo.nHeight)
            + "], FrameNum[" + Convert.ToString(stOutFrame.stFrameInfo.nFrameNum) + "]);

        if (nSaveImageSize > (nSaveDataLen + stOutFrame.stFrameInfo.nFrameLen))
        {
            // Copy one image to the buffer named pSaveImageBuf
            Marshal.Copy(stOutFrame.pBufAddr, bSaveImageBuf, Convert.ToInt32(nSaveDataLen),
                Convert.ToInt32(stOutFrame.stFrameInfo.nFrameLen));
            nSaveDataLen += stOutFrame.stFrameInfo.nFrameLen;
        }

        nRet = device.MV_CC_FreelImageBuffer_NET(ref stOutFrame);
        if (nRet != MyCamera.MV_OK)
        {
            Console.WriteLine("Free Image Buffer fail:{0:x8}", nRet);
        }
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
}

MyCamera.MV_SAVE_POINT_CLOUD_PARAM stSavePoCloudPar = new
MyCamera.MV_SAVE_POINT_CLOUD_PARAM();

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

byte[] bDstImageBuf = new byte[stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4)]
```

```
+ 2048];
uint nDstImageSize = stSavePoCloudPar.nLineNum * stSavePoCloudPar.nLinePntNum * (16 * 3 + 4) + 2048;

stSavePoCloudPar.enPointCloudFileType =
MyCamera.MV_SAVE_POINT_CLOUD_FILE_TYPE.MV_PointCloudFile_PLY;
stSavePoCloudPar.enSrcPixelFormat = stOutFrame.stFrameInfo.enPixelFormat;
stSavePoCloudPar.nSrcDataLen = nSaveDataLen;

GCHandle hSrcData = GCHandle.Alloc(bSaveImageBuf, GCHandleType.Pinned);
stSavePoCloudPar.pSrcData = hSrcData.AddrOfPinnedObject();

GCHandle hDstData = GCHandle.Alloc(bDstImageBuf, GCHandleType.Pinned);
stSavePoCloudPar.pDstBuf = hDstData.AddrOfPinnedObject();

stSavePoCloudPar.nDstBufSize = nDstImageSize;

//Save point cloud data
nRet = device.MV_CC_SavePointCloudData_NET(ref stSavePoCloudPar);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Save point cloud data fail:{0:x8}", nRet);
    break;
}

FileStream file = new FileStream("PointCloudData.ply", FileMode.Create, FileAccess.Write);
file.Write(bDstImageBuf, 0, Convert.ToInt32(stSavePoCloudPar.nDstBufLen));
file.Close();
Console.WriteLine("Save point cloud data succeed");

hSrcData.Free();
hDstData.Free();

//Stop acquiring image
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

//Close device
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

//Destroy device
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
```



```
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    //Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

B.18 Set Multicast Mode

Set the transport mode to multicast mode.

```
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.Threading;
using System.Collections;
using System.IO;

namespace MultiCast
{
    class MultiCast
    {
        private static MyCamera device;
        public static bool g_bExit = false;

        static void WorkThread()
        {
            // Get package size
            MyCamera.MVCC_INTVALUE stParam = new MyCamera.MVCC_INTVALUE();
            int nRet = device.MV_CC_GetIntValue_NET("PayloadSize", ref stParam);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Get PayloadSize failed:{0:x8}", nRet);
                return;
            }
            UInt32 nPayloadSize = stParam.nCurValue;
```

```
IntPtr pBufForDriver = Marshal.AllocHGlobal((int)nPayloadSize);
MyCamera.MV_FRAME_OUT_INFO_EX FrameInfo = new MyCamera.MV_FRAME_OUT_INFO_EX();
while (true)
{
    nRet = device.MV_CC_GetOneFrameTimeout_NET(pBufForDriver, nPayloadSize, ref FrameInfo, 1000);
    //:Get one image
    if (MyCamera.MV_OK == nRet)
    {
        Console.WriteLine("Get One Frame:" + "Width[" + Convert.ToString(FrameInfo.nWidth) + "], Height[" +
Convert.ToString(FrameInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(FrameInfo.nFrameNum) + "]);
    }
    else
    {
        Console.WriteLine("No data:{0:x8}", nRet);
    }
    if (g_bExit)
    {
        break;
    }
}
Marshal.FreeHGlobal(pBufForDriver);
return;
}

static void Main(string[] args)
{
    device = new MyCamera();
    int nRet = MyCamera.MV_OK;

    do{
        // Enumerate device
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;

        // Print device info
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
```

```
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nLayerType)
{
    MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
    uint nSepIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
    uint nSepIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
    uint nSepIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
    uint nSepIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
    Console.WriteLine("\n" + i.ToString() + ": [GigE] User Define Name : " +
stGigEDeviceInfo.chUserDefinedName);
    Console.WriteLine("device IP : " + nSepIp1 + "." + nSepIp2 + "." + nSepIp3 + "." + nSepIp4);
}
else if (MyCamera.MV_USB_DEVICE == stDevInfo.nLayerType)
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("\n" + i.ToString() + ": [U3V] User Define Name : " +
stUsb3DeviceInfo.chUserDefinedName);
    Console.WriteLine("\n Serial Number : " + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("\n Device Number : " + stUsb3DeviceInfo.nDeviceNumber);
}
}

Int32 nDevIndex = 0;
Console.WriteLine("\nPlease input index (0 -- {0:d}) : ", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.WriteLine("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.WriteLine("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{

```

```
        Console.WriteLine("Create device failed:{0:x8}", nRet);
        break;
    }

    // Query the mode used by the user
    bool monitorMode = false;
    {
        string key = "";

        // Ask the user to launch the multicast controlling application or the multicast monitoring application.
        Console.WriteLine("Start multicast sample in (c)ontrol or in (m)onitor mode? (c/m)\n");
        do
        {
            key = Convert.ToString(Console.ReadLine());
            while ((key != "c") && (key != "m") && (key != "C") && (key != "M"));
            monitorMode = (key == "m") || (key == "M");
        }

        //Open device
        if (monitorMode)
        {
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Monitor, 0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                break;
            }
        }
        else
        {
            nRet = device.MV_CC_OpenDevice_NET(MyCamera.MV_ACCESS_Control, 0);
            if (MyCamera.MV_OK != nRet)
            {
                Console.WriteLine("Open device failed:{0:x8}", nRet);
                break;
            }
        }
    }

    // Detection network optimal package size(It only works for the GigE camera)
    if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE && false == monitorMode)
    {
        int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
        if (nPacketSize > 0)
        {
            nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
            if (nRet != MyCamera.MV_OK)
            {
                Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
            }
        }
        else
        {
            {

```

```
        Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
    }
}

// Specified multicast IP
string strIp = "239.0.1.23";
var parts = strIp.Split('.');
int nIp1 = Convert.ToInt32(parts[0]);
int nIp2 = Convert.ToInt32(parts[1]);
int nIp3 = Convert.ToInt32(parts[2]);
int nIp4 = Convert.ToInt32(parts[3]);
int nIp = (nIp1 << 24) | (nIp2 << 16) | (nIp3 << 8) | nIp4;

// Multicast port
MyCamera.MV_CC_TRANSMISSION_TYPE stTransmissionType = new
MyCamera.MV_CC_TRANSMISSION_TYPE();

stTransmissionType.enTransmissionType =
MyCamera.MV_GIGE_TRANSMISSION_TYPE.MV_GIGE_TRANSTYPE_MULTICAST;
stTransmissionType.nDestIp = (uint)nIp;
stTransmissionType.nDestPort = 8787;

nRet = device.MV_GIGE_SetTransmissionType_NET(ref stTransmissionType);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("MV_GIGE_SetTransmissionType fail! nRet [%x]\n", nRet);
    break;
}

// Start acquiring image
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Thread thr = new Thread(WorkThread);
thr.Start();

Console.WriteLine("Press enter to exit");
Console.ReadLine();

g_bExit = true;
Thread.Sleep(1000);

// en:Stop grabbing
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}
```

```
    }

    //Close device
    nRet = device.MV_CC_CloseDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Close device failed{0:x8}", nRet);
        break;
    }

    //:Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
        break;
    }
} while (false);

if (MyCamera.MV_OK != nRet)
{
    // Destroy device
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
```

B.19 Spatial Denoising

The sample code below shows how to denoise the image of a camera.

```
using System;
using System.Collections.Generic;
using MvCamCtrl.NET;
using System.Runtime.InteropServices;
using System.IO;

namespace SpatialDenoise
{
    class SpatialDenoise
    {
        public static MyCamera.cbOutputExdelegate ImageCallback;
```

```
public static MyCamera device = new MyCamera();
static bool g_IsNeedNoiseEstimate = true;
static IntPtr g_pNoiseProfile = IntPtr.Zero;
static IntPtr g_pDstData = IntPtr.Zero;
static uint g_nNoiseProfileSize = 0;
static uint g_nDstDataSize = 0;

static void ImageCallbackFunc(IntPtr pData, ref MyCamera.MV_FRAME_OUT_INFO_EX pFrameInfo, IntPtr pUser)
{
    int nRet = MyCamera.MV_OK;
    Console.WriteLine("Get one frame: Width[" + Convert.ToString(pFrameInfo.nWidth) + "], Height[" +
Convert.ToString(pFrameInfo.nHeight)
        + "], FrameNum[" + Convert.ToString(pFrameInfo.nFrameNum) + "]);
    // Judge whether to estimate noise.
    if (true == g_IsNeedNoiseEstimate)
    {
        // Estimate noise.
        MyCamera.MV_CC_NOISE_ESTIMATE_PARAM stEstimateParam = new
MyCamera.MV_CC_NOISE_ESTIMATE_PARAM();
        stEstimateParam.nWidth = pFrameInfo.nWidth;
        stEstimateParam.nHeight = pFrameInfo.nHeight;
        stEstimateParam.enPixelType = pFrameInfo.enPixelType;
        stEstimateParam.pSrcBuf = pData;
        stEstimateParam.nSrcBufLen = pFrameInfo.nFrameLen;

        // If you want to estimate the noise of the whole image, set nROINum to 0 and set pstROIRect to NULL.
        MyCamera.MV_CC_RECT_I stROIRect = new MyCamera.MV_CC_RECT_I();
        stROIRect.nX = 0;
        stROIRect.nY = 0;
        stROIRect.nWidth = pFrameInfo.nWidth;
        stROIRect.nHeight = pFrameInfo.nHeight;
        stEstimateParam.pstROIRect = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(MyCamera.MV_CC_RECT_I)));
        Marshal.StructureToPtr(stROIRect, stEstimateParam.pstROIRect, false);
        stEstimateParam.nROINum = 1;

        // Noise estimation parameters (Bayer format). Invalid for MONO8/RGB format.
        stEstimateParam.nNoiseThreshold = 1024;

        stEstimateParam.pNoiseProfile = IntPtr.Zero;
        nRet = device.MV_CC_NoiseEstimate_NET(ref stEstimateParam);
        if (MyCamera.MV_OK != nRet)
        {
            if (g_pNoiseProfile == IntPtr.Zero || g_nNoiseProfileSize < stEstimateParam.nNoiseProfileLen)
            {
                if (g_pNoiseProfile != IntPtr.Zero)
                {
                    Marshal.FreeHGlobal(g_pNoiseProfile);
                    g_pNoiseProfile = IntPtr.Zero;
                    g_nNoiseProfileSize = 0;
                }

                g_pNoiseProfile = Marshal.AllocHGlobal((int)stEstimateParam.nNoiseProfileLen);
            }
        }
    }
}
```

```
        if (g_pNoiseProfile == IntPtr.Zero)
        {
            Console.WriteLine("malloc pNoiseProfile failed");
            return;
        }
        g_nNoiseProfileSize = stEstimateParam.nNoiseProfileLen;
    }

    stEstimateParam.pNoiseProfile = g_pNoiseProfile;
    stEstimateParam.nNoiseProfileSize = g_nNoiseProfileSize;
    nRet = device.MV_CC_NoiseEstimate_NET(ref stEstimateParam);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Noise estimate failed:{0:x8}", nRet);
        return;
    }
}

Marshal.FreeHGlobal(stEstimateParam.pstROIRect);

// Save noise reduction characteristic file to local disk.
byte[] EstimateData = new byte[stEstimateParam.nNoiseProfileLen];
Marshal.Copy(stEstimateParam.pNoiseProfile, EstimateData, 0, (int)stEstimateParam.nNoiseProfileLen);
FileStream pFile = null;
try
{
    pFile = new FileStream("./NoiseProfile.bin", FileMode.Create);
    pFile.Write(EstimateData, 0, EstimateData.Length);
}
catch
{
    Console.WriteLine("Saving failed");
}
finally
{
    pFile.Close();
}

g_IsNeedNoiseEstimate = false;
}

// Spatial noise reduction
if (g_pDstData == IntPtr.Zero || g_nDstDataSize < pFrameInfo.nFrameLen)
{
    if (g_pDstData != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(g_pDstData);
        g_pDstData = IntPtr.Zero;
        g_nDstDataSize = 0;
    }

    g_pDstData = Marshal.AllocHGlobal((int)pFrameInfo.nFrameLen);
}
```



```
if (g_pDstData == IntPtr.Zero)
{
    Console.WriteLine("malloc pDstData failed");
    return;
}
g_nDstDataSize = pFrameInfo.nFrameLen;
}

MyCamera.MV_CC_SPATIAL_DENOISE_PARAM stDisnoiseParam = new
MyCamera.MV_CC_SPATIAL_DENOISE_PARAM();
stDisnoiseParam.nWidth = pFrameInfo.nWidth;      // Image width
stDisnoiseParam.nHeight = pFrameInfo.nHeight;    // Image height
stDisnoiseParam.enPixelFormat = pFrameInfo.enPixelFormat; // Input pixel format
stDisnoiseParam.pSrcBuf = pData;                  // Input data buffer
stDisnoiseParam.nSrcBufLen = pFrameInfo.nFrameLen; // Size of input data
stDisnoiseParam.pDstBuf = g_pDstData;             // Output data buffer
stDisnoiseParam.nDstBufSize = g_nDstDataSize;      //Size of output buffer

stDisnoiseParam.pNoiseProfile = g_pNoiseProfile;
stDisnoiseParam.nNoiseProfileLen = g_nNoiseProfileSize;

// Spatial noise reduction parameters (Bayer format). Not available for Mono8/RGB format.
stDisnoiseParam.nBayerDenoiseStrength = 50;        //Denoising strength (0-100)
stDisnoiseParam.nBayerSharpenStrength = 16;        //Sharpening strength (0-32)
stDisnoiseParam.nBayerNoiseCorrect = 1024;         //Noise correction factor (0-1280)

// Spatial noise reduction parameters (Mono8/RGB format). Not available for Bayer format.
stDisnoiseParam.nNoiseCorrectLum = 1000;
stDisnoiseParam.nNoiseCorrectChrom = 500;
stDisnoiseParam.nStrengthLum = 50;
stDisnoiseParam.nStrengthChrom = 100;
stDisnoiseParam.nStrengthSharpen = 300;
nRet = device.MV_CC_SpatialDenoise_NET(ref stDisnoiseParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Spatial denoise failed:{0:x8}", nRet);
    return;
}

if (pFrameInfo.nFrameNum < 10)
{
    // Save image to file.
    MyCamera.MV_SAVE_IMG_TO_FILE_PARAM stSaveFileParam = new
MyCamera.MV_SAVE_IMG_TO_FILE_PARAM();
    stSaveFileParam.enImageType = MyCamera.MV_SAVE_IAMGE_TYPE.MV_Image_Bmp;
    stSaveFileParam.enPixelFormat = pFrameInfo.enPixelFormat;
    stSaveFileParam.nWidth = pFrameInfo.nWidth;
    stSaveFileParam.nHeight = pFrameInfo.nHeight;
    stSaveFileParam.nDataLen = pFrameInfo.nFrameLen;
    stSaveFileParam.pData = pData;
    stSaveFileParam.pImagePath = "BeforeImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + pFrameInfo.nFrameNum.ToString() + ".bmp";
```

```
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    return;
}

stSaveFileParam.pData = g_pDstData;
stSaveFileParam.pImagePath = "AfterImage_w" + stSaveFileParam.nWidth.ToString() + "_" +
stSaveFileParam.nHeight.ToString() + "_fn" + pFrameInfo.nFrameNum.ToString() + ".bmp";
nRet = device.MV_CC_SaveImageToFile_NET(ref stSaveFileParam);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("SaveImageToFile failed:{0:x8}", nRet);
    return;
}
}
}

static void Main(string[] args)
{
    int nRet = MyCamera.MV_OK;
    do
    {
        // Enumerate devices.
        MyCamera.MV_CC_DEVICE_INFO_LIST stDevList = new MyCamera.MV_CC_DEVICE_INFO_LIST();
        nRet = MyCamera.MV_CC_EnumDevices_NET(MyCamera.MV_GIGE_DEVICE | MyCamera.MV_USB_DEVICE,
ref stDevList);
        if (MyCamera.MV_OK != nRet)
        {
            Console.WriteLine("Enum device failed:{0:x8}", nRet);
            break;
        }
        Console.WriteLine("Enum device count : " + Convert.ToString(stDevList.nDeviceNum));
        if (0 == stDevList.nDeviceNum)
        {
            break;
        }

        MyCamera.MV_CC_DEVICE_INFO stDevInfo;                // General device information.

        // Print device information.
        for (Int32 i = 0; i < stDevList.nDeviceNum; i++)
        {
            stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[i],
typeof(MyCamera.MV_CC_DEVICE_INFO));

            if (MyCamera.MV_GIGE_DEVICE == stDevInfo.nTLayerType)
            {
                MyCamera.MV_GIGE_DEVICE_INFO stGigEDeviceInfo =
(MyCamera.MV_GIGE_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stGigEInfo,
typeof(MyCamera.MV_GIGE_DEVICE_INFO));
```

```
uint nIp1 = ((stGigEDeviceInfo.nCurrentIp & 0xff000000) >> 24);
uint nIp2 = ((stGigEDeviceInfo.nCurrentIp & 0x00ff0000) >> 16);
uint nIp3 = ((stGigEDeviceInfo.nCurrentIp & 0x0000ff00) >> 8);
uint nIp4 = (stGigEDeviceInfo.nCurrentIp & 0x000000ff);
Console.WriteLine("[device " + i.ToString() + "]:");
Console.WriteLine("DevIP:" + nIp1 + "." + nIp2 + "." + nIp3 + "." + nIp4);
Console.WriteLine("UserDefineName:" + stGigEDeviceInfo.chUserDefinedName + "\n");
}
else if (MyCamera.MV_USB_DEVICE == stDevInfo.nTLayerType)
{
    MyCamera.MV_USB3_DEVICE_INFO stUsb3DeviceInfo =
(MyCamera.MV_USB3_DEVICE_INFO)MyCamera.ByteToStruct(stDevInfo.SpecialInfo.stUsb3VInfo,
typeof(MyCamera.MV_USB3_DEVICE_INFO));
    Console.WriteLine("[device " + i.ToString() + "]:");
    Console.WriteLine("SerialNumber:" + stUsb3DeviceInfo.chSerialNumber);
    Console.WriteLine("UserDefineName:" + stUsb3DeviceInfo.chUserDefinedName + "\n");
}
}

Int32 nDevIndex = 0;
Console.Write("Please input index  (0 -- {0:d})  :", stDevList.nDeviceNum - 1);
try
{
    nDevIndex = Convert.ToInt32(Console.ReadLine());
}
catch
{
    Console.Write("Invalid Input!\n");
    break;
}

if (nDevIndex > stDevList.nDeviceNum - 1 || nDevIndex < 0)
{
    Console.Write("Input Error!\n");
    break;
}
stDevInfo = (MyCamera.MV_CC_DEVICE_INFO)Marshal.PtrToStructure(stDevList.pDeviceInfo[nDevIndex],
typeof(MyCamera.MV_CC_DEVICE_INFO));

// Create device.
nRet = device.MV_CC_CreateDevice_NET(ref stDevInfo);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Create device failed:{0:x8}", nRet);
    break;
}

// Open the device.
nRet = device.MV_CC_OpenDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Open device failed:{0:x8}", nRet);
}
```

```
        break;
    }

    // Get the optimal package size (GigE camera only).
    if (stDevInfo.nTLayerType == MyCamera.MV_GIGE_DEVICE)
    {
        int nPacketSize = device.MV_CC_GetOptimalPacketSize_NET();
        if (nPacketSize > 0)
        {
            nRet = device.MV_CC_SetIntValue_NET("GevSCSPPacketSize", (uint)nPacketSize);
            if (nRet != MyCamera.MV_OK)
            {
                Console.WriteLine("Warning: Set Packet Size failed {0:x8}", nRet);
            }
        }
        else
        {
            Console.WriteLine("Warning: Get Packet Size failed {0:x8}", nPacketSize);
        }
    }

    // Set trigger mode to Off.
    nRet = device.MV_CC_SetEnumValue_NET("TriggerMode", 0);
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Set TriggerMode failed!");
        break;
    }

    // Judge whether it can be imported locally.
    if (true == File.Exists("./NoiseProfile.bin"))
    {
        FileStream fs = new FileStream("./NoiseProfile.bin", FileMode.Open);
        byte[] data = new byte[fs.Length];

        if (g_pNoiseProfile == IntPtr.Zero || g_nNoiseProfileSize < fs.Length)
        {
            if (g_pNoiseProfile != IntPtr.Zero)
            {
                Marshal.FreeHGlobal(g_pNoiseProfile);
                g_pNoiseProfile = IntPtr.Zero;
                g_nNoiseProfileSize = 0;
            }

            g_pNoiseProfile = Marshal.AllocHGlobal((int)fs.Length);
            if (g_pNoiseProfile == IntPtr.Zero)
            {
                Console.WriteLine("malloc pNoiseProfile failed");
                break;
            }
            g_nNoiseProfileSize = (uint)fs.Length;
        }
    }
}
```

```
fs.Read(data, 0, data.Length);
fs.Close();

Marshal.Copy(data, 0, g_pNoiseProfile, (Int32)g_nNoiseProfileSize);

g_IsNeedNoiseEstimate = false;
}

// Register image callback.
ImageCallback = new MyCamera.cbOutputExdelegate(ImageCallbackFunc);
nRet = device.MV_CC_RegisterImageCallBackEx_NET(ImageCallback, IntPtr.Zero);
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Register image callback failed!");
    break;
}

// Start image acquisition.
nRet = device.MV_CC_StartGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Start grabbing failed:{0:x8}", nRet);
    break;
}

Console.WriteLine("Press enter to exit");
Console.ReadLine();

// Stop image acquisition.
nRet = device.MV_CC_StopGrabbing_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Stop grabbing failed{0:x8}", nRet);
    break;
}

// Close the device.
nRet = device.MV_CC_CloseDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Close device failed{0:x8}", nRet);
    break;
}

// Destroy the device.
nRet = device.MV_CC_DestroyDevice_NET();
if (MyCamera.MV_OK != nRet)
{
    Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    break;
}
```

```
} while (false);

if (g_pNoiseProfile != IntPtr.Zero)
{
    Marshal.FreeHGlobal(g_pNoiseProfile);
    g_pNoiseProfile = IntPtr.Zero;
    g_nNoiseProfileSize = 0;
}

if (g_pDstData != IntPtr.Zero)
{
    Marshal.FreeHGlobal(g_pDstData);
    g_pDstData = IntPtr.Zero;
    g_nDstDataSize = 0;
}

if (MyCamera.MV_OK != nRet)
{
    // Destroy the device.
    nRet = device.MV_CC_DestroyDevice_NET();
    if (MyCamera.MV_OK != nRet)
    {
        Console.WriteLine("Destroy device failed:{0:x8}", nRet);
    }
}

Console.WriteLine("Press enter to exit");
Console.ReadKey();
}
}
}
```

