China Daheng Group, Inc. Beijing Image Vision Technology Branch

C++ SDK
Programmer's Guide

Version：V1.2.2

Date：2020-10-12

DAHENG IMAGING 大恒图像

Notice

Web: http://www.daheng-imaging.com

Sales Email: isales@daheng-imaging.com

Sales Tel: +86 10 8282 8878

Support Email: isupport@daheng-imaging.com

Contents

1. Introduction

The **GxIAPICPPEx.dll** library is an encapsulated general and unified application programming interface which based on the object-oriented method; it is suit for MERCURY, MARS series of DAHENG IMAGING camera.

The main modules and functions as shown in Table 1-1:

Main module	Function
IGXFactory	Initialize the interface library, enumerate devices, open devices.
IGXDevice	Device objects, via this object as an entrance to control the objects features, image acquisition, get the cameras' events.
IGXStream	Stream object, get from **IGXDevice**, responsible for image acquisition and etc.
IGXFeatureControl	Features control object, you can get the feature control objects from **IGXDevice** and **IGXStream**, and to handle various features control.
IImageData	The callback acquisition method returns image structure, including the image acquisition/output results: Image buffer and image information, and comes with image format conversion/image effect enhance function.
GXBitmap	This object exists in the form of source code in the Sample program, responsible for the image display and storage functions, as shown in the Sample program.

Table 1-1 Main modules and functions

- Supported programming environment

  Microsoft Visual Studio 2005

  Microsoft Visual Studio 2008

  Microsoft Visual Studio 2010

  Microsoft Visual Studio 2012

  Microsoft Visual Studio 2013

2. Programming Guide

2.1. Build Programming Environment

Take VS2008 platform for example, the header file and .lib file are needed during the compiling stage, you can refer the **Sample** program project configuration under the installation directory.

2.1.1. Selection of the Library

Since the **GxIAPICPP** library has compiler compatibility issues, the newly released **GxIAPICPP** library will be upgraded to **GxIAPICPPEx** after 2019. The old **GxIAPICPP** library is still retained (but no longer releases its Lib files), it will not affect the normal operation of your compiled program, but if you need to recompile or develop a new program, please use **GxIAPICPPEx** library for development, the configuration process is as follows.

2.1.2. Configuration Header File

Select the project which created by the user in the solution resource management window, then click the menu **project->properties**, pop-up the **Property page** window. Select **Configuration Properties->C/C++->General**, fill in the directory path address (based on the actual installation directory) of the **GalaxyIncludes.h** in the **Additional Include Directories**, and as shown in Figure 2-1.

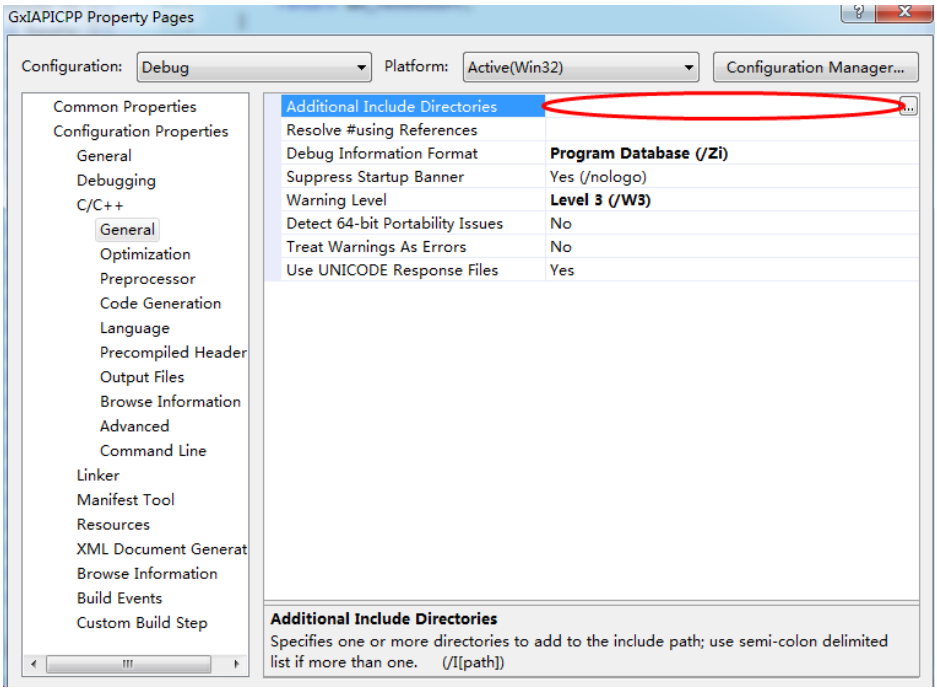

Figure 2-1 Configuration header file

2.1.3. Configuration Lib Files

Select **Configuration Properties->Linker->General**, fill in the directory path address (based on the actual installation directory) of the **GxIAPICPPEx.lib** in the **Additional Library Directories**, and as shown in Figure 2-2.

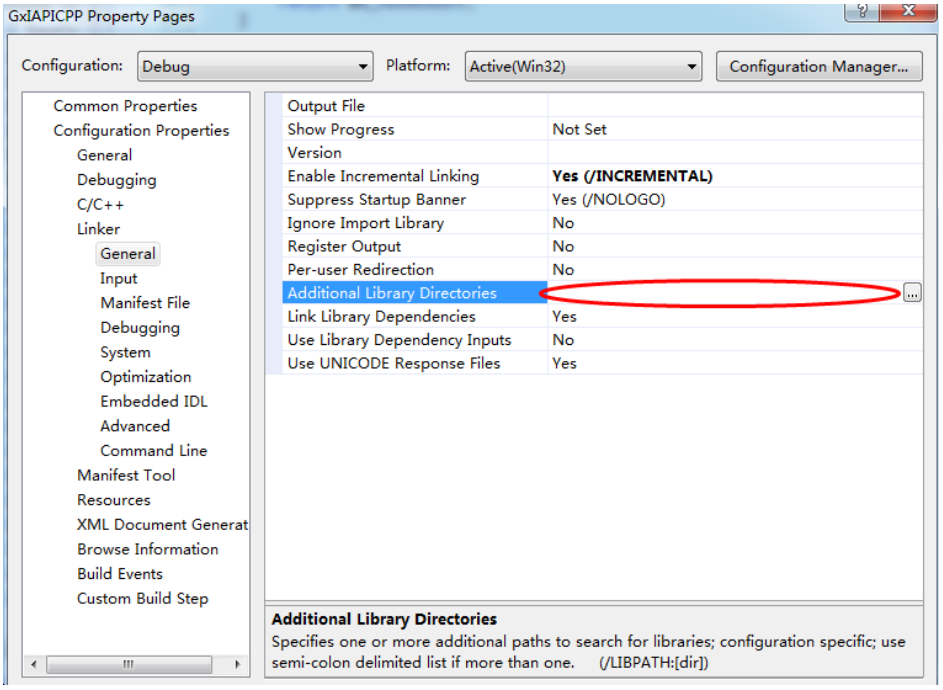

Figure 2-2 Configuration lib file path

Select **Configuration Properties->Linker->Input**, fill in the **GxIAPICPPEx.lib** in the **Additional Dependencies**, and as shown in Figure 2-3.

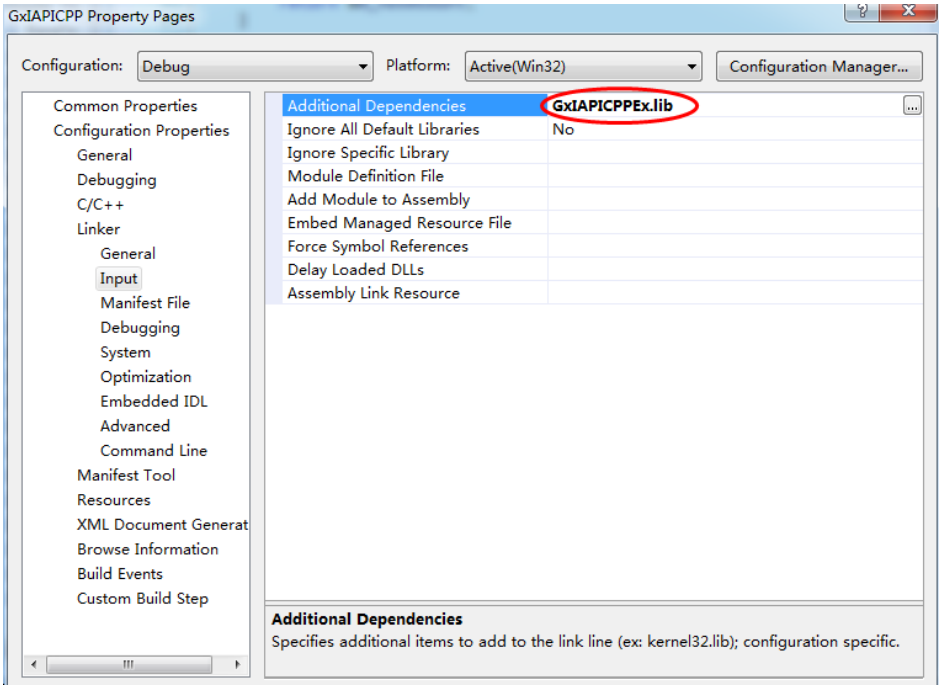

Figure 2-3 Configurations the lib name

2.2. Precautions for Debugging the GigE Vision Cameras

When the Windows users use Visual studio development platform debugging the GigE Vision cameras, they may encounter device offline, which is caused by the heartbeat timeout. The application must send heartbeat packet to the camera in defined intervals. If the camera doesn't receive this heartbeat packet, it will consider the current connection as broken and won't accept any commands from the application.

When the user running the application, it will send heartbeat packet normally and keep connect with the camera, but in debug mode, when the application runs to the breakpoint, the debugger will suspend all the thread, including the sending heartbeat packet thread, so in debug mode, the application will not send heartbeat packet to the camera.

Users can add heartbeat timeout time to resolve it.

Users can change the heartbeat timeout by two ways:

The first one is: open the device in the program and add the following code:

Only in debugging mode can use this code snippet, don't use it in the release program.

```
//Open device, here, open the device through Serial Number (SN), the SN will be
//subject to the actual device. Users can also open the device by other ways.
#ifdef_DEBUG
CGXDevicePointer objDevicePtr =IGXFactory::GetInstance().OpenDeviceBySN(
                                        "RN0001007012",GX_ACCESS_EXCLUSIVE);

//Get remote device feature controller.
CGXFeatureControlPointer objFeatureControl =
                                    objDevicePtr->GetRemoteFeatureControl();

//Set heartbeat timeout for 5 minutes.
objFeatureControl->GetIntFeature("GevHeartbeatTimeout")->SetValue(300000);
#endif
```

The other way is: add the environment variable "MER_HEARTBEAT_TIMEOUT" in the system and assign a value greater than zero, then open the device with application program, the device's heartbeat timeout will automatically become the environment variable's value. Only to add the environment variable in the development system, and they will work in the application no matter in debug mode or release mode.

Note: if the user set the heartbeat timeout very long, when close the program it will not call the close device interface to close the device, this will lead to the device cannot to be reset in the heartbeat time, thus when the user try again to open the device, it will fail. This problem can be solved by resetting or reconnecting the device. There are two methods to reset or reconnect the device:

1) Select "Reset Device" or "Reconnect Device" button directly in the IP Configurator.

2) Reset or reconnect the device by the interface 'GXGigEResetDevice'.

Reset device: It is equivalent to powering off and powering up the device, and the programs in the camera are completely reloaded.

Reconnect device: It is equivalent to the software interface close the device. After doing this, the user can reopen the device.

2.3. Initialization and Uninitialization

The **GxIAPICPPEx** library must be initialized before use. Before using any other interfaces, the **IGXFactory::GetInstance ().Init ()** must be called first.

```
//Call this function to initialize before using any other interfaces.
IGXFactory::GetInstance().Init();
```

The **IGXFactory.GetInstance().Uninit()** must be called to release all the resources of the **GxIAPICPPEx** application before the process exits.

```
//After closing the device,can't call any other interface library.
```

```
IGXFactory::GetInstance().Uninit();
```

2.4. Error Handling

All the **GxIAPICPPEx** API functions throw exceptions when internal errors are detected. The exception type is **CGalaxyException**, which inherited from the **runtime_error**. Users can call the function **CGalaxyException::GetErrorCode ()** to get the error codes, and the error code type can be referred to GX_STATUS_LIST; users can also access the exception class properties by **CGalaxyException::What ()** to get detailed error description information string.

A typical error handling code might look like:

```
try
{
    IGXFactory::GetInstance().Init();
}
catch (CGalaxyException&e)
{
    cout <<"error code: "<< e.GetErrorCode() <<endl;
    cout <<"error message: "<< e.what() <<endl;
}
```

2.5. Enumerating Devices

Users can call function **IGXFactory::GetInstance().UpdateDeviceList** to enumerate all the current available devices, and get a device information list, the list type is **GxIAPICPP::gxdeviceinfo_vector**, enumeration device code snippet as follows:

```
GxIAPICPP::gxdeviceinfo_vector vectorDeviceInfo;
IGXFactory::GetInstance().UpdateDeviceList(1000, vectorDeviceInfo);
for (uint32_t_t i = 0; i < vectorDeviceInfo.size(); i++)
{
    cout << vectorDeviceInfo[i].GetVendorName() <<endl;
    cout << vectorDeviceInfo[i].GetModelName() <<endl;
    //More devices information can refer IGXDeviceInfo interface.
}
```

Note: Except the enumerated interface above, **GxIAPICPPEx** also provides another enumerated interface **IGXFactory::GetInstance().UpdateAllDeviceList**. For non-GigE Vision cameras, they are the same in the function, but for GigE Vision cameras, they are different in the internal enumeration mechanism.

UpdateAllDeviceList：Enumerate in the whole network.

UpdateDeviceList：Enumerate in the same subnet.

2.6. Configure Camera IP Address

Users can call the following two different interface to set the IP address of Camera.

IGXFactory::GetInstance().GigEIpConfiguration

IGXFactory::GetInstance().GigEForceIp

Use the function **GigEIpConfiguration** can set the camera static(permanent) IP address, and provide four settings modes: direct write static IP address, use DHCP server to assign IP addresses, use LLA (Link-Local Address) to configure camera IP and use default mode to set IP. When you choose the default mode,

the internal camera will enable the other three configure modes, but the static IP as a preferred way to configure camera IP address.

Use the function **GigEForceIp** can perform ForceIp operations on the camera. ForceIp means that setting the IP address is valid only for this use. When the camera power off and power on will restore the original IP address.

Two ways to configure the IP address of the camera sample code is as follows:

```
//Sample MAC address，actual camera MAC address can be obtained by enumeration.
GxIAPICPP::gxstring  strMAC= "00-21-49-00-00-00";
GxIAPICPP::gxstring  strIPAddress= "192.168.10.10";
GxIAPICPP::gxstring  strSubnetMask= "255.255.255.0";
GxIAPICPP::gxstring  strDefaultGateway= "192.168.10.2";
GxIAPICPP::gxstring  strUserID= "Daheng Imaging";
GX_IP_CONFIGURE_MODE emIpConfigureMode  = IP_CONFIGURE_STATIC_IP;

IGXFactory::GetInstance().GigEIpConfiguration(strMAC, emIpConfigureMode,
                                              strIpAddress, strSubnetMask,
                                              strDefaultGateway, strUserID);
```

```
//Sample MAC address，actual camera MAC address can be obtained by enumeration.
GxIAPICPP::gxstring strMAC= "00-21-49-00-00-00";
GxIAPICPP::gxstring strIPAddress= "192.168.10.10";
GxIAPICPP::gxstring strSubnetMask= "255.255.255.0";
GxIAPICPP::gxstring strDefaultGateway= "192.168.10.2";

//ForceIp
IGXFactory::GetInstance().GigEForceIp (strMAC, strIpAddress, strSubnetMask,
                                       strDefaultGateway);
```

Note:

1) Before calling these two interfaces, you must first enumeration,and when you do this, the camera can’t be opened.
2) The maximum length of the user-defined name (UserID) allowed is 16 characters.

2.7. Open and Close Devices

Users can call the following four different interface to open the device:

1) IGXFactory::GetInstance().OpenDeviceBySN
2) IGXFactory::GetInstance().OpenDeviceByUserID
3) IGXFactory::GetInstance().OpenDeviceByMAC
4) IGXFactory::GetInstance().OpenDeviceByIP

SN: device serial number.

UserID: user defined name (if the device does not support **UserID**, assign this item as a null string).

MAC: the MAC address of the device (for non-GigE Vision cameras, assign this item as a null string).

IP: the device’s IP address (for non-GigE Vision cameras, assign this item as a null string).

It is strongly recommended the user to call the enumeration device method and update the library internal device list before open the device, otherwise may open the device failure. The sample code snippet as follows:

```
GxIAPICPP::gxdeviceinfo_vector vectorDeviceInfo;
IGXFactory::GetInstance().UpdateDeviceList(1000, vectorDeviceInfo);

if (vectorDeviceInfo.size()> 0)
{
    //Open the first device in the list.
    CGXDevicePointer objDevicePtr;
    GxIAPICPP::gxstring strSN     = vectorDeviceInfo[0].GetSN();
    GxIAPICPP::gxstring strUserID = vectorDeviceInfo[0].GetUserID();
    GxIAPICPP::gxstring strMAC    = vectorDeviceInfo[0].GetMAC();
    GxIAPICPP::gxstring strIP     = vectorDeviceInfo[0].GetIP();

    //Users can also assign the device information of the opened device
    //directly. The information shown in the following code is indicative only,
    //the actual information may differ.
    //GxIAPICPP::gxstring strSN     = "GA0140100002";
    //GxIAPICPP::gxstring strUserID = "MyUserName";
    //GxIAPICPP::gxstring strMAC    = "A1-0B-32-7C-6F-81";
    //GxIAPICPP::gxstring strIP     = "192.168.0.100";
    objDevicePtr = IGXFactory::GetInstance().OpenDeviceBySN(strSN,
                                                    GX_ACCESS_EXCLUSIVE);
    //objDevicePtr = IGXFactory::GetInstance().OpenDeviceByUserID(strUserID,
                                                    GX_ACCESS_EXCLUSIVE);
    //objDevicePtr = IGXFactory::GetInstance().OpenDeviceByMAC(strMAC,
                                                    GX_ACCESS_EXCLUSIVE);
    //objDevicePtr = IGXFactory::GetInstance().OpenDeviceByIP(strIP,
                                                    GX_ACCESS_EXCLUSIVE);
}
```

Note: For GigE Vision devices, users can open the device directly without enumeration, because the network mechanism allows to establishment connection directly by IP address.

Users can call the **IGXDevice::Close** to close device, release all the device resources. Code snippet as follows:

```
//After closing the device, user can not to call the function IDevice and all
//the IFeatureControl&IStream functions of the device.
objDevicePtr->Close();
```

Note : After closing the device, all the resources about the device will be released, including the **CGXFeatureControlPointer** object and **CGXStreamPointer** object which get from the device object, so when the device is closed, these objects' interface are not allowed to be called again.

2.8. Features Control

2.8.1. Controller Type

There are three feature controllers, see as follows:

1) IGXFeatureControl IGXDevice::GetRemoteFeatureControl //Including the device's main information, such as: width, height, exposure, gain etc. General users mainly use this feature controller.

2) IGXFeatureControl IGXDevice::GetFeatureControl //Including some local features, the device functions of different type may differ.

3) IGXFeatureControl IGXStream::GetFeatureControl //Stream object feature controller, a feature access controller about acquisition control and data statistics.

Running the **Galaxyview** demo to open the device, you can see the parameters in the right properties controller tree, there are three parts: the upper one is the properties which return from **IGXDevice::GetRemoteFeatureControl (Figure 2-4)**, and the middle one is the properties which return from **IGXDevice::GetFeatureControl (Figure 2-4)**. The lower one is the return properties from **IGXStream::GetFeatureControl (Figure 2-4)**.

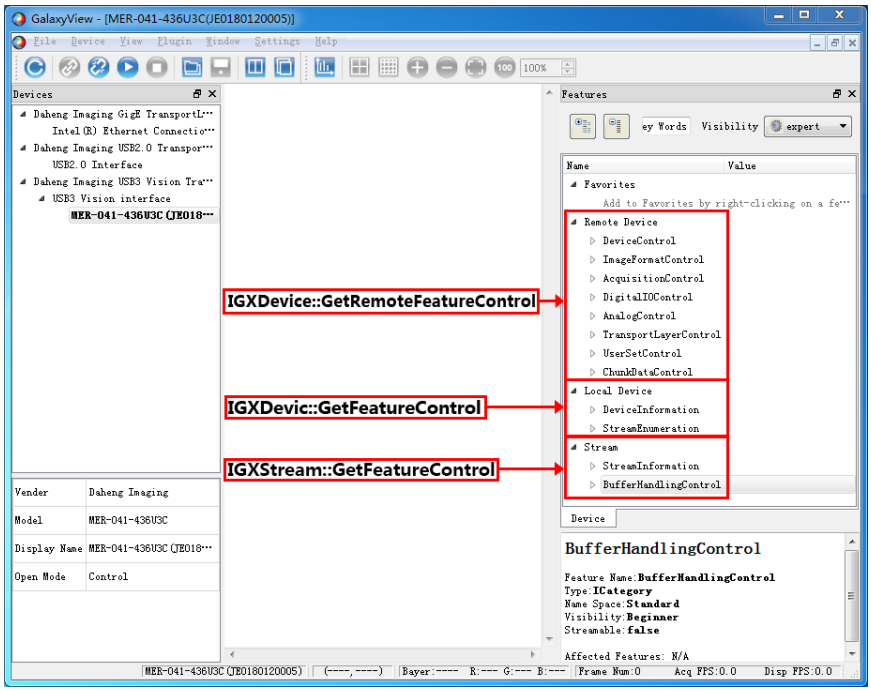

Figure 2-4 Far-end, local and stream features

2.8.2. Data Type

There are seven data types, see as follows:

1) CIntFeaturePointerPointerIGXFeatureControl::GetIntFeature //Integer

2) CFloatFeaturePointerIGXFeatureControl::GetFloatFeature //Float

3) CBoolFeaturePointerIGXFeatureControl::GetBoolFeature //Boolean

4) CEnumFeaturePointerIGXFeatureControl::GetEnumFeature //Enumerated

5) CStringFeaturePointerIGXFeatureControl::GetStringFeature //String

6) CCommandFeaturePointerIGXFeatureControl::GetCommandFeature //Command

7) CRegisterFeaturePointerIGXFeatureControl::GetRegisterFeature //Register

2.8.3. Access Type

There are three access types, see as follows:

1) bool IGXFeatureControl::IsImplemented /Whether the current property controller supports this feature

2) bool IGXFeatureControl::IsReadable // Whether this function is readable

3) bool IGXFeatureControl::IsWritable // Whether this function is writable

2.8.4. How to Get the Camera's Parameters

Get from the function interface: **IGXFeatureControl.GetFeatureNameList**, returns the function name string which the current features control supported.

Get from the **GalaxyView** demo: open the device by **GalaxyView** demo, view the right property control tree, the English name of every control node, as shown in the Figure 2-5.

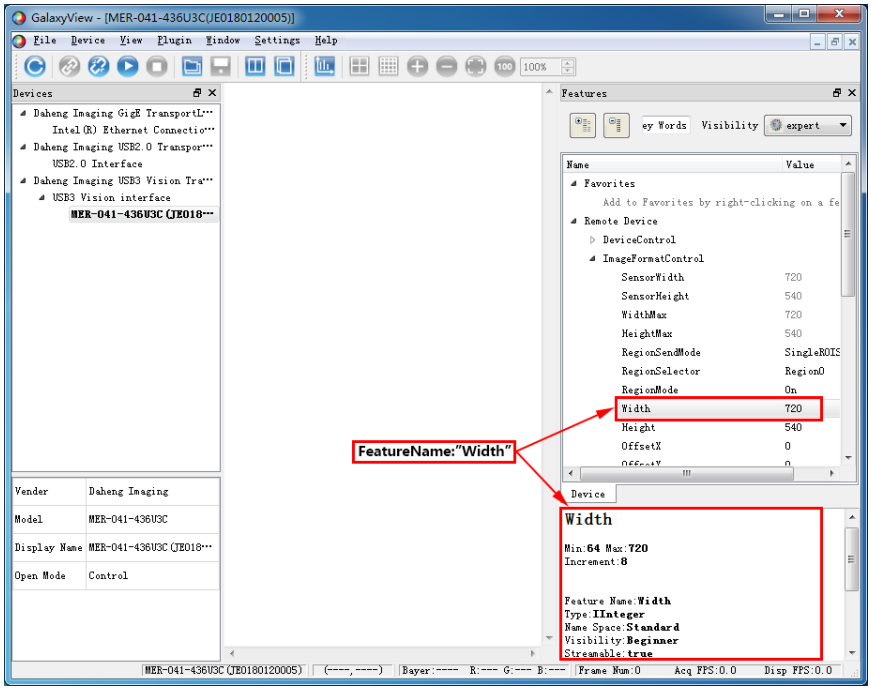

Figure 2-5 Get cameras parameters

As shown in the Figure 2-5, when the mouse selects the **"width"**, the function name, maximum, minimum, step etc. will be shown in the **Function Properties Description Area**.

Note: The function name string is case sensitive.

2.8.5. How to Get the Sample Code for Feature Read & Write

The GalaxyView.exe provides sample codes for feature read & write. In the demo program menu bar: View -> Feature Document, you can open the window, as shown in the Figure 2-6.

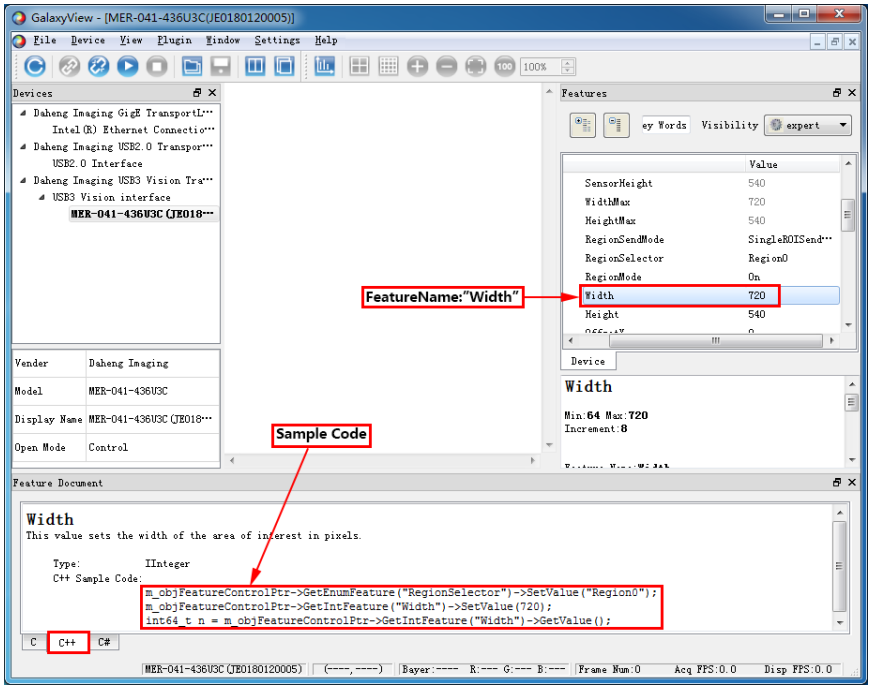

Figure 2-6

In the view of feature document, three development languages are available. Switch to C++ Tab to get the code for reading and writing user-specified feature in C++ language. The code can be directly copied to the user's development project.

2.8.6. Different Types of Feature Sample Code

2.8.6.1. Read & Write Access Control

All functions are controlled by function string. Check to see if the feature is currently available, and then to read or write.

```
CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();
//Get width node.
bool bIsImplemented = objFeatureControlPtr->IsImplemented("Width");
if (bIsImplemented)
{
    //Is readable?
    bool bIsReadable = objFeatureControlPtr->IsReadable("Width");
    //Is writable?
    bool bIsWritable = objFeatureControlPtr->IsWritable("Width");
}
```

2.8.6.2. Integer Control

According to the data type, the camera's function is divided into seven categories, integer is one of them, including width, height and other integer control parameters, users can use the integer controller **CIntFeaturePointer** to read the current value, inquire the maximum, minimum and step.

```
CGXFeatureControlPointer objFeatureControlPtr=
                                    objDevicePtr->GetRemoteFeatureControl();
CIntFeaturePointer objIntPtr = objFeatureControlPtr->GetIntFeature("Width");
int64_t nMax = objIntPtr->GetMax();              //Get the maximum value.
int64_t nMin = objIntPtr->GetMin();              //Get the minimum value.
int64_t nInc = objIntPtr->GetInc();              //Get the increment value.
```

```
int64_t nValue = objIntPtr->GetValue();          //Get the current value.
objIntPtr->SetValue(nValue);                     //Set the current value.
```

2.8.6.3. Float Control

To access the float data by **CFloatFeaturePointer**, you can read and write the current value, check the Maximum, Minimum, whether to support increment, increment value and unit.

```
CGXFeatureControlPointer objFeatureControlPtr=
                                    objDevicePtr->GetRemoteFeatureControl();
CFloatFeaturePointer objFloatPtr = objFeatureControlPtr->GetFloatFeature(
                                                                    "Gain");
double dMax  = objFloatPtr->GetMax();            //Get the maximum value.
double dMin  = objFloatPtr->GetMin();            //Get the minimum value.
bool bHasInc = objFloatPtr->HasInc();
if (bHasInc)
{
    double dInc = objFloatPtr->GetInc();         //Get the increment value.
}
GxIAPICPP::gxstring strUnit = objFloatPtr->GetUnit(); //Get the unit.
double dValue = objFloatPtr->GetValue();         //Get the current value.
objFloatPtr->SetValue(dValue);                   //Set the current value.
```

2.8.6.4. Boolean Control

Boolean is relatively simple, only **True** and **False**.

```
CGXFeatureControlPointer objFeatureControlPtr =
                                     objDevicePtr->GetRemoteFeatureControl();
CBoolFeaturePointer objBoolPtr = objFeatureControlPtr->GetBoolFeature(
                                                        "ChunkModeActive");
bool bValue = objBoolPtr->GetValue();           //Get the current value.
objBoolPtr->SetValue(bValue);                   //Set the current value.
```

2.8.6.5. Enumeration Control

Enumerated can check the options, and read/write interfaces which the enumerated function supported.

```
CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();
CEnumFeaturePointer objEnumPtr = objFeatureControlPtr->GetEnumFeature(
                                                        "AcquisitionMode");
GxIAPICPP::gxstring_vector vectorEnumEntry;
vectorEnumEntry = objEnumPtr->GetEnumEntryList();//Get the enum items list.
for(int i=0; i<vectorEnumEntry.size(); i++)
{
    cout << vectorEnumEntry[i] << endl;          //Print all the enum items.
}
GxIAPICPP::gxstring strValue = objEnumPtr->GetValue();//Get the current item.
objEnumPtr->SetValue(strValue);                      //Set the current item.
```

2.8.6.6. String Control

String provides the function to read/write the current value, check the maximum string length supported before write.

```
CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();
CStringFeaturePointer objStringPtr = objFeatureControlPtr->GetStringFeature(
                                                          "DeviceUserID");
```

```
//Get the current value.
GxIAPICPP::gxstring strValue = objStringPtr->GetValue();
//Get the maximum length can be written to.
int64_t nMaxLength = objStringPtr->GetStringMaxLength();
//Write the current value.
objStringPtr->SetValue(strValue);
```

2.8.6.7. Command Control

The command type is simplest, only one executes method.

```
CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();
CCommandFeaturePointer objCommandPtr =
                objFeatureControlPtr->GetCommandFeature("AcquisitionStart");
objCommand->Execute();                  //Execute the command.
```

2.8.7. Precautions

After obtaining the features controller **CGXFeatureControlPointer** from the device object, it is valid until the device closed. Once the device is closed, the features controller is invalid; you cannot call any interfaces of the features controller.

2.9. Acquisition Control and Image Processing

All the interfaces and control related to Acquisition are in the **CGXStreamPointer** object. How to get and open the stream object please see the following code:

```
uint32_t_t nStreamNum = objDevicePtr->GetStreamCount();
if (nStreamNum > 0)
{
    CGXStreamPointer objStreamPtr = objDevicePtr->OpenStream(0);

    //Stream object control or acquisition.

    //Closing it when no longer to use the stream object.
    objStreamPtr->Close();
}
```

There two acquisition ways: callback acquisition and get a single image.

2.9.1. Get a Single Image

After opening the stream object acquisition and sending start acquisition command to the device, you can call the **GetImage** to get a single image, please see the following code:

```
CGXStreamPointer objStreamPtr = objDevicePtr->OpenStream(0);

//Improve the acquisition performance of the network camera (currently only
//the Gigabit network camera supports the optimal packet length).
GX_DEVICE_CLASS_LIST objDeviceClass =
                              objDevicePtr->GetDeviceInfo().GetDeviceClass();
if (GX_DEVICE_CLASS_GEV == objDeviceClass)
{
    //Determine whether the device supports the stream channel packet function.
    if (true == objFeatureControlPtr->IsImplemented("GevSCPSPacketSize"))
    {
      //Get the optimal packet length value of the current network environment.
       int nPacketSize = objStreamPtr->GetOptimalPacketSize();
```

```
        //Set the optimal packet length value to the stream channel packet
        //length of the current device.

objFeatureControlPtr->GetIntFeature("GevSCPSPacketSize")->SetValue(
                                                                nPacketSize);
    }
}

//Open stream channel acquisition.
objStreamPtr->StartGrab();

//Send start acquisition command to the device.
CGXFeatureControlPointer objFeatureControlPtr =
                                        objDevicePtr->GetRemoteFeatureControl();
objFeatureControlPtr->GetCommandFeature("AcquisitionStart")->Execute();

//Get a single image.
CImageDataPointer objImageDataPtr;

//Set timeout is 500ms, users can set the value by themselves.
objImageDataPtr = objStreamPtr->GetImage(500);
if (objImageDataPtr->GetStatus() == GX_FRAME_STATUS_SUCCESS)
{
    //If the acquisition is success and the image is a full frame, then the
    //user can process the image.
}

//Stop acquisition.
objFeatureControlPtr->GetCommandFeature("AcquisitionStop")->Execute();
objStreamPtr->StopGrab();

//Close stream channel.
objStreamPtr->Close();
```

Note: Users must call the **StartGrab** to start the stream channel acquisition first, and then send start acquisition command to the device; otherwise the start acquisition command will invalid. **When using a high-resolution camera for high-speed data acquisition, recommend you to use callback acquisition mode, because the buffer copy inside the GetImage will affect the transmission performance.**

2.9.2. Acquisition with Callback Functions

Users need to inherit the virtual base class **ICaptureEventHandler** to implement the callback handler class, see the following code:

```
class CSampleCaptureEventHandler : public ICaptureEventHandler
 {
    public:
        void DoOnImageCaptured(CImageDataPointer& objImageDataPointer, void* pUserParam)
            {
                if (objImageDataPointer->GetStatus() == GX_FRAME_STATUS_SUCCESS)
                 {
                    //If the image acquisition is a full frame, then you can read
                   //the image width,height,data format and etc.
                    uint64_t nWidth  = objImageDataPointer->GetWidth();
                    uint64_t nHeight = objImageDataPointer->GetHeight();
```

```
                    GX_PIXEL_FORMAT_ENTRY emPixelFormat =
                                        objImageDataPointer->GetPixelFormat();

                    //More images information please refer the IImageData
                    //interface define.
                }
            }
 };
```

Users can register callback function to get images, see the following code:

```
CGXStreamPointer objStreamPtr = objDevicePtr->OpenStream(0);

CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();

//Improve the acquisition performance of the network camera
//(currently only the Gigabit network camera supports the optimal packet length).
GX_DEVICE_CLASS_LIST objDeviceClass =
                                objDevicePtr->GetDeviceInfo().GetDeviceClass();
if (GX_DEVICE_CLASS_GEV == objDeviceClass)
{
    //Determine whether the device supports the stream channel packet function.
    if (true == objFeatureControlPtr->IsImplemented("GevSCPSPacketSize"))
    {
       //Get the optimal packet length value of the current network environment.
        int nPacketSize = objStreamPtr->GetOptimalPacketSize();

        //Set the optimal packet length value to the stream channel packet
       //length of the current device.
        objFeatureControlPtr->GetIntFeature("GevSCPSPacketSize")->SetValue(
                                                                    nPacketSize);
    }
}

//Register the acquisition callback function. Note: the first parameter is a
//user's private parameter, user can set any type of object, also can set it null.
//The user's private parameter used within the callback function for restore,
//if not use the private parameter,you can set it null.
ICaptureEventHandler* pCaptureEventHandler = new CSampleCaptureEventHandler();
objStreamPtr->RegisterCaptureCallback(pCaptureEventHandler,NULL);

//Start stream channel acquisition.
objStreamPtr->StartGrab();

//Send start acquisition command to the device.
objFeatureControlPtr->GetCommandFeature("AcquisitionStart")->Execute();

//Callback acquisition process, please refer the callback function.

//Stop acquisition, unregister acquisition callback function.
objFeatureControlPtr->GetCommandFeature("AcquisitionStop")->Execute();
objStreamPtr->StopGrab();
objStreamPtr->UnregisterCaptureCallback();
delete pCaptureEventHandler;
```

```
pCaptureEventHandler = NULL;

//Close stream channel.
objStreamPtr->Close();
```

Note: Users must call the function **StartGrab** to start stream channel acquisition first, and then send start acquisition command to the device; otherwise the start acquisition command will invalid.

2.9.3. Set the Number of Acquisition Buffer

After opening the stream, users can call SetAcqusitionBufferNumber to set the number of acquisition buffer, see the following code:

```
CGXStreamPointer objStreamPtr = objDevicePtr->OpenStream(0);

CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();
//Improve the acquisition performance of the network camera.
//(currently only the Gigabit network camera supports the optimal packet length).
GX_DEVICE_CLASS_LIST objDeviceClass =
                                objDevicePtr->GetDeviceInfo().GetDeviceClass();
if (GX_DEVICE_CLASS_GEV == objDeviceClass)
{
    //Determine whether the device supports the stream channel packet function.
    if (true == objFeatureControlPtr->IsImplemented("GevSCPSPacketSize"))
    {
        //Get the optimal packet length value of the current network environment.
        int nPacketSize = objStreamPtr->GetOptimalPacketSize();

       //Set the optimal packet length value to the stream channel packet
       //length of the current device.
       objFeatureControlPtr->GetIntFeature("GevSCPSPacketSize")->SetValue(
                                                                nPacketSize);
    }
}

//Set the number of acquisition buffer.
objStreamPtr->SetAcqusitionBufferNumber(10);

//Start stream channel acquisition.
objStreamPtr->StartGrab();

// Send start acquisition command to the device.
objFeatureControlPtr->GetCommandFeature("AcquisitionStart")->Execute();

// ……

//Stop acquisition.
objFeatureControlPtr->GetCommandFeature("AcquisitionStop")->Execute();
objStreamPtr->StopGrab();

//Close stream channel.
objStreamPtr->Close();
```

Note:

1) This interface is an optional interface. When multiple cameras are acquisiting at the same time, if the frame rate of individual cameras is 0, you can call this interface to adjust the number of acquisition buffers of all cameras to ensure that all cameras have buffers for acquisition.

2) The buffer number must be set before calling StartGrab to start the stream channel acquisition, otherwise the setting is invalid.

2.9.4. Image Processing

2.9.4.1. Image Format Conversion

Function description: Specified for an 8 bit valid data. Assume that the raw data is non 8-bit, no matter mono or color, you can call the function **CovertToRaw8** to specify for an 8-bit data, and the return value type is **void***. The specified memory size is **Width * Height**.

Code Sample：

```
void* pRaw8Buffer = NULL;
//Assume the raw data is mono8 image.
pRaw8Buffer = objImageDataPtr->ConvertToRaw8(GX_BIT_0_7);
//Assume the raw data is Mono12 image.
pRaw8Buffer = objImageDataPtr->ConvertToRaw8(GX_BIT_4_11);
//Assume the raw data is BayerRG8 image.
pRaw8Buffer = objImageDataPtr->ConvertToRaw8(GX_BIT_0_7);
//Assume the raw data is BayerRG12 image.
pRaw8Buffer = objImageDataPtr->ConvertToRaw8(GX_BIT_4_11);
```

Function description: Bayer format convert to RGB24 format, no matter the current image output is mono or color, 8-bit or non 8-bit, you can call the function **ConvertToRGB24** to complete data interpolation processing, and the return value **void*** is RGB24 format. The memory size is Width*Height*3.

Code Sample：

```
void* pRGB24Buffer = NULL;
//Assume the raw data is mono8 image.
pRGB24Buffer = objImageDataPtr->ConvertToRGB24(GX_BIT_0_7,
                                               GX_RAW2RGB_NEIGHBOUR, true);
//Assume the raw data is Mono12 image.
pRGB24Buffer = objImageDataPtr->ConvertToRGB24(GX_BIT_4_11,
                                               GX_RAW2RGB_NEIGHBOUR, true);
//Assume the raw data is BayerRG8 image.
pRGB24Buffer = objImageDataPtr->ConvertToRGB24(GX_BIT_0_7,
                                               GX_RAW2RGB_NEIGHBOUR, true);
//Assume the raw data is BayerRG12 image.
pRGB24Buffer = objImageDataPtr->ConvertToRGB24(GX_BIT_4_11,
                                               GX_RAW2RGB_NEIGHBOUR, true);
```

2.9.4.2. Image Effect Enhanced

The interface library also provides the image effect enhanced interface in software side, users can do some image effect enhanced operate selectively, such as: defective-pixel correction, sharpening, contrast, brightness and etc. The image effect enhanced involves the functions **CImageProcessConfigPointer** and **CImageDataPointer**.

Code Sample：

```
//Use the device object to construct image processing configuration object.
CImageProcessConfigPointer objImageProcessConfigPtr =
                                    objDevicePtr->CreateImageProcessConfig();

//The parameter objIBseDataPtr can be introduced by the acquisition callback
//function,and also can be acquired by the function GetImage.
void* pRGB24Processed = NULL;

//The return result is RGB24 format data which is handled by image effect enhanced.
pRGB24Processed = objIBaseData->ImageProcess(objImageProcessConfigPtr);
```

For different data type cameras, the support of effect enhanced from interface library is also different. Details as the Table 2-1:

Function	Corresponding interface	Raw data			
		Mono8	Mono non8	Bayer8	Bayer non8
Set valid bit	SetValidBit	Yes	Yes	Yes	Yes
Defective Pixel correction switch	EnableDefectivePixelCorrect	Yes	Yes	Yes	Yes
Sharpen switch	EnableSharpen	Yes	Yes	Yes	Yes
Sharpen factor	SetSharpenParam	Yes	Yes	Yes	Yes
Contrast ratio	SetContrastParam	Yes	Yes	Yes	Yes
Gamma	SetGammaParam	Yes	Yes	Yes	Yes
Brightness	SetLightnessParam	Yes	Yes	Yes	Yes
Noise reduction switch	EnableDenoise	No	No	Yes	Yes
Saturation	SetSaturationParam	No	No	Yes	Yes
RGB24 interpolation type	SetConvertType	No	No	Yes	Yes
RGB24 interpolation flip switch	EnableConvertFlip	No	No	Yes	Yes
Color correction switch	EnableColorCorrection	No	No	Yes	Yes

Table 2-1 Image effect enhanced comparison table

The parameter range of Image effect enhance and the detail explanation as Table 2-2:

Functions	Default value	Range	Remarks
Set valid bit	GX_BIT_0_7	Refer GX_VALID_BIT_LIST	● For 8 bit data , only one to select : GX_BIT_0_7 ● For 10 bit data, you can select GX_BIT_0_7、GX_BIT_1_8、GX_BIT_2_9, recommend GX_BIT_2_9 ● For 12bit data, you can select GX_BIT_0_7、GX_BIT_1_8、GX_BIT_2_9、GX_BIT_3_10、GX_BIT_4_11 ,recommend

			GX_BIT_4_11
Defective Pixel correction switch	false	True, false	true: switch on Defective Pixel correction; false: switch off Defective Pixel correction
Sharpen switch	false	True, false	true: switch on sharpen; false: switch off sharpen;
Sharpen factor	0.1	[0.1, 5]	Increase sharpening effect gradually
Contrast ratio	0	[-50, 100]	0: the contrast ratio does not change > 0: increasing contrast ratio <0: decrease contrast ratio
Gamma	1.0	[0.1, 10]	-
Brightness	0	[-150, 150]	0 : the brightness does not change > 0: increasing brightness < 0: decrease brightness
Noise reduction switch	false	True, false	true：switch on noise reduction; false: switch off noise reduction;
Saturation	64	[0, 128]	64: saturation does not change >64: increasing saturation < 64: decrease saturation 128: double the current saturation 0: mono image
RGB24 interpolation type	GX_RAW2RGB_NEIGHBOUR	Refer GX_BAYER_CONVERT_TYPE_LIST	-
RGB24 interpolation flip switch	false	True, false	true: flip; false: non-flip

Table 2-2 Image effect enhanced parameters range and description

The advanced user can fine-tune the parameter after getting **CImageProcessConfigPointer** object, see as follows:

```
//Through the device object to construct image processing configuration object.
CImageProcessConfigPointer objImageProcessConfigPtr =
                                    objDevicePtr->CreateImageProcessConfig();

//Initialization the default configuration parameter when constructing the
//objImageProcessConfigPtr object, the user can choose to fine-tune the
//configuration parameter:
//Select significance bit 0-7.
objImageProcessConfigPtr->SetValidBit(GX_BIT_0_7);
//Enable Defective Pixel correction.
objImageProcessConfigPtr->EnableDefectivePixelCorrect(true);
//Enable sharpen
objImageProcessConfigPtr->EnableSharpen(true);
//Set sharpen factor is 1.
objImageProcessConfigPtr->SetSharpenParam(1);
//Set contrast ratio control parameter.
objImageProcessConfigPtr->SetContrastParam(0);
```

```
//Set Gamma factor.
objImageProcessConfigPtr->SetGammaParam(1);
//Set brightness control parameter.
objImageProcessConfigPtr->SetLightnessParam(0);
//Enable noise reduction switch, not support mono camera.
objImageProcessConfigPtr->EnableDenoise(true);
//Set saturation control factor, not support mono camera.
objImageProcessConfigPtr->SetSaturationParam(0);
//Set interpolation algorithm, not support mono camera.
objImageProcessConfigPtr->SetConvertType(GX_RAW2RGB_NEIGHBOUR);
//Enable interpolation flip, not support mono camera.
objImageProcessConfigPtr->EnableConvertFlip(true);


//User can also choose to restore the optimum default parameter configuration.
objImageProcessConfigPtr->Reset();

//The parameter objImageDataPtr can be introduced by the acquisition callback
//function, and also can be acquired by the function GetImage.
void*pRGB24Processed = NULL;
//The return result is RGB24 format data which is handled by image effect enhanced.
pRGB24Processed = objImageDataPtr->ImageProcess(objImageProcessConfigPtr);
```

2.9.4.3. The Result of Image Effect Enhanced

According to the cameras' type and usage scenario is different, the image effect enhanced configuration parameters are also different, but there are still a few recommend functions can meet the needs of most users. Take MER-200-20GC GigE Vision camera for example, the image which the camera output after image format conversion to RGB24 bit, the result as Figure 2-7 without any effect enhance processing:

Figure 2-7 Neighborhood interpolation RGB image (color correction: off; saturation: 64; Gamma: 1; contrast: 0)

On the basis of Figure 2-7, switch on the color correction function; see the result as Figure 2-8:

Figure 2-8 Switch on color correction based on figure 2-8 (color correction: on; saturation: 64; Gamma: 1; Contrast: 0)

On the basis of Figure 2-8, increase the saturation to 80; see the result as Figure 2-9:

Figure 2-9 Set saturation to 80(color correction: on; saturation: 80; Gamma: 1; Contrast: 0)

On the basis of Figure 2-9, adjust the Gamma to 1.98, make the image more close to what the users see in the actual scene (Figure 2-10), the value 1.98 is just for an example, for different cameras the Gamma value will differ, for the USB2.0 and USB3.0 cameras, users can read the gamma by the read/write interface, but for the GigE Vision camera, the user should adjust the gamma by himself, there is not a interface to be called directly.

Figure 2-10 Set the Gamma is 1.98(color correction: on; saturation: 80; Gamma: 1.98; Contrast: 0)

On the basis of Figure 2-10, set the contrast is 40, see the result as Figure 2-11:

Figure 2-11 Set the contrast is 40(color correction: on; saturation: 80; Gamma: 1.98; Contrast: 40)

2.9.5. Stream Object Features Control

We have mentioned that the stream layer has its own features control in the chapter '**features control**', users can get it by calling the interface **IGXStream::GetFeatureControl**. The control features of the stream layer include the acquisition control and statistical information. Take the Gigabit Ethernet stream

layer statistical for example, such as to acquire the current acquisition and statistical information in the process of acquisition.

```
//objGXStream is an CGXStreamPointer object which get from the interface
//IGXDevice::OpenStream.
CGXFeatureControlPointer objStreamFeatureControlPtr =
                                          objGXStream->GetFeatureControl();

//Acquire the acquisition and statistical information.
//The lost frame count which caused by the buffer lack.
objStreamFeatureControlPtr->GetIntFeature("StreamLostFrameCount")->GetValue();

//The count of incomplete frame.
objStreamFeatureControlPtr->GetIntFeature(
                                  "StreamIncompleteFrameCount")->GetValue();

//The delivered packet count.
objStreamFeatureControlPtr->GetIntFeature(
                                  "StreamDeliveredPacketCount")->GetValue();

//The resend packet count.
objStreamFeatureControlPtr->GetIntFeature(
                                     "StreamResendPacketCount")->GetValue();

//Set acquisition configuration parameter.
//Set the block timeout to 200ms.
objStreamFeatureControlPtr->GetIntFeature("BlockTimeout")->SetValue(200);
```

The "StreamBufferHandlingMode" in the stream object features can set the processing mode of the Buffer. Three Buffer processing modes are available:

1) OldestFirst: The default value. The image buffer follows the first-in-first-out principle. After all the buffers are filled, the new image data will be discarded until the user completes the processing of the buffer that has filled the image data. The typical application is to receive each frame of images acquired by the camera without losing frames. In order to achieve no frame loss, the speed of image data transmission and processing need to be as fast as possible (at least less than the frame period).
2) OldestFirstOverwrite: Follow the first-in-first-out principle. The difference from the OldestFirst mode is that when all the buffers are filled, the SDK will automatically discard one frame of image buffer with the oldest timestamp to receive new image data. The typical application is that it does not require receiving each frame of images acquired by the camera, and the image data transmission and processing speed is slow.
3) NewestOnly: In this mode, the user always receives the latest image received by the SDK. Each time the SDK receives a new frame of image data, it will automatically discard the image with the old timestamp. Therefore, when the user's image processing is not timely or the speed is slow, frame loss will occur. In the main applications, the real-time requirements of image acquisition and display are high, and it is not required to receive each frame of images acquired by the camera. However, depending on the camera's frame rate, memory cache, transmission speed, and user applications, there may be a delay between the latest image received by the SDK and the latest image exposed by the camera.

Note: The above code snippet just shown the stream layer control function of the GigE Vision cameras, for the other cameras, please use the **GalaxyView** demo to open the device, or call the function **IGXFeatureControl::GetFeatureNameList** to get the function list.

2.9.6. Precautions

After obtaining the stream object from the device object, the stream object is valid, until the device is closed. Once the device is closed, the stream object will invalid; you cannot call any interfaces of the stream object.

2.10. Get Devices Events

DAHENG IMAGING GigE Vision cameras can send event messages. For example, when a sensor exposure has finished, the camera can send an end-of-exposure event to the PC. The event can be received by the PC before the image data for the finished exposure has been completely transferred. In this section will illustrate how to obtain events and event data.

2.10.1. Selection Events

Feature name string	Function type	Description	Options (subject to the current device)
EventSelector	enumeration	Event source select	Maybe include the following options: ExposureEnd [Exposure end.] BlockDiscard [Block discard.] EventOverrun [Event queue overflow.] FrameStartOvertrigger [Trigger signal overflow.] BlockNotEmpty [Image buffer is not empty.] InternalError [Internal error]

2.10.2. Enabled Events

Feature name string	Function type	Description	Options (subject to the current device)
EventNotification	enumeration	Event enable	include the following options: Off [off] On [on]

2.10.3. Registered Events Message Callback Function

The function interface which the register event used is **IGXFeatureControl.RegisterFeatureCallback**, the first parameter is the function code of the event which to be registered, the event function code to be selected is as follows:

Feature name string	Function type	Description
EventExposureEnd	integer	exposure end event ID
EventBlockDiscard	integer	data block discard event ID
EventOverrun	integer	event queue overflow ID
EventFrameStartOvertrigger	integer	trigger signal is shield event ID
EventBlockNotEmpty	integer	the frame buffer is not empty event ID
EventInternalError	Integer	internal error event ID

2.10.4. Get Events Data Information

User can call the function **CGXFeatureControlPointer** to read the current event information within the event callback function, the section 2.9.3 declared 5 event types, the event information of the 5 events types can be getting by the following control codes:

String for Register event	Related event data string	Function type	Description
EventExposureEnd	EventExposureEndTimestamp	integer	exposure end event timestamp
	EventExposureEndFrameID	integer	exposure end event ID
EventBlockDiscard	EventBlockDiscardTimestamp	integer	data block discard event timestamp
EventOverrun	EventOverrunTimestamp	integer	event queue overflow timestamp
EventFrameStartOvertrigger	EventFrameStartOvertriggerTimestamp	integer	trigger signal is shield event timestamp
EventBlockNotEmpty	EventBlockNotEmptyTimestamp	integer	the frame buffer is not empty event timestamp
EventInternalError	EventInternalErrorTimestamp	integer	internal error event time stamp

2.10.5. Sample Code

Take the getting exposure end event for example, the code snippet as follows.

Users need to inherit the virtual base class **IFeatureEventHandler** to implement the callback handler class, see the following code:

```
class CSampleFeatureEventHandler : public IFeatureEventHandler
{
    public:
        void DoOnFeatureEvent(constGxIAPICPP::gxstring& strFeatureName,
                              void* pUserParam)
        {
            cout <<"EventExposureEnd!"<<endl;
            //The pUserParam is introduced during the user register callback
            //function, here restore it to get event data.
            CGXFeatureControlPointer* pObjFeatureControlPtr =
                                      (CGXFeatureControlPointer*)pUserParam;
            //Get exposure end event timestamp.
             (*pObjFeatureControlPtr)->GetIntFeature(
                                "EventExposureEndTimestamp")->GetValue();
             //Get exposure end event frameID.
             (*pObjFeatureControlPtr)->GetIntFeature(
                                   "EventExposureEndFrameID")->GetValue();
        }
};

//The objDevicePtr is the CGXDevicePointer device object, the device has been opened.
//If the device event's features is on the far-end device feature controller,
//get the far-end device feature controller first.
CGXFeatureControlPointer objFeatureControlPtr =
                                    objDevicePtr->GetRemoteFeatureControl();

//Select the event source.
objFeatureControlPtr->GetEnumFeature("EventSelector")->SetValue(
                                                          "ExposureEnd");
```

```
//Enable the event.
objFeatureControlPtr->GetEnumFeature("EventNotification")->SetValue("On");

//Register event callback function, note that the third parameter is an user
//private parameter, the user can introduce any pointer.
//Here we will show how to introduce the feature controller pointer, and then
//we will use the object to get exposuere end event data.
//The private paremeter can be restored for the user's use within the callback
//function, if the user does not use the private parameter,the parameter can
//be set to null.
GX_FEATURE_CALLBACK_HANDLE hEventHandle = NULL;
IFeatureEventHandler* pFeatureEventHandler = new CSampleFeatureEventHandler();
hEventHandle = objFeatureControlPtr->RegisterFeatureCallback("EventExposureEnd",
                                pFeatureEventHandler, &objFeatureControlPtr);

//Start stream channel acquisition.
objStreamPtr->StartGrab();

//Send start acquisition command to the device.
objFeatureControlPtr->GetCommandFeature("AcquisitionStart")->Execute();

//Send start acquisition command and the camera start exposuring and outputing
//images, when the exposure is end, an exposure end event will generate, then
//will activate the callback OnFeatureCallback interface.

//Receive the exposure end event, please refer to the
//OnFeatureCallbackfunction.

//Send stop acquisition command to the device.
objFeatureControlPtr->GetCommandFeature("AcquisitionStop")->Execute();
objStreamPtr->StopGrab();

//Unregister event.
objFeatureControlPtr->UnregisterFeatureCallback(objEventHandle);
delete pFeatureEventHandler;
pFeatureEventHandler = NULL;
```

Note: Users must send the start acquisition command first, then the camera can output exposure end event. So the user must call the**IGXStream::StartGrab** first, and then get the **AcquisitionStart** command node by the features controller and to execute the command **Execute**.

2.11. Get Offline Event

Only GigE Vision camera provides device offline event mechanism; users can get the device offline event by callback function, and register the event after open the device.

Steps of get the offline events:

1) Register the offline events.
2) Activate the callback function after the device offline.
3) Unregister the offline events.

User must inherit virtual base class **IDeviceOfflineEventHandler** callback function, see as follows:

```
class CSampleDeviceOfflineEventHandler : public IDeviceOfflineEventHandler
{
```

```
    public:
        void DoOnDeviceOfflineEvent(void* pUserParam)
        {
            cout <<" Offline!"<<endl;
        }
};
```

Register offline events code snippet as follows:

```
GX_DEVICE_OFFLINE_CALLBACK_HANDLE hDeviceOffline = NULL;
//ObjDevice is the CGXDevicePointer device object, the device has been opened.
//The second parameter is user private parameter, user can restore it within
//the callback function.If not use it, set it to null.
IDeviceOfflineEventHandler* pDeviceOfflineEventHandler =
                                        new CSampleDeviceOfflineEventHandler();
hDeviceOffline = objDevicePtr->RegisterDeviceOfflineCallback(
                                            pDeviceOfflineEventHandler,NULL);
```

Unregister events before close the device:

```
objDevicePtr->UnregisterDeviceOfflineCallback(hDeviceOffline);
delete pDeviceOfflineEventHandler;
pDeviceOfflineEventHandler = NULL;
```

Note: Not allowed to execute close the device operation within the offline callback function.

2.12. Import and Export Cameras' Configuration Parameters

Open device by running the **GalaxyView** Demo, expanding the menu **File**, you can see the menu **import device configuration** and **export device configuration**, see as Figure 2-12:

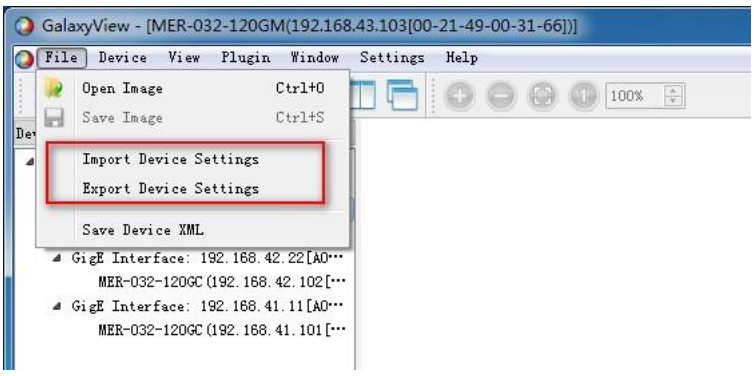

Figure 2-12 Import/export device configuration

Select **Export Device Configuration** menu, a new text file will create in the local disk to save the current parameter of the device.

Select **Import Device Configuration** menu, select the local file and import it.

There is also having corresponding user interface to be called in the interface library.

Import/export configuration files, sample code as follows:

```
//Open device, here we open the device by the serial number, the serial number
//is subject to the actual device.Users can open the device by other ways.
CGXDevicePointerobjDevicePtr = objIGXFactory->OpenDeviceBySN("RN0001007012",
                                                    GX_ACCESS_EXCLUSIVE);

//Open the first stream object of the device.
```

```
CGXStreamPointerobjStreamPtr = objDevicePtr->OpenStream(0);

//Export configuration files, users can free to take the file name, but the
//export catalog must be existed first.
objDevicePtr->ExportConfigFile("D:\\Config.txt");

//Import configuration files.
objDevicePtr->ImportConfigFile("D:\\Config.txt");
```

Note: Before executing the import/export configuration file, you should open the device first, and open the first stream object of the device.

3. Smart Pointer Object Type Definition

The control objects which the C++ interface library opens to the users are mostly encapsulated in the form of smart pointer, for more details you can see the following definition of the smart pointer type in the GXSmartPtr.h head file.

Type redefinition
typedef GXSmartPtr<IGXDevice> CGXDevicePointer
typedef GXSmartPtr<IGXStream> CGXStreamPointer
typedef GXSmartPtr<IImageProcessConfig> CImageProcessConfigPointer
typedef GXSmartPtr<IImageData> CImageDataPointer
typedef GXSmartPtr<IGXFeatureControl> CGXFeatureControlPointer
typedef GXSmartPtr<IIntFeature> CIntFeaturePointer
typedef GXSmartPtr<IFloatFeature> CFloatFeaturePointer
typedef GXSmartPtr<IEnumFeature> CEnumFeaturePointer
typedef GXSmartPtr<IBoolFeature> CBoolFeaturePointer
typedef GXSmartPtr<IStringFeature> CStringFeaturePointer
typedef GXSmartPtr<ICommandFeature> CCommandFeaturePointer
typedef GXSmartPtr<IRegisterFeature> CRegisterFeaturePointer

To avoid users getting the pointer inside the library directly, the smart pointer is encapsulated to smart pointer object and opens to the users. Users can use these redefined smart pointer objects as pointers. These smart pointer objects have the same function as the encapsulated smart pointers. For example, users can use the 'ObjDevicePointer' which is the object of CGXDevicePointer's directly, ObjDevicePointer->OpenStream(0), these interface definition can be find in 'IGXDevice' head file statement.

Smart pointer objects do not need to be destroyed deliberately, and they use the internal reference counting mechanism to destroy the resources when the reference count is 0.

4. Data Type Definition

4.1. GX_STATUS_LIST

Definition	Value	Explanation
GX_STATUS_SUCCESS	0	Success.
GX_STATUS_ERROR	-1	Error unexpected.
GX_STATUS_NOT_FOUND_TL	-2	Cannot find the TL library.
GX_STATUS_NOT_FOUND_DEVICE	-3	Cannot find devices.
GX_STATUS_OFFLINE	-4	The current device status is offline.
GX_STATUS_INVALID_PARAMETER	-5	Invalid parameter.
GX_STATUS_INVALID_HANDLE	-6	Invalid handle.
GX_STATUS_INVALID_CALL	-7	Invalid interface calling.
GX_STATUS_INVALID_ACCESS	-8	Current function cannot be accessed or the device access mode is incorrect.
GX_STATUS_NEED_MORE_BUFFER	-9	User's buffer is not enough.
GX_STATUS_ERROR_TYPE	-10	Feature type error, such as using IntFeature to handle with float function.
GX_STATUS_OUT_OF_RANGE	-11	Input value is out of the range.
GX_STATUS_NOT_IMPLEMENTED	-12	Unsupported function.
GX_STATUS_NOT_INIT_API	-13	Not call initialization interface.
GX_STATUS_TIMEOUT	-14	Timeout error.

4.2. GX_DEVICE_CLASS_LIST

Definition	Value	Explanation
GX_DEVICE_CLASS_UNKNOWN	0	The class of the Device is unknown.
GX_DEVICE_CLASS_USB2	1	USB2.0 devices.
GX_DEVICE_CLASS_GEV	2	GigE devices.
GX_DEVICE_CLASS_U3V	3	USB3.0 devices.

4.3. GX_ACCESS_STATUS

Definition	Value	Explanation
GX_ACCESS_STATUS_UNKNOWN	0	The status of the current device is unknown.
GX_ACCESS_STATUS_READWRITE	1	The status of the current device is readable/writable.
GX_ACCESS_STATUS_READONLY	2	The status of the current device is read-only.

GX_ACCESS_STATUS_NOACCESS	3	The status of the current device is not readable / writable.

4.4. GX_ACCESS_MODE

Definition	Value	Explanation
GX_ACCESS_READONLY	2	The access mode is read-only mode.
GX_ACCESS_CONTROL	3	The access mode is control mode.
GX_ACCESS_EXCLUSIVE	4	The access mode is exclusive mode.

4.5. GX_PIXEL_FORMAT_ENTRY

Definition	Value	Explanation
GX_PIXEL_FORMAT_UNDEFINED	0x00000000	-
GX_PIXEL_FORMAT_MONO8	0x01080001	Monochrome 8-bit.
GX_PIXEL_FORMAT_MONO8_SIGNED	0x01080002	Monochrome 8-bit signed.
GX_PIXEL_FORMAT_MONO10	0x01100003	Monochrome 10-bit unpacked.
GX_PIXEL_FORMAT_MONO12	0x01100005	Monochrome 12-bit unpacked.
GX_PIXEL_FORMAT_MONO14	0x01100025	Monochrome 14-bit unpacked.
GX_PIXEL_FORMAT_MONO16	0x01100007	Monochrome 16-bit.
GX_PIXEL_FORMAT_BAYER_GR8	0x01080008	Bayer Green-Red 8-bit.
GX_PIXEL_FORMAT_BAYER_RG8	0x01080009	Bayer Red-Green 8-bit.
GX_PIXEL_FORMAT_BAYER_GB8	0x0108000A	Bayer Green-Blue 8-bit.
GX_PIXEL_FORMAT_BAYER_BG8	0x0108000B	Bayer Blue-Green 8-bit.
GX_PIXEL_FORMAT_BAYER_GR10	0x0110000C	Bayer Green-Red 10-bit.
GX_PIXEL_FORMAT_BAYER_RG10	0x0110000D	Bayer Red-Green 10-bit.
GX_PIXEL_FORMAT_BAYER_GB10	0x0110000E	Bayer Green-Blue 10-bit.
GX_PIXEL_FORMAT_BAYER_BG10	0x0110000F	Bayer Blue-Green 10-bit.
GX_PIXEL_FORMAT_BAYER_GR12	0x01100010	Bayer Green-Red 12-bit.
GX_PIXEL_FORMAT_BAYER_RG12	0x01100011	Bayer Red-Green 12-bit.
GX_PIXEL_FORMAT_BAYER_GB12	0x01100012	Bayer Green-Blue 12-bit.
GX_PIXEL_FORMAT_BAYER_BG12	0x01100013	Bayer Blue-Green 12-bit.
GX_PIXEL_FORMAT_BAYER_GR16	0x0110002E	Bayer Green-Red 16-bit.
GX_PIXEL_FORMAT_BAYER_RG16	0x0110002F	Bayer Red-Green 16-bit.

GX_PIXEL_FORMAT_BAYER_GB16	0x01100030	Bayer Green-Blue 16-bit.
GX_PIXEL_FORMAT_BAYER_BG16	0x01100031	Bayer Blue-Green 16-bit.
GX_PIXEL_FORMAT_RGB8_PLANAR	0x02180021	Red-Green-Blue 8-bit planar.
GX_PIXEL_FORMAT_RGB10_PLANAR	0x02300022	Red-Green-Blue 10-bit unpacked.
GX_PIXEL_FORMAT_RGB12_PLANAR	0x02300023	Red-Green-Blue 12-bit unpacked.
GX_PIXEL_FORMAT_RGB16_PLANAR	0x02300024	Red-Green-Blue 16-bit planar.

4.6. GX_FRAME_STATUS_LIST

Definition	Value	Explanation
GX_FRAME_STATUS_SUCCESS	0	The frame status is successful.
GX_FRAME_STATUS_INCOMPLETE	-1	The frame status is incomplete.

4.7. GX_FEATURE_TYPE

Definition	Value	Explanation
GX_FEATURE_INT	0x10000000	Integer function.
GX_FEATURE_FLOAT	0x20000000	Float function.
GX_FEATURE_ENUM	0x30000000	Enumerated function.
GX_FEATURE_BOOL	0x40000000	Boolean function.
GX_FEATURE_STRING	0x50000000	String function.
GX_FEATURE_BUFFER	0x60000000	Register function.
GX_FEATURE_COMMAND	0x70000000	Command function.

4.8. GX_BAYER_CONVERT_TYPE_LIST

Definition	Value	Explanation
GX_RAW2RGB_NEIGHBOUR	0	Neighborhood average interpolation algorithm.
GX_RAW2RGB_ADAPTIVE	1	Edge adaptive interpolation algorithm.
GX_RAW2RGB_NEIGHBOUR3	2	Larger neighborhood average algorithm.

4.9. GX_VALLID_BIT_LIST

Definition	Value	Explanation
GX_BIT_0_7	0	bit 0~7.
GX_BIT_1_8	1	bit 1~8.
GX_BIT_2_9	2	bit 2~9.

GX_BIT_3_10	3	bit 3~10.
GX_BIT_4_11	4	bit 4~11.

4.10. GX_IP_CONFIGURE_MODE_LIST

Definition	Value	Explanation
GX_IP_CONFIGURE_LLA	4	Start LLA mode to configure IP address.
GX_IP_CONFIGURE_STATIC_IP	5	Use static IP mode to configure IP address.
GX_IP_CONFIGURE_DHCP	6	Use DHCP mode, use the DHCP server to configure IP address.
GX_IP_CONFIGURE_DEFAULT	7	Use the default mode to configure IP address.

4.11. GX_RESET_DEVICE_MODE

Definition	Value	Explanation
GX_MANUFACTURER_SPECIFIC_RECONNECT	1	Device reconnection. It is equivalent to the software interface close the device.
GX_MANUFACTURER_SPECIFIC_RESET	2	Device reset. It is equivalent to powering off and powering up the device.

4.12. COLOR_TRANSFORM_FACTOR

Definition	Default value	Value range	Explanation
float fGain00	1.0	-4.0 ~ 4.0	Red Channel Gain contribution to the red pixel.
float fGain01	0.0	-4.0 ~ 4.0	Green Channel Gain contribution to the red pixel.
float fGain02	0.0	-4.0 ~ 4.0	Blue Channel Gain contribution to the red pixel.
float fGain10	0.0	-4.0 ~ 4.0	Red Channel Gain contribution to the green pixel.
float fGain11	1.0	-4.0 ~ 4.0	Green Channel Gain contribution to the green pixel.
float fGain12	0.0	-4.0 ~ 4.0	Blue Channel Gain contribution to the green pixel.
float fGain20	0.0	-4.0 ~ 4.0	Red Channel Gain contribution to the blue pixel.
float fGain21	0.0	-4.0 ~ 4.0	Green Channel Gain contribution to the blue pixel.
float fGain22	1.0	-4.0 ~ 4.0	Blue Channel Gain contribution to the blue pixel.

5. Handle Type Definition

Definition	Explanation
GX_DEVICE_OFFLINE_CALLBACK_HANDLE	The handle of Offline event, which can be get by calling the interface ‘baIGXDevice::RegisterDeviceOfflineCallback’ .
GX_FEATURE_CALLBACK_HANDLE	The handle of Feature update event, which can be get by calling the interface: ‘IGXFeatureControl:: RegisterFeatureCallback’.

6. Callback Event Virtual Base Class

Type	Definition
Acquisition callback	For more details please see ICaptureEventHandler.
Device offline event	For more details please see IDeviceOfflineEventHandler.
Feature update event	For more details please see IFeatureEventHandler.

7. Module Interface Definition

7.1. IGXFactory

The 'IGXFactory' is responsible for initializing global resources, enumerating devices and opening devices.

Interface List:

GetInstance	Static function, and can be called globally. Return to the IGXFactory object instance.
Init	Initialize the library resources.
Uninit	Release library resources.
UpdateDeviceList	Enumerate the device (subnet enumeration for GigE Vision device).
UpdateAllDeviceList	Enumerate the device (whole network enumeration for GigE Vision device).
OpenDeviceByIP	Open the device by IP address.
OpenDeviceByMAC	Open the device by MAC address.
OpenDeviceBySN	Open the device by SN.
OpenDeviceByUserID	Open the device by user ID.
GigEIPConfiguration	Configure the camera static (perpetual) IP address.
GigEForceIP	Execute Force IP operation.
GigEResetDevice	Execute device reconnection or device reset operation.

7.1.1. GetInstance

Interface definition:

static IGXFactory& GetInstance()

Function description:

static function, and can be called globally. Return to the IGXFactory object instance.

7.1.2. Init

Interface definition:

void Init(void)

Function description:

initialize library resource.

7.1.3. Uninit

Interface definition:

void Uninit(void)

Function description:

release library resource.

7.1.4. UpdateDeviceList

Interface definition:

void UpdateDeviceList(uint32_t nTimeout, GxIAPICPP::gxdeviceinfo_vector& vectorDeviceInfo)

Function description:

enumerate devices (subnet enumeration for GigE Vision device).

Parameter1: timeout time, unit: ms.

Parameter2: the returned device information list.

7.1.5. UpdateAllDeviceList

Interface definition:

void UpdateAllDeviceList(uint32_t nTimeout, GxIAPICPP::gxdeviceinfo_vector& vectorDeviceInfo)

Function description:

enumerate devices (all subnet enumeration for GigE Vision device).

Parameter1: timeout, unit: ms.

Parameter2: the returned device information list.

7.1.6. OpenDeviceByIP/MAC/SN/UserID

Interface definition:

CGXDevicePointer OpenDeviceByIP(const GxIAPICPP::gxstring& strIP,
GX_ACCESS_MODE emAccessMode)

Function description:

open devices by IP address.

Parameter1: the device’s IP address.

Parameter2: the mode of open device.

Return value: CGXDevicePointer object instance.

7.1.7. GigEIPConfiguration

Interface definition:

void GigEIpConfiguration(const GxIAPICPP::gxstring& strDeviceMacAddress,
GX_IP_CONFIGURE_MODE emIpConfigMode,
const GxIAPICPP::gxstring& strIpAddress,
const GxIAPICPP::gxstring& strSubnetMask,
const GxIAPICPP::gxstring& strDefaultGateway,
const GxIAPICPP::gxstring& strUserID)

Function description:

configure the camera static (perpetual) IP address.

Parameter1: The MAC address of the device.

Parameter2: The IP configuration mode.

Parameter3: The IP address to be configured.

Parameter4: The subnet mask to be configured.

Parameter5: The default gateway to be configured.

Parameter6: The user-defined name to be configured.

Return value: NULL.

7.1.8. GigEForceIP

Interface definition:

```
void GXGigEForceIp(const GxIAPICPP::gxstring& pszDeviceMacAddress,
                   const GxIAPICPP::gxstring& strIpAddress,
                   const GxIAPICPP::gxstring& strSubnetMask,
                   const GxIAPICPP::gxstring& strDefaultGateway)
```

Function description:

execute Force IP operation.

Parameter1: The MAC address of the device.

Parameter2: The IP address value to be set.

Parameter3: The subnet mask to be set.

Parameter4: The default gateway to be set.

Return value: NULL.

7.1.9. GigEResetDevice

Interface definition:

```
void GigEResetDevice(const GxIAPICPP::gxstring& strDeviceMacAddress,
                     GX_RESET_DEVICE_MODE ui32FeatureInfo)
```

Function description:

execute device reconnection or device reset operation. See section 2.2 for details.

Device reconnection is usually used when debugging GigE cameras. The device has been opened, the program is abnormal, and then the device is reopened immediately, and an error is reported (because the heartbeat time is 5 minutes, the device is still open). In this case, you can use the device reconnection function to make the device unopened. Then open the device again and it will succeed.

Device reset is usually used when the camera status is abnormal. In this case, the device reconnection function does not work, and there is no condition to repower the device. Try to use the device reset function

to power off and power up the device. After the device is reset, you need to enumerate device and open device again.

Note：

1) Device reset takes about 1s, so you need to ensure that the enumeration interface is called after 1s.
2) If the device is acquiring normally, you cannot use the device reset and device reconnection function, otherwise the device will offline.

Parameter1: The MAC address of the device.

Parameter2: Reset device mode.

Return value: NULL.

7.2. IGXDeviceInfo

Device information storage unit, stores all the information of the device. You can get the information by the enumeration interface: IGXFactory::UpdateDeviceList or IGXFactory::UpdateAllDeviceList.

Interface list:

GetVendorName	Get the name of the manufacturer.
GetModelName	Get the model name of the device.
GetSN	Get the Series Number of the device.
GetDisplayName	Get the display name of the device.
GetDeviceID	Get the DeviceID, and the ID can be used to identify the device uniquely.
GetUserID	Get the user-defined name.
GetAccessStatus	Get the current addressable status of the device.
GetDeviceClass	Get the device class, such as USB2.0, USB3.0, and GigE.
GetMAC	Get the MAC address of the device, set the value to NULL for non-GigE device.
GetIP	Get the IP address of the device, set the value to NULL for non-GigE device.
GetSubnetMask	Get the subnet mask of the device, set the value to NULL for non-GigE device.
GetGateway	Get the default gateway of the device, set the value to NULL for non-GigE device.
GetNICMAC	Get the corresponding NIC MAC address of the device, set the value to NULL for non-GigE device.
GetNICIP	Get the corresponding NIC IP address of the device, set the value to NULL for non-GigE device.
GetNICSubnetMask	Get the corresponding NIC subnet mask of the device, set the value to NULL for non-GigE device.
GetNICGateway	Get the corresponding NIC default gateway of the device, set the value to NULL for non-GigE device.
GetNICDescription	Get the corresponding NIC description information of the device, set the value

to NULL for non-GigE device.

7.2.1. GetVendorName

Interface definition:

GxIAPICPP::gxstring GetVendorName() const

Function description:

Get the name of the manufacturer.

7.2.2. GetModelName

Interface definition:

GxIAPICPP::gxstring GetModelName() const

Function description:

Get the model name of the device.

7.2.3. GetSN

Interface definition:

GxIAPICPP::gxstring GetSN() const

Function description:

Get the Series Number of the device.

7.2.4. GetDisplayName

Interface definition:

GxIAPICPP::gxstring GetDisplayName() const

Function description:

Get the display name of the device.

7.2.5. GetDeviceID

Interface definition:

GxIAPICPP::gxstring GetDeviceID() const

Function description:

Get the DeviceID, and the ID can be used to identify the device uniquely.

7.2.6. GetUserID

Interface definition:

GxIAPICPP::gxstring GetUserID() const

Function description:

Get the user-defined name.

7.2.7. GetAccessStatus

Interface definition:

GX_ACCESS_STATUS GetAccessStatus() const

Function description:

Get the current addressable status of the device.

7.2.8. GetDeviceClass

Interface definition:

GX_DEVICE_CLASS_LIST GetDeviceClass() const

Function description:

Get the device class, such as USB2.0, USB3.0, and GigE.

7.2.9. GetMAC

Interface definition:

GxIAPICPP::gxstring GetMAC() const

Function description:

Get the MAC address of the device, set the value to NULL for non-GigE device.

7.2.10. GetIP

Interface definition:

GxIAPICPP::gxstring GetIP() const

Function description:

Getthe IP address of the device, set the value to NULL for non-GigE device.

7.2.11. GetSubnetMask

Interface definition:

GxIAPICPP::gxstring GetSubnetMask() const

Function description:

Get the subnet mask of the device, set the value to NULL for non-GigE device.

7.2.12. GetGateway

Interface definition:

GxIAPICPP::gxstring GetGateway() const

Function description:

Get the default gateway of the device, set the value to NULL for non-GigE device.

7.2.13. GetNICMAC

Interface definition:

GxIAPICPP::gxstring GetNICMAC() const

Function description:

Get the NIC MAC address of the device, set the value to NULL for non-GigE device.

7.2.14. GetNICIP

Interface definition:

GxIAPICPP::gxstring GetNICIP() const

Function description:

Get the NIC IP address of the device, set the value to NULL for non-GigE device.

7.2.15. GetNICSubnetMask

Interface definition:

GxIAPICPP::gxstring GetNICSubnetMask() const

Function description:

Get the NIC subnet mask of the device, set the value to NULL for non-GigE device.

7.2.16. GetNICGateway

Interface definition:

GxIAPICPP::gxstring GetNICGateway() const

Function description:

Get the NIC default gateway of the device, set the value to NULL for non-GigE device.

7.2.17. GetNICDescription

Interface definition:

GxIAPICPP::gxstring GetNICDescription() const

Function description:

Get the NIC description information of the device, set the value to NULL for non-GigE device.

7.3. IGXDevice

The device object, which is responsible for getting device information, feature controller (control channel), stream object (acquisition channel), and device offline events. You can get the information by calling the interface: IGXFactory::OpenDeviceByIP/MAC/SN/UserID.

Interface List:

GetDeviceInfo	Get the information object of the device.
GetStreamCount	Get the number of stream channel.

OpenStream	The user can open the stream by the stream channel number which is specified, and get the stream channel object.
GetFeatureControl	Get the local device layer feature controller, and control all the functions of the local device via this controller.
GetRemoteFeatureControl	Get the remote device layer feature controller, and control all the functions of the remote device via this controller.
GetEventNumInQueue	Get the event queue length of the device, represents how much cache data is in the current event queue.
FlushEvent	Flush the device event queue.
RegisterDeviceOfflineCallback	Register offline callback function.
UnregisterDeviceOfflineCallback	Unregister offline callback function.
CreateImageProcessConfig	Creating the image processing configure parameter object.
ExportConfigFile	Export the current configuration parameter to the text file.
ImportConfigFile	Import the parameters in the configuration file to the camera.
Close	Close the device.

7.3.1. GetDeviceInfo

Interface definition:

const CGXDeviceInfo& GetDeviceInfo()

Function description:

Get the information object of the device.

7.3.2. GetStreamCount

Interface definition:

uint32_t GetStreamCount()

Function description:

Get the number of stream channel.

7.3.3. OpenStream

Interface definition:

CGXStreamPointer OpenStream(uint32_t nStreamID)

Function description:

parameter 1: ‘nStreamID’ stream channel number, starts from 0. The user can open the stream by the stream channel number which is specified, and get the stream channel object.

7.3.4. GetFeatureControl

Interface definition:

CGXFeatureControlPointer GetFeatureControl()

Function description:

Get the local device layer feature controller, and control all the functions of the local device via this controller.

7.3.5. GetRemoteFeatureControl

Interface definition:

CGXFeatureControlPointer GetRemoteFeatureControl()

Function description:

Get the remote device layer feature controller, and control all the functions of the remote device via this controller.

7.3.6. GetEventNumInQueue

Interface definition:

uint32_t GetEventNumInQueue()

Function description:

Get the event queue length of the device, represents how much cache data is in the current event queue.

7.3.7. FlushEvent

Interface definition:

void FlushEvent()

Function description:

Flush the device event queue.

7.3.8. RegisterDeviceOfflineCallback

Interface definition:

GX_DEVICE_OFFLINE_CALLBACK_HANDLE RegisterDeviceOfflineCallback(
IDeviceOfflineEventHandler* pEventHandler, void* pUserParam)

Function description:

Register offline callback function.

7.3.9. UnregisterDeviceOfflineCallback

Interface definition:

void UnregisterDeviceOfflineCallback(GX_DEVICE_OFFLINE_CALLBACK_HANDLE hCallBack)

Function description:

Unregister offline callback function.

7.3.10. CreateImageProcessConfig

Interface definition:

CImageProcessConfigPointer CreateImageProcessConfig()

Function description:

Creating the image processing configuration parameter object.

7.3.11. ExportConfigFile

Interface definition:

void ExportConfigFile(const GxIAPICPP::gxstring& strFilePath)

Function description:

Export the current configuration parameter of the camera to the text file.

7.3.12. ImportConfigFile

Interface definition:

void ImportConfigFile(const GxIAPICPP::gxstring& strFilePath)

Function description:

Import the parameter which is in the configuration file to the camera.

7.3.13. Close

Interface definition:

void Close()

Function description:

Close the device.

7.4. IGXFeatureControl

Feature controller, the device object 'CGXDevicePointer' and stream object 'CGXStreamPointer' have their own feature controllers. Users can control all the function features of the camera via the feature controller, such as getting an integer function controller and then controlling read-write function via the integer controller. You can also register a feature update callback function for the feature. Users can get three different types of feature controllers by these three ways: IGXDevice::GetFeatureControl, IGXDevice::GetRemoteFeatureControl and IGXStream::GetFeatureControl.

Interface list:

GetFeatureNameList	Get all the function name string list which the current controller supports.
GetFeatureType	Get the data type of the current string feature: integer, float, enumerated etc., for more details please see the function definition: GX_FEATURE_TYPE.
IsImplemented	Check to see if the current feature control is available.

IsReadable	Check to see if the current feature is readable.
IsWritable	Check to see if the current feature is writable.
GetIntFeature	Get an integer controller.
GetFloatFeature	Get a float controller.
GetEnumFeature	Get an enumerated controller.
GetBoolFeature	Get a boolean controller.
GetStringFeature	Get a string controller.
GetCommandFeature	Get a command controller.
GetRegisterFeature	Get a register controller.
RegisterFeatureCallback	Register the function feature update callback function.
UnregisterFeatureCallback	Unregister the function feature update callback function.

7.4.1. GetFeatureNameList

Interface definition:

void GetFeatureNameList(GxIAPICPP::gxstring_vector& vectorFeatureNameList)

Function description:

Get all the function name string list which the current controller supports.

7.4.2. GetFeatureType

Interface definition:

GX_FEATURE_TYPE GetFeatureType(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get the data type of the current string feature: integer, float, enumerated etc., for more details please see the function definition: GX_FEATURE_TYPE.

7.4.3. IsImplemented

Interface definition:

bool IsImplemented(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Check to see if the current feature control is supported.

7.4.4. IsReadable

Interface definition:

bool IsReadable(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Check to see if the current feature is readable.

7.4.5. IsWritable

Interface definition:

bool IsWritable(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Check to see if the current feature is writable.

7.4.6. GetIntFeature

Interface definition:

CIntFeaturePointer GetIntFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get an integer feature controller.

7.4.7. GetFloatFeature

Interface definition:

CFloatFeaturePointer GetFloatFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get a float feature controller.

7.4.8. GetEnumFeature

Interface definition:

CEnumFeaturePointer GetEnumFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get an enumerated controller.

7.4.9. GetBoolFeature

Interface definition:

CBoolFeaturePointer GetBoolFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get a Boolean controller.

7.4.10. GetStringFeature

Interface definition:

CStringFeaturePointer GetStringFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get a string controller.

7.4.11. GetCommandFeature

Interface definition:

CCommandFeaturePointer GetCommandFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get a command controller.

7.4.12. GetRegisterFeature

Interface definition:

CRegisterFeaturePointer GetRegisterFeature(const GxIAPICPP::gxstring& strFeatureName)

Function description:

Get a register controller.

7.4.13. RegisterFeatureCallback

Interface definition:

GX_FEATURE_CALLBACK_HANDLE RegisterFeatureCallback(const GxIAPICPP::gxstring& strFeatureName,
IFeatureEventHandler* pEventHandler, void* pUserParam)

Function description:

Register the function feature update callback function.

7.4.14. UnregisterFeatureCallback

Interface definition:

void UnregisterFeatureCallback(GX_FEATURE_CALLBACK_HANDLE hCallback)

Function description:

Unregister the function feature update callback.

7.5. IIntFeature

Integer function node controller, corresponds to one integer node and can be get from IGXFeatureControl::GetIntFeature.

Interface list:

GetMin	Get the minimum value of integer function.
GetMax	Get the maximum value of integer function.
GetInc	Get the step length of integer function.
GetValue	Get the current value of integer function.
SetValue	Set the current value of integer function.

7.5.1. GetMin

Interface definition:

int64_t GetMin()

Function description:

Get the minimum value of integer function.

7.5.2. GetMax

Interface definition:

int64_t GetMax()

Function description:

Get the maximum value of integer function.

7.5.3. GetInc

Interface definition:

int64_t GetInc()

Function description:

Get the step length of integer function.

7.5.4. GetValue

Interface definition:

int64_t GetValue()

Function description:

Get the current value of integer function.

7.5.5. SetValue

Interface definition:

void SetValue(int64_t nValue)

Function description:

Set the current value of integer function.

7.6. IFloatFeature

The floating-point function node controller is corresponding to one float function node and can be get by the function: IGXFeatureControl::GetFloatFeature.

Interface list:

GetMin	Get the minimum value of float type.
GetMax	Get the maximum value of float type.
HasInc	Check to see if the float type value has step length.
GetInc	Get the step length of float type value. If HasInc interface returns ‘false’, then it will return 0 when calling this interface.

GetUnit	Get the unit of the float value.
GetValue	Get the current value of float type value.
SetValue	Set the current value of float type value.

7.6.1. GetMin

Interface definition:

double GetMin()

Function description:

Get the minimum value of float type.

7.6.2. GetMax

Interface definition:

double GetMax()

Function description:

Get the maximum value of float type.

7.6.3. HasInc

Interface definition:

bool HasInc()

Function description:

Check to see if the float type value has step length.

7.6.4. GetInc

Interface definition:

double GetInc()

Function description:

Get the step length of float type value. If HasInc interface returns ‘false’, then it will return 0 when calling this interface.

7.6.5. GetUnit

Interface definition:

GxIAPICPP::gxstring GetUnit()

Function description:

Get the unit of the float value.

7.6.6. GetValue

Interface definition:

double GetValue()

Function description:

Get the current value of the float type value.

7.6.7. SetValue

Interface definition:

void SetValue(double dValue)

Function description:

Set the current value of the float type value.

7.7. IEnumFeature

The enumeration function node controller is corresponding to one enumeration function node and can be get by the function: IGXFeatureControl::GetEnumFeature.

Interface list:

GetEnumEntryList	Get the enumeration item list which enumeration function supports.
GetValue	Get the current enumeration item value of enumeration function.
SetValue	Set the current enumeration item value of enumeration function.

7.7.1. GetEnumEntryList

Interface definition:

GxIAPICPP::gxstring_vector GetEnumEntryList()

Function description:

Get the enumeration item list which enumeration function supports.

7.7.2. GetValue

Interface definition:

GxIAPICPP::gxstring GetValue()

Function description:

Get the current enumeration item value of enumeration function.

7.7.3. SetValue

Interface definition:

void SetValue(const GxIAPICPP::gxstring& strValue)

Function description:

Set the current enumeration item value of enumeration function.

7.8. IBoolFeature

The boolean function node controller is corresponding to one Boolean function node and can be get by the function: IGXFeatureControl::GetBoolFeature.

Interface list:

GetValue	Get the current value of Boolean function.
SetValue	Set the current value of Boolean function.

7.8.1. GetValue

Interface definition:

bool GetValue()

Function description:

Get the current value of Boolean function.

7.8.2. SetValue

Interface definition:

void SetValue(bool bValue)

Function description:

Set the current value of Boolean function.

7.9. IStringFeature

The string function node controller is corresponding to one string function node and can be get by the function: IGXFeatureControl::GetStringFeature.

Interface list:

GetValue	Get the current value of the string.
SetValue	Set the current value of the string.
GetStringMaxLength	Get the maximum length of the string that can be set, not including the terminator.

7.9.1. GetValue

Interface definition: GxIAPICPP::gxstring GetValue()

Function description:

Get the current value of the string function.

7.9.2. SetValue

Interface definition:

void SetValue (const GxIAPICPP::gxstring&strValue)

Function description:

Set the current value of the string function.

7.9.3. GetStringMaxLength

Interface definition:

int64_t GetStringMaxLength()

Function description:

Get the maximum length of the string that can be set, not including the terminator '\0'.

7.10. ICommandFeature

The command function node controller is corresponding to one command function node and can be get by the function: IGXFeatureControl::GetCommandFeature.

Interface list:

Execute	Execute command.

7.10.1. Execute

Interface definition:

void Execute()

Function description:

Execute command.

7.11. IRegisterFeature

The register function node controller is corresponding to one command function node and can be get by the function: IGXFeatureControl:: GetRegisterFeature.

Interface list:

GetLength	Get the length of the register buffer. The user reads this length to read-write buffer.
GetBuffer	Get the value of the current register buffer. The length of input parameter 'ptrBuffer' must equal to the length that the interface 'GetLength' returned.
SetBuffer	Set the value of the current register buffer. The length of input parameter 'ptrBuffer' must equal to the length that the interface 'GetLength' returned.

7.11.1. GetLength

Interface definition:

int64_t GetLength()

Function description:

Get the length of the register buffer. The user reads this length to read-write buffer.

7.11.2. GetBuffer

Interface definition:

void GetBuffer(uint8_t* pBuffer, int64_t nLength)

Function description:

Get the value of the current register buffer. The length of input parameter 'ptrBuffer' must equal to the length that the interface 'GetLength' returned.

7.11.3. SetBuffer

Interface definition:

void SetBuffer(uint8_t* pBuffer, int64_t nLength)

Function description:

Set the value of the current register buffer. The length of input parameter 'ptrBuffer' must equal to the length that the interface 'GetLength' returned.

7.12. IGXSream

IGXStream is an abstract interface of the stream object, and it is responsible for the related operation of grabbing images, such as unregistering callback function and getting a single image. It can be get by the interface function: IGXDevice::OpenStream.

Interface list:

StartGrab	Start stream layer acquisition, whether using callback acquisition or a single image acquisition, the user must call the interface function StartGrab first.
StopGrab	Stop stream layer acquisition.
RegisterCaptureCallback	Register the acquisition callback function.
UnregisterCaptureCallback	Unregister the acquisition callback function.
GetImage	Get a single image.
GetFeatureControl	Get the stream layer feature controller to control the function feature of the stream layer.
FlushQueue	Flush the cache images in the acquisition queue.
SetAcqusitionBufferNumber	Set the count of acquisition buffer.
Close	Close the stream object.
GetOptimalPacketSize	Get the optimal packet length of the device.

7.12.1. StartGrab

Interface definition:

void StartGrab()

Interface description:

Start stream layer acquisition, whether using callback acquisition or a single image acquisition, the user must call the interface function StartGrab first and start stream layer acquisition, including start acquisition thread and resource allocated.

7.12.2. StopGrab

Interface definition:

void StopGrab()

Interface description:

Stop stream layer acquisition, including stop acquisition thread and release resources operation etc.

7.12.3. RegisterCaptureCallback

Interface definition:

void RegisterCaptureCallback(ICaptureEventHandler* pEventHandler, void *pUserParam)

Interface description:

Register the user acquisition callback function. Parameter 1 is the user callback object; parameter 2 is the user private parameter.

Function requirements:

1) After starting acquisition, you do not allowed to call the register interface function.
2) The user can register the acquisition callback function repeatedly, and using the overwrite mechanism.

7.12.4. UnregisterCaptureCallback

Interface definition:

void UnregisterCaptureCallback()

Interface description:

Unregister acquisition callback function.

7.12.5. GetImage

Interface definition:

CImageDataPointer GetImage(uint32_t nTimeout)

Interface description:

Get a single image. Parameter 1 is timeout value, the unit of parameter ‘nTimeout’ is ms. If the image is not captured within the given time, the timeout error will be threw.

Function requirements:

If the user registers the acquisition delegate function, it is not allowed to call GetImage interface. If call this interface forcibly, it will throw illegal call exception error.

7.12.6. GetFeatureControl

Interface definition:

CGXFeatureControlPointer GetFeatureControl()

Interface description:

Get stream layer feature controller, to control the stream layer function feature.

7.12.7. FlushQueue

Interface definition:

void FlushQueue()

Interface description:

Flush the cache images in the acquisition queue.

7.12.8. SetAcquisitionBufferNumber

Interface definition:

void SetAcqusitionBufferNumber(uint64_t nBufferNum)

Interface description:

Set the count of acquisition buffer.

Function requirements:

1) This interface is optional and is not necessary in the start acquisition process.
2) The setting value must be greater than 0, if the setting value is too large and exceed the system capacity, it will not to start acquisition successfully.
3) If need to call this interface, it must be called before starting acquisition and it is not allowed to call this interface during the acquisition process.
4) Once the user has set up the buffer number and has been successfully captured, the buffer number will remain valid until the stream object is closed.

7.12.9. Close

Interface definition:

void Close()

Interface description:

Close the stream object.

7.12.10. GetOptimalPacketSize

Interface definition:

uint32_t GetOptimalPacketSize(void)

Interface description:

Get the optimal packet length of the device.

Function requirements:

1) This interface only supports Gigabit network cameras to obtain optimal packet length.
2) If need to call this interface, it must be called before starting acquisition and it is not allowed to call this interface during the acquisition process.

7.13. IImageData

Image object, including image data buffer and image information. After the user registering acquisition callback function, when the images come, the image object will be passed to the acquisition callback function via an argument.

Interface list:

GetStatus	Get the status of the current frame, the status Indicates whether it is complete.
GetPayloadSize	Get the current frame size.
GetWidth	Get the width of the current frame.
GetHeight	Get the height of the current frame.
GetPixelFormat	Get the pixel format of the current frame.
GetFrameID	Get the current frame ID.
GetTimeStamp	Get the timestamp of the current frame.
GetBuffer	Get the image data buffer, return the IntPtr type pointer and point to the unmanaged memory directly.
ConvertToRaw8	The interface is used for non 8-bit data. The user can get the specified 8 valid bit in non 8-bit data.
ConvertToRGB24	Convert the raw data to the RGB24 bit data that is interpolated.
ImageProcess	To enhance the current image, and return the image after enhancing effect.

7.13.1. GetStatus

Interface definition:

GX_FRAME_STATUS_LIST GetStatus()

Function description:

Get the status of the current frame, the status indicates whether it is complete.

7.13.2. GetPayloadSize

Interface definition:

uint64_t GetPayloadSize()

Function description:

Get the current frame payloadload and the chunk data size. If the chunk mode was activated, this value is the sum of image data size and chunk data size; otherwise this value is the image data size only.

7.13.3. GetWidth

Interface definition:

uint64_t GetWidth()

Function description:

Get the width of the current frame.

7.13.4. GetHeight

Interface definition:

uint64_t GetHeight()

Function description:

Get the height of the current frame.

7.13.5. GetPixelFormat

Interface definition:

GX_PIXEL_FORMAT_ENTRY GetPixelFormat()

Function description:

Get the pixel format of the current frame.

7.13.6. GetFrameID

Interface definition:

uint64_t GetFrameID()

Function description:

Get the current frame ID.

7.13.7. GetTimeStamp

Interface definition:

uint64_t GetTimeStamp()

Function description:

Get the timestamp of the current frame.

7.13.8. GetBuffer

Interface definition:

void* GetBuffer()

Function description:

Get the image data buffer, return the IntPtr type pointer and point to the unmanaged memory directly.

7.13.9. ConvertToRaw8

Interface definition:

void*ConvertToRaw8(GX_VALID_BIT_LIST emValidBits)

Function description:

The interface is used for non 8-bit data. The user can get specified 8 valid bit in non 8-bit data.

7.13.10. ConvertToRGB24

Interface definition:

void*ConvertToRGB24(GX_VALID_BIT_LIST emValidBits, GX_BAYER_CONVERT_TYPE_LIST emConvertType, Bool bFlip)

Function description:

Convert the raw data to the RGB24 bit data by interpolation algorithm.

7.13.11. ImageProcess

Interface definition:

void*ImageProcess(CImageProcessConfigPointer &objImageProcessConfigPtr)

Function description:

To enhance the current image, and returns the enhanced image data.

Note:

1) If the 'IImageProcessConfig::IsAccelerate' returns 'true', then it will process the images by accelerating mode. At this time the image height must be an integer multiple of four, otherwise an error will occur in the interface.
2) The mono camera void* returns 8bit image data, the image size is image width * image height.
3) The color camera void* returns RGB image data, image size is image width * image height * 3.

7.14. IImagProcessConfig

The configurable parameter object of Image enhancement, which has a set of configuration parameter inside, and can be get by the interface function: IGXDevice::CreateImageProcessConfig method.

Interface list:

SetValidBit	Select the specified 8 bit valid data that to be get, and the interface is used to specify which 8bit is selected.
GetValidBit	Get the 8 valid bits which are currently specified.
EnableDefectivePixelCorrect	Enable defective-pixel correction.
IsDefectivePixelCorrect	Check whether the defective-pixel correction is enabled currently.
EnableSharpen	Enable sharpness function.
IsSharpen	Check whether the sharpness function is enabled.
EnableAccelerate	Enable accelerating image processing.
IsAccelerate	Check whether it works in the accelerating enable status currently.
SetSharpenParam	Set the sharpness intensity factor.
GetSharpenParam	Get the sharpness intensity factor currently used.
SetContrastParam	Set contrast adjustment parameter.
GetContrastParam	Get the contrast adjustment parameter currently used.
SetGammaParam	Set Gamma coefficient.
GetGammaParam	Get Gamma coefficient.
SetLightnessParam	Set brightness adjustment parameter.
GetLightnessParam	Get brightness adjustment parameter.

EnableDenoise	Enable denoise function.
IsDenoise	Check whether the denoise function is enabled currently.
SetSaturationParam	Set saturation coefficient.
GetSaturationParam	Get saturation coefficient.
SetConvertType	Set image format conversion algorithm.
GetConvertType	Get image format conversion algorithm.
EnableConvertFlip	Enable image format conversion flip.
IsConvertFlip	Check whether the image format conversion flip function is enabled currently.
EnableColorCorrection	Enable color correction.
IsColorCorrection	Check whether the color correction function is enabled.
Reset	Restore the default adjustment parameter.
IsUserSetCCParam	Check whether the color correction function is user-set mode.
EnableUserSetCCParam	Enable user-set mode for color correction function.
SetUserCCParam	Set color transform factor by user.
GetUserCCParam	Get color transform factor which set by user.

7.14.1. SetValidBit

Interface definition:

void SetValidBit(GX_VALID_BIT_LISTemValidBits)

Function description:

Select the specified 8 bit valid data that to be get, and the interface is used for non-8bit raw data.

7.14.2. GetValidBit

Interface definition:

GX_VALID_BIT_LIST GetValidBit()

Function description:

Get specified 8 bit valid data, and the interface is used for non-8bit raw data.

7.14.3. EnableDefectivePixelCorrect

Interface definition:

void EnableDefectivePixelCorrect(bool bEnable)

Function description:

Enable defective-pixel correction function. The defective pixel correction is enabled when the ‘bEnable’ is ‘ture’, the defective pixel correction is disable when the ‘bEnable’ is ‘false’.

7.14.4. IsDefectivePixelCorrect

Interface definition:

bool IsDefectivePixelCorrect()

Function description:

Get the status of defective pixel correction.

7.14.5. EnableSharpen

Interface definition:

void EnableSharpen(bool bEnable)

Function description:

Enable sharpen function. When the ‘bEnable’ is ‘ture’, the sharpness function is enabled; when the ‘bEnable’ is ‘false’ the sharpness function is disabled.

7.14.6. IsSharpen

Interface definition:

bool IsSharpen()

Function description:

Get the status of sharpness.

7.14.7. EnableAccelerate

Interface definition:

void EnableAccelerate(bool bEnable)

Function description:

Image processing accelerating enable. If it is set ‘true’, the accelerating function is enabled and if it is set ‘false’, the accelerating function is disabled. If the CPU of the current PC is not allowed to accelerate because of itself limitation, an error will occurs when the user sets ‘true’.

Note: When the acceleration function is enabled, the current image height must be an integer multiple of four, otherwise an error will occurs in the IImageData::ImageProcess interface.

7.14.8. IsAccelerate

Interface definition:

bool IsAccelerate()

Function description:

Check whether the device works in the accelerating enable status currently. ‘True’ means accelerating and ‘false’ means not accelerating.

7.14.9. SetSharpenParam

Interface definition:

void SetSharpenParam(double dParam)

Function description:

Set the sharpness intensity factor.

7.14.10. GetSharpenParam

Interface definition:

double GetSharpenParam()

Function description:

Get the sharpness intensity factor currently used.

7.14.11. SetContrastParam

Interface definition:

void SetContrastParam(int32_t nParam)

Function description:

Set contrast adjustment parameter.

7.14.12. GetContrastParam

Interface definition:

int32_t GetContrastParam()

Function description:

Get the contrast adjustment parameter currently used.

7.14.13. SetGammaParam

Interface definition:

void SetGammaParam(double dParam)

Function description:

Set Gama coefficient.

7.14.14. GetGammaParam

Interface definition:

double GetGammaParam()

Function description:

Get Gamma coefficient.

7.14.15. SetLightnessParam

Interface definition:

void SetLightnessParam(int32_t nParam)

Function description:

Set brightness adjustment parameter.

7.14.16. GetLightnessParam

Interface definition:

int32_t GetLightnessParam()

Function description:

Get brightness adjustment parameter.

7.14.17. EnableDenoise

Interface definition:

void EnableDenoise(bool bEnable)

Function description:

Enable denoise switch (not support for monochrome cameras), when the ‘bEnable’ is ‘ture’, the denoise function is enabled; when the ‘bEnable’ is ‘false’, the denoise function is disabled.

7.14.18. IsDenoise

Interface definition:

bool IsDenoise()

Function description:

Get denoise switch (not support for monochrome cameras).

7.14.19. SetSaturationParam

Interface definition:

void SetSaturationParam(int32_t nParam)

Function description:

Set saturation coefficient (not support for monochrome cameras).

7.14.20. GetSaturationParam

Interface definition:

int32_t GetSaturationParam()

Function description:

Get saturation coefficient (not support for monochrome cameras).

7.14.21. SetConvertType

Interface definition:

void SetConvertType(GX_BAYER_CONVERT_TYPE_LIST emConvertType)

Function description:

Set image format conversion algorithm (not support for monochrome cameras).

7.14.22. GetConvertType

Interface definition:

_GX_BAYER_CONVERT_TYPE_LIST GetConvertType()

Function description:

Get interpolation algorithm (not support for monochrome cameras).

7.14.23. EnableConvertFlip

Interface definition:

void EnableConvertFlip(bool bFlip)

Function description:

Enable interpolation flip (not support for monochrome cameras).

7.14.24. IsConvertFlip

Interface definition:

bool IsConvertFlip()

Function description:

Get interpolation flip identification (not support for monochrome cameras), ‘true’ means flipping and ‘false’ means not flipping.

7.14.25. EnableColorCorrection

Interface definition:

void EnableColorCorrection(bool bEnable)

Function description:

Enable color correction (not support for monochrome cameras), ‘true’ means enabling color correction and ‘false’ means disabling color correction.

7.14.26. IsColorCorrection

Interface definition:

bool IsColorCorrection()

Function description:

Check whether the color correction is enabled (not support monochrome cameras).

7.14.27. Reset

Interface definition:

void Reset()

Function description:

The user can select to restore the optimal default parameter configuration.

7.14.28. IsUserSetCCParam

Interface definition:

bool IsUserSetCCParam()

Function description:

Check whether the color correction function is user-set mode (not support for monochrome cameras). Returns true if it is user-set mode, otherwise returns false.

7.14.29. EnableUserSetCCParam

Interface definition:

void EnableUserSetCCParam(bool bIsUserMode)

Function description:

Enable user-set mode for color correction function (not support for monochrome cameras). If bIsUserMode is true the user-set mode is enabled; otherwise it is disabled.

7.14.30. SetUserCCParam

Interface definition:

void SetUserCCParam(COLOR_TRANSFORM_FACTOR stColorTransformFactor)

Function description:

Set stColorTransformFactor to color correction function (only supported for user-set mode) (The recommended range of color correction structure parameters is -4 ~ 4. If the parameter setting is less than -4, the correction effect is consistent with -4. If the parameter setting is greater than 4, the correction effect is consistent with 4.).

7.14.31. GetUserCCParam

Interface definition:

COLOR_TRANSFORM_FACTOR GetUserCCParam()

Function description:

Get current stColorTransformFactor which used to color correction function (only supported for user-set mode).

7.15. ICaptureEventHandler

ICaptureEventHandler is a pure virtual class, and cannot construct objects directly. The user needs to inherit this class themselves to implement callback function DoOnImageCaptured internal operation.

Interface list:

DoOnImageCaptured	Callback response function, call this function when the frame is received.

7.15.1. DoOnImageCaptured

Interface definition:

void DoOnImageCaptured(CImageDataPointer& objImageDataPointer, void* pUserParam)

Function description:

Callback response function, callback this function after receiving one frame.

7.16. IDeviceOfflineEventHandler

IDeviceOfflineEventHandler is a pure virtual class, and cannot construct objects directly. The user needs to inherit this class themselves to implement callback function DoOnDeviceOfflineEvent internal operation.

Interface list:

DoOnDeviceOfflineEvent	Callback response function, call this function when the device is offline.

7.16.1. DoOnDeviceOfflineEvent

Interface definition:

void DoOnDeviceOfflineEvent(void* pUserParam)

Function description:

Callback response function, call this function when the device is offline.

7.17. IFeatureEventHandler

IFeatureEventHandler is a pure virtual class, and cannot construct objects directly. The user needs to inherit this class themselves to implement callback function DoOnFeatureEvent internal operation.

Interface list:

DoOnFeatureEvent	Callback response function, call this function when the device event comes.

7.17.1. DoOnFeatureEvent

Interface definition:

void DoOnFeatureEvent(const GxIAPICPP::gxstring& strFeatureName, void* pUserParam)

Function description:

Callback response function, call this function when the device event comes.

8. Sample Project

Below is a complete example of console code that can register device offline events, register remote device events, and register an acquisition callback event.

```
#include"stdafx.h"
#include<iostream>
using namespace std;

// Please configure the project header file directory in advance, it
//needs to include the GalaxyIncludes.h.
#include"GalaxyIncludes.h"

//The user inherits the offline event handle class.
class CSampleDeviceOfflineEventHandler : public IDeviceOfflineEventHandler
{
public:
    void DoOnDeviceOfflineEvent(void* pUserParam)
    {
        cout<<" Received the device offline event!"<<endl;
    }
};

// The user inherits the feature update event process class.
class CSampleFeatureEventHandler : public IFeatureEventHandler
{
public:
    void DoOnFeatureEvent(const GxIAPICPP::gxstring&strFeatureName,
                          void* pUserParam)
    {
        cout<<" Received the exposure end event!"<<endl;
    }
};

// The user inherits the acquisition event handle class.
class CSampleCaptureEventHandler : public ICaptureEventHandler
{
public:
    void DoOnImageCaptured(CImageDataPointer&objImageDataPointer,
                           void* pUserParam)
    {
        cout<<" Captured one image!"<<endl;
        cout<<"ImageInfo: "<<objImageDataPointer->GetStatus() <<endl;
        cout<<"ImageInfo: "<<objImageDataPointer->GetWidth() <<endl;
        cout<<"ImageInfo: "<<objImageDataPointer->GetHeight() <<endl;
        cout<<"ImageInfo: "<<objImageDataPointer->GetPayloadSize() <<endl;
    }
};

int _tmain(int argc, _TCHAR* argv[])
{
    // Declare the event callback object pointer.
    ///< Offline event callback object.
    IDeviceOfflineEventHandler* pDeviceOfflineEventHandler = NULL;

     ///< Remote device event callback object.
    IFeatureEventHandler* pFeatureEventHandler = NULL;

    ///<Acquisition callback object.
    ICaptureEventHandler* pCaptureEventHandler = NULL;
```

```
// Initialization.
IGXFactory::GetInstance().Init();

try
{
    do
    {
        // Enumeration device.
        gxdeviceinfo_vector vectorDeviceInfo;
        IGXFactory::GetInstance().UpdateDeviceList(1000, vectorDeviceInfo);
        if (0 == vectorDeviceInfo.size())
        {
            cout<<" NO device available!"<<endl;
            break;
        }

        // Open the first device and the first stream of the device.
        CGXDevicePointer ObjDevicePtr =
                              IGXFactory::GetInstance().OpenDeviceBySN(
                                          vectorDeviceInfo[0].GetSN(),
                                          GX_ACCESS_EXCLUSIVE);
        CGXStreamPointer ObjStreamPtr = ObjDevicePtr->OpenStream(0);

        // Get the remote device feature controller.
        CGXFeatureControlPointer ObjFeatureControlPtr =
                             ObjDevicePtr->GetRemoteFeatureControl();

        // Get the stream layer feature controller.
        CGXFeatureControlPointer objStreamFeatureControlPtr =
                                  ObjStreamPtr->GetFeatureControl();

        // To improve the acquisition performance of the GigE camera,
        //please refer to the following code for the setting method
        //(Only GigE cameras support setting the optimal packet size).
        GX_DEVICE_CLASS_LIST objDeviceClass =
                        ObjDevicePtr->GetDeviceInfo().GetDeviceClass();
        if (GX_DEVICE_CLASS_GEV == objDeviceClass)
        {
        // Determine whether the device supports the stream channel
        //data packet function.
            if (true == ObjFeatureControlPtr->IsImplemented(
                                                  "GevSCPSPacketSize"))
            {
                // Get the optimal packet size of the current network
                //environment.
                int nPacketSize = ObjStreamPtr->GetOptimalPacketSize();

                // Set the optimal packet size to the stream channel
                //packet size of the current device.
                ObjFeatureControlPtr->GetIntFeature(
                           "GevSCPSPacketSize")->SetValue(nPacketSize);
            }
        }

        // Set the buffer processing mode.
        objStreamFeatureControlPtr->GetEnumFeature(
                "StreamBufferHandlingMode")->SetValue("OldestFirst");

        // Register the device offline event (for GigE cameras only).
        GX_DEVICE_OFFLINE_CALLBACK_HANDLE hDeviceOffline = NULL;
        pDeviceOfflineEventHandler =
```

```
                                    new CSampleDeviceOfflineEventHandler();
        hDeviceOffline =ObjDevicePtr->RegisterDeviceOfflineCallback(
                                        pDeviceOfflineEventHandler, NULL);

        // Set the exposure time (the value below is just for example)
        //ObjFeatureControlPtr->GetFloatFeature(
                                           "ExposureTime")->SetValue(50);

        // Register the remote device event: exposure end event (for
        //GigE cameras only).
        // Select the event source.
        ObjFeatureControlPtr->GetEnumFeature(
                           "EventSelector")->SetValue("ExposureEnd");

        // Enable events.
        ObjFeatureControlPtr->GetEnumFeature(
                             "EventNotification")->SetValue("On");
        GX_FEATURE_CALLBACK_HANDLE hFeatureEvent = NULL;
        pFeatureEventHandler = new CSampleFeatureEventHandler();
        hFeatureEvent = ObjFeatureControlPtr->RegisterFeatureCallback(
                     "EventExposureEnd", pFeatureEventHandler, NULL);

        // Register the callback acquisition.
        pCaptureEventHandler = new CSampleCaptureEventHandler();
        ObjStreamPtr->RegisterCaptureCallback(pCaptureEventHandler,
                                              NULL);

        // Send the start acquisition command.
        ObjStreamPtr->StartGrab();
        ObjFeatureControlPtr->GetCommandFeature(
                                    "AcquisitionStart")->Execute();

        // Start acquisition successfully, the console prints
        //information until any key is entered to continue.
        getchar();

        // Send the stop acquisition command.
        ObjFeatureControlPtr->GetCommandFeature(
                                           "AcquisitionStop")->Execute();
        ObjStreamPtr->StopGrab();

        // Unregister the acquisition callback.
        ObjStreamPtr->UnregisterCaptureCallback();
        // Uunregister the remote device event.
        ObjFeatureControlPtr->UnregisterFeatureCallback(hFeatureEvent);
        // Unregister the device offline event.
        ObjDevicePtr->UnregisterDeviceOfflineCallback(hDeviceOffline);

        // Release resources.
        ObjStreamPtr->Close();
        ObjDevicePtr->Close();
    } while (0);
}
catch(CGalaxyException&e)
{
    cout<<" Error code: "<<e.GetErrorCode() <<endl;
    cout<<" The description information of error: "<<e.what() <<endl;
}
catch(std::exception&e)
{
    cout<<" The description information of error: "<<e.what() <<endl;
```

```
    }

    // Uninitialization library.
    IGXFactory::GetInstance().Uninit();

    // Destroy the event callback pointer.
    if (NULL != pCaptureEventHandler)
    {
        delete pCaptureEventHandler;
        pCaptureEventHandler = NULL;
    }
    if (NULL != pDeviceOfflineEventHandler)
    {
        delete pDeviceOfflineEventHandler;
        pDeviceOfflineEventHandler = NULL;
    }
    if (NULL != pFeatureEventHandler)
    {
        delete pFeatureEventHandler;
        pFeatureEventHandler = NULL;
    }
    return 0;
}
```

9. Revision History

No.	Version	Changes	Date
1	V1.0.1	Initial release	2015-11
2	V1.0.4	1. Add some informations of GetImage 2. Added description that remote device are only supported by gigabit networks 3. Add description and link of callback event virtual base class 4. Add a section about the calling process	2016-02-01
3	V1.0.6	Modifiy the description of GetImage	2016-02-04
4	V1.0.7	Modify bugs of 674, 675, 676 in the first round of C++ system testing	2016-03-21
5	V1.0.9	Add the interface description of GigEIPConfiguration and GigEForceIp	2016-07-20
6	V1.0.11	Add the interface description of the acquisition buffer	2016-10-24
7	V1.0.12	Added description of camera internal error event: internal error event ID and time stamp	2016-12-14
8	V1.1.1	Modify section 2.10.4 Gamma default value to 1.0, contrast default value to 0, saturation default value to 64	2019-07-01
9	V1.1.2	1. Add new device reset and reconnect port 2. Add some description information to the ImageProcess interface	2019-07-16
10	V1.1.3	1. Added an interface for obtaining the optimal packet length of the device 2. Add the code for setting the optimal packet length of the device	2019-08-15
11	V1.1.4	1. Add a data type: COLOR_TRANSFORM_FACTOR 2. Add 4 user-set color correction factor interfaces in IImageProcessConfig	2019-09-29
12	V1.2.0	Add related parameters of StreamBufferHandlingMode	2019-12-25
13	V1.2.1	Add the description of how to get the sample code for feature read & write in section 2.8.5	2020-08-28
14	V1.2.2	Add the heartbeat timeout time modified method	2020-10-12