# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>SUMMARY</strong></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>STATEMENT</strong></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td><strong>OUTLINE</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>FILE STRUCTURE</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>DEVELOPMENT ENVIRONMENT</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>PRECAUTIONS (COMMON PROBLEM)</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>INDUSTRIAL CAMERA COMMON WORDS EXPLAIN</td>
<td>3</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Line camera and Plane Array Camera</td>
<td>3</td>
</tr>
<tr>
<td>1.4.2</td>
<td>Rolling shutter and Rolling shutter</td>
<td>4</td>
</tr>
<tr>
<td>1.4.3</td>
<td>Hardware trigger (External trigger)</td>
<td>4</td>
</tr>
<tr>
<td>1.4.4</td>
<td>Sensor size (Target surface)</td>
<td>5</td>
</tr>
<tr>
<td>1.4.5</td>
<td>Frame buffer and caching</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td><strong>QUICK DEVELOPMENT GUIDE</strong></td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>CAMERA OPERATION PROCESS OVERVIEW</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>DEVELOP OF DEMO</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Basic of demo</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Advanced of demo</td>
<td>11</td>
</tr>
<tr>
<td>2.2.3</td>
<td>MultiCamera demo (using multiple cameras)</td>
<td>12</td>
</tr>
<tr>
<td>2.2.4</td>
<td>OCX of demo(ActiveX)</td>
<td>12</td>
</tr>
<tr>
<td>2.2.5</td>
<td>MultiExposure of demo(A camera double exposure, dynamic range effects)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>ImageFormat&amp;Saving of demo (Gray, RGB24, RGB32 Formatting)</td>
<td>13</td>
</tr>
<tr>
<td>2.2.7</td>
<td>TriggerAndStrobe of demo (Control trigger signal and the flash)</td>
<td>14</td>
</tr>
<tr>
<td>2.2.8</td>
<td>UserDataTest of demo (To customize the camera to read and write data)</td>
<td>15</td>
</tr>
<tr>
<td>2.2.9</td>
<td>SnapshotOnPreview of demo (Small resolution preview high-speed, high-resolution camera)</td>
<td>16</td>
</tr>
<tr>
<td>2.2.10</td>
<td>RawTransTest of test (Offline RAW files into BMP, JPG file)</td>
<td>17</td>
</tr>
<tr>
<td>2.2.11</td>
<td>LineScan of demo (With Line scan mode)</td>
<td>17</td>
</tr>
<tr>
<td>2.2.12</td>
<td>ROI of demo (How to customize Region of interest)</td>
<td>18</td>
</tr>
<tr>
<td>2.2.13</td>
<td>GPIO of demo(Only for GPIO models)</td>
<td>18</td>
</tr>
<tr>
<td>2.2.14</td>
<td>SaveFile of demo (A preview image of each successive save to disk)</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>DEBUGGING CAMERA PARAMETERS</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1</td>
<td>How to set the exposure time (Anti-Strobe, No smear, Enhance the dynamic range)</td>
<td>20</td>
</tr>
</tbody>
</table>
2.3.2 How to get better image color ................................................. 21
2.3.3 How to improve image clarity ................................................. 22
2.3.4 BayerRestore Selection Algorithm (Contribute to a better image contour extraction) .................................................. 22
2.3.5 How to reduce CPU usage .................................................. 22
2.3.6 The camera parameters are saved and loaded ................................................. 22

3 SDK DATA TYPE DEFINITION ................................................. 25
3.1 STRUCTURE IS DEFINED .................................................................. 25
3.2 PARAMETER TYPE DEFINITION ................................................. 31
3.3 THE RETURN VALUE IS DEFINED INTERFACES (ERROR CODE) .......... 51

4 SDK INTERFACE FUNCTION DESCRIPTION (C/C++ VB DELPHI C# COMMON) ................................................. 57

5 SDK INTERFACE FUNCTION BY FUNCTION EXPLANATION ................. 165
5.1 CAMERA INITIALIZATION AND DEINITIALIZATION ......................... 165
5.2 SAVING AND LOADING THE CAMERA PARAMETERS ....................... 167
5.3 CAMERAS CAPTURE (ACTIVE CAPTURE OR CALLBACK MODE) ........... 168
5.4 PREVIEW IMAGE BY THE DISPLAY CONTROLS ................................................. 170
5.5 ADJUST THE CAMERA IMAGE BRIGHTNESS (SET THE EXPOSURE TIME) .... 170
5.6 SWITCH TO A DIFFERENT RESOLUTION AND A CUSTOM RESOLUTION (ROI FUNCTION) ....... 171
5.7 SET THE CAMERA’S CONTRAST, GAMMA, SATURATION, SHARPNESS AND OTHER PARAMETERS ISP 171
5.8 COLOR CAMERAS SWITCH TO BLACK AND WHITE CAMERA ................. 172
5.9 MULTIPLE CAMERAS SIMULTANEOUSLY, AND ESTABLISH A CORRESPONDENCE BETWEEN THE CAMERA ........................................................................... 173
5.10 FLIP AND MIRROR IMAGE ROTATION ................................................. 174
5.11 TEXT OVERLAY FEATURE ON THE IMAGE ................................................. 175
5.12 SAVE PICTURES AND VIDEO FEATURES ................................................. 175
5.13 SETTING THE CAMERA IMAGE OUTPUT BIT DEPTH ............................. 176
5.14 SET THE PIXEL FORMAT OF THE IMAGE (8-BIT GRAYSCALE, 24,32,48-BIT COLOR) .... 176
5.15 TO THE ORIGINAL RAW IMAGE PROCESSING ................................................. 177
5.16 THE GIGE CAMERA USE THE API TO DYNAMICALLY SET IP, GATEWAY, SUBNET MASK 177
5.17 SET THE CAMERA’S FRAME RATE ................................................. 178
5.18 USE A SOFT OR HARD TRIGGER TRIGGER FUNCTION ............................. 178
5.19 RESISTANCE TO INTERFERENCE EXTERNAL TRIGGER MODE .................. 179
5.20 SET THE TRIGGER DELAY TIME UNDER EXTERNAL TRIGGER MODE ........ 179
5.21 DETECTING CAMERA DROPPED AND AUTOMATICALLY RECONNECT .......... 179
5.22 READ AND WRITE THE SERIAL NUMBER OF THE CAMERA .................. 180
5.23 READING AND WRITING CUSTOM DATA IN THE CAMERA .................. 180
5.24 GET A STRING DESCRIBING THE ERROR CODE CORRESPONDING TO THE INFORMATION. 180
GIGE VISION AND USB3 VISION INTERPRETATION OF THE DEFINITION OF XML DOCUMENTS

THE HALCON DEVELOPMENT GUIDANCE

7.1 SECONDARY DEVELOPMENT BY HALCON
7.2 C / C++, VB, C# DEVELOPMENT BY HALCON
7.3 SIMULTANEOUS USE OF MULTIPLE CAMERA DEVELOPMENT

THE LABVIEW DEVELOPMENT GUIDANCE

8.1 SECONDARY DEVELOPMENT BY LABVIEW
8.2 BASE DEVELOPED THROUGH DLL CALLS
8.3 USING MULTIPLE THE CAMERA IN THE LABVIEW
8.4 MULTI-CAMERA DISTINGUISH LABVIEW

WHEN INTEGRATED THE CAMERA INSTALLATION FILES RELEASED

10 TECHNICAL SUPPORT
Summary

This document provides for the use of our industrial cameras for secondary development of the user, to use the system for the development of a dynamic link library function is described in detail, and provides a quick programming guide, so that in a short time be able to be integrated into the camera the user’s system.
Statement

The software copyright, copyright and intellectual property rights to all of the Company, and by the "People's Republic of China Copyright Law", "Computer Software Protection organized", "Intellectual Property Protection Ordinance" and related international copyright treaties, laws, regulations, and other knowledge protection of property rights laws and treaties. Any unit or individual can not be used without our authorization, modify or republish any part of the Software, or will be deemed illegal infringement, the company claimed prosecuted under the law are reserved. This provision also applies to the company have full rights of text, pictures, tables and other content.

This article documents the information and the like is provided only for your convenience and may be superseded by the updated information, the Company without notice. Ensure that your application meets with your specifications is your own responsibility. The company of such information does not make any express or implied, written or oral, statutory or other representations or warranties, including without limitation for its use, the applicability of representations or warranties of quality, performance, merchantability or fitness for a particular purpose.
1 Outline

1.1 File Structure

In the second development, it is necessary directly to the library file located in the installation directory of the SDK folder, divided into two 32-bit and 64-bit development kit. 64-bit SDK development files, located SDK / X64 folder, relative to the 32-bit SDK file, 64-bit SDK file names are _X64 end. content. MVCAMSDK.DLL file SDK directory is the camera's SDK dynamic link library, all the cameras provide external interface functions, VC / C ++, VB, VB.net, Delphi, C # routines are references to the library file. At the same time, for ease of reference, MVCAMSDK.DLL will be copied to the system when you install the System32 directory (WIN64 system copy when in syswow64 directory), you are in the secondary development, you can use the file name directly accessed without concern MVCAMSDK.DLL path file, the system will automatically find the file in the System32 directory.

If you need to publish your installation package, please refer to Chapter 7 of the method.

Develop a routine located in the installation directory the following subdirectories:
- Demo / VC ++. Based on VC ++ development routines provided VC6 and VS2010 project file.
- Demo / VB6. VB6 developed routines based.
- Demo / VB.net. Based VB.net development routines.
- Demo / C #. Based on C # (VS2010) developed routines.
- Demo / C ++ / OpenCV. Based on the OpenCV example VS2010.
- Demo / LabView. Based on ActiveX and DLL interface calls the development of routines.

Wherein VC ++ provides Basic, BasicEx, Advanced, OCX, MultiCamera, MultiExposure more routine, VB6, Delphi6, C # provides only Basic routines that. Basic and Advanced routines include the two most of the functional operation of the camera, you can select different routines based on development needs to learn how to use our SDK. About their own characteristics and scope of Basic and Advanced routines, refer to Chapter 2 content.

1.2 Development environment

SDK is a standard C language interface dynamic link library that can be loaded into C, C ++, C #, VB, Labview, Delphi and other development tools.
Currently, we offer based on VC ++ 6.0, VS2010, VB6.0, VB.net, Delphi6, C #, Labview (8.6 9.0 2010 2011 2012), Halcon (9,10,11,12), OpenCV (2.4) routines. For more than VC6 version of the development tools such as VS2003 - VS2013 and so on, you can turn on VC ++ 6.0 DEMO project DSP or DSW file directly, then follow the prompts to convert engineering.

1.3 Precautions (common problem)

- Since the SDK requires access to the registry, so in WIN7 and WIN8 system, if a non-administrator user logs on, you need to run with administrator privileges camera DEMO program to access the camera, or the camera will be reported to initialize like -13 error. Common practice is to log on the system administrator, or you develop the program, the application administrator privileges. For MFC project, in the Project Settings -> Linker -> manifest file, the following settings:

```
requireAdministrator (/level="requireAdministrator")
```

If you are using DELPHI, C #, VB6 like development language, administrator privileges can be obtained by a similar method.

- to switch between different versions of the installation package. Different versions of the SDK, we maintain compatibility API interface, but the camera's kernel driver between different software versions, you may need to manually switch, after you install a new version, the camera prompts initialization fails, you need to manually in the Device Manager updated camera kernel driver.

1.4 Industrial Camera common words explain

1.4.1 Line camera and Plane Array Camera

Linear array camera, the camera is the use of the line image sensor. CCD line image sensors to the main line data to a few tens of K or K, but a height of only a few pixels, the line frequency is very high, you can go to the tens of thousands of lines per second, suitable for very high precision, wide-frame scanning.
Plane array camera, the camera side is the use of image sensors, CMOS and CCD camera has a planar array, plane array camera resolution compared to the width of a line linear array camera will be much smaller, but the whole picture-frame sensor, an imaging pixel height will be much larger than the linear array camera. And the program developed relatively simple, once you can get a whole sub-picture without stitching each row of data.

Currently, most of the applications, or the use of the line camera-based. Prices, line scan cameras are expensive, often plane array camera several times to several times.

1.4.2 Rolling shutter and Rolling shutter

Rolling shutter and global shutter mainly for the purposes of plane array camera.

Rolling shutter of the camera, in the photographic, is carried out line by line, from the first line, scroll to the last line has been sensitive, the exposure starting time point of each line is not the same, while exposing the side of the output image data; a global shutter camera, when the photosensitive, the entire surface of the array beginning at the same time, while the end of the whole after the end of a one-time data read out.

Due to limitations rolling shutter principle on the rolling shutter of the camera is not suitable for shooting fast moving picture, compared to a global shutter camera rolling shutter camera shooting a moving picture, each line of the image will have some displacement deviations, eventually leading image distortion. Here to explain this distortion, not a smear, many people mistakenly call this phenomenon is understood to smear, smear is due to the velocity of the object shooting too much and too long exposure time and camera settings cause, smear can cause image blur, and rolling shutter distortion caused by, but each line of the image sharpness is not affected.

1.4.3 Hardware trigger (External trigger)

Under normal mode, the camera began to work, is to keep a continuous image acquisition, post-acquisition End of the frame, immediately began collecting the next frame, and so on.

In some industrial applications, the camera does not need to have been continuously capture images, but after waiting for a specific event occurs only want the camera to capture an image and get the results of treatment, in this case, you need to use hardware (external ) trigger mode, this way of working to support industrial cameras will leave special trigger connector on the camera, usually 4-pin to 12-pin connectors, ranging from aviation. After entering the trigger mode, the camera will wait for a valid signal, the validity of the signal can be set by software, such as high, low, or the way the upper and lower edge transition mode, if there has been no valid signal, the camera will not output any image data. When you find a program capture timeout, check to see if the camera is set to trigger mode on the interface.
1.4.4 Sensor size (Target surface)

The optical size refers to the size of the camera's sensor area. Common sizes 1/4”, 1/3” 1/2.5” 1/2”, 2/3,” and so on, you need to select the size of the lens, in order to achieve the effect of the matching.

1.4.5 Frame buffer and caching

Industrial camera with caching function means that the camera has certain data cached image data capabilities, but does not have the capacity of the entire one image data is cached, therefore, when the transmission bandwidth is not enough, or the transmission line is not enough reliable and stable, it may result in buffer overflow, resulting in an image frame can not be rebuilt, resulting in dropped frames or completely not figure phenomenon.

Industrial camera with a frame memory function means that the camera has stored inside the camera under the complete image of the ability to frame, so that, when the transmission bandwidth is not enough, or the transmission line is not enough reliable and stable, with the camera frame buffer function can still be broken point resume, on the PC side can be reconstructed image frame.

Industrial cameras are certainly with caching functionality, but not necessarily with a frame memory function. A typical example to illustrate the benefits of the camera frame buffer: When connected to 16 cameras to take pictures with the external trigger mode, upon receipt of a trigger signal, which began at the same time 16 camera exposure, and the images are stored in the camera first, then PC end can follow any order to read out the image of the 16 camera, without worrying about the length of time to read and bandwidth adequacy. And if there is no frame memory function, which 16 camera images may be lost. But without the camera frame buffer also has its advantages, cost-effective, simple structure, suitable for a computer connected to the camera 1-2.
2 Quick Development Guide

2.1 Camera operation Process Overview

For you to quickly and accurately develop, please be patient reading this chapter content.

We recommend that you follow the following procedures to operate the camera (some of which step is optional, it has been indicated):

1) the method of loading the SDK dynamic link library MVCAMSDK.DLL.
   You can use dynamic or static load in two ways.

   - If you use the C / C ++ development, engineering reference CameraApi.h
     header file (located in the installation directory of the SDK / DEMO / VC ++ / include), and MVCAMSDK.lib library files (located in the installation directory of the SDK folder), and then when you can reference the SDK interface function directly in the project, but MVCAMSDK.DLL and your application must be placed in the same directory system32 directory or the system, placed in another directory, you must set the system environment variables (PATH).

   If you are using VB for development, it can be loaded by our VB6 routines in a similar manner to directly define and specify the interface functions SDK DLL files to their references. SDK \ Demo \ VB6 \ Module \ CameraApi.bas modules cover all SDK interfaces, each a function interface MVCAMSDK.DLL are exported image function can become VB call.

   If you use Delphi development, it can be loaded by our Delphi6 routine similar manner to directly define and specify the interface functions SDK DLL files to their references. SDK \ Demo \ Delphi6 \ Units \ CameraApi.pas unit encompasses all SDK interfaces, each a function interface MVCAMSDK.DLL are exported image became Delphi function can be called.

   If you are using C # development, can be loaded by our C # routine similar methods, C # loading MVCAMSDK.DLL procedures and VB, Delphi Similarly, in order to facilitate the use of our C # DEMO available in two projects, a it is MVSDK, this project designed to define data structures and development kit SDK API function to load; the other is the Basic project, this project is to achieve a set of preview, capture, camera settings as one of the routines. Basic engineering project by calling MVSDK code to indirectly access MVCAMSDK.DLL.

Second, initialize the SDK. After completion of the SDK is loaded, before using other interfaces, call CameraSdkInit function to initialize.
Enumerates devices. Call CameraEnumerateDevice function enumerates the device, obtain a list of currently connected PC to the camera on the device, including the list of device names (can modify their own), version number, a unique serial number, camera model and other information.

2) initialize the device. Depending on the camera device enumeration information obtained in the third step, calling CameraInit function initializes the specified camera, get the camera handle. If you need to open multiple cameras, the use of multiple camera device name CameraInit multiple calls to get more cameras handle the follow-up to the operation of the camera, the camera will need this time to get a handle of the object to specify camera operation.

3) Let SDK into the image acquisition mode. Camera Play function calls, let the camera into the working mode, and the SDK starts receiving the image from the camera.

4) capture images. SDK provides two ways to obtain image data, efficiency of these two methods are the same, the underlying use of zero-copy mechanisms to improve efficiency, you can develop your habits to select one of them.

CameraGetImageBuffer active call to get an image data. This function is used to get an SDK internal address buffer receives the image data, and header information. At the same time, this function can set the timeout, the image is not acquired within the specified time (thread is suspended), it returns a timeout.

In the third step, after the camera is initialized, calls CameraSetCallbackFunction to set a callback function. This approach is passive, only within the SDK after receiving a valid frame of image data, will call you to set the callback function to pass a frame of image data received.

Note: You can also use the above two ways to get the image, but not in CameraSetCallbackFunction set the callback function to invoke CameraGetImageBuffer get the image again, this will produce a deadlock.

5) image processing. Previous image frames acquired the original format camera output, the company most camera models, the raw Bayer output format or is YUV formats, which information will be automatically added to the header information, call CameraImageProcess to get the image processing effects such as color gain adjustment, white balance correction, saturation, LUT conversion, noise reduction, etc., and YUV format, or Bayer raw data into 24BIT bitmap format (RGB888).

6) superimposed crosshairs, auto exposure reference window, white balance and other additional content reference window (if your development, the superimposed information is not required, this step can be skipped). CameraImageOverlay call function, is set to visible crosshair reference window automatic exposure, white balance reference window will be superimposed on the input image. CameraImageOverlay input must be in
bitmap format, we recommend that you get after calling CameraImageProcess bitmap format, then call CameraImageOverlay function.

7) Save the image or display an image (if your development, on the other image processing, without the need to save the image as a file or display, you can skip this step).

8) If you need to save the image to a file, in the sixth step, the seventh or eighth step after step, calling CameraSaveImage function to save the image, SDK supports PNG, BMP, JPG and raw data in four ways. If you want to preserve the original data, you should call later on in the sixth step CameraSaveImage function; if saved as BMP, PNG, JPG format, you should call CameraSaveImage function after the seventh step; if saved as BMP, PNG, JPG format at the same time You need superimposed crosshairs and automatic exposure, white balance reference position of the window, then you can call CameraSaveImage function after the eighth step.

9) If you need to display an image, the following two ways:

10) a. According to its own development environment to achieve image display, such as the use of OpenGL, DirectDraw, Windows GDI and other ways to achieve the image appears.

11) b. Using our SDK in encapsulated display interface to display images. After the fourth step initialize the camera, calling CameraDisplayInit function to initialize the display interface, this function needs to be displayed to incoming control handle (HWND type), only for VC / C ++, VS, VB, VB.NET, Delphi, C #, etc. Use Windows GDI interface development tools, there are some limitations, but if you meet the conditions of use, we recommend you to use our packaged display interface.

12) 8) Turn the camera off before exiting the program (deinitialization, very important, if direct closure program without anti-initialize the camera, the program may have reported memory errors). When the camera is turned off, the call CameraUnInit function.

2.2 Develop of demo

For your more rapid development and use of our SDK, we prepared a number of routines for you. Characteristics of each routine is not the same, are not suitable for the development of people, you can according to their needs, choose a different routines to start. This article documents only for VC routine will be described, similar to the rest of the development of language VC routine instructions and routines.

Develop a routine located in the installation directory Demo folder, according to the development of the language barrier, into VC ++, VB6, C #, VB.NET, Delphi6 etc directory. Currently, there is provided a Basic, Advanced, MultiCamera, OCX, UserDataTest, TriggerAndStrobe, ImageFormat & Saving etc. several routines in order VC ++ based, C # / Delphi / VB6 / VB.NET and other development language can refer to VC ++ example calls SDK interface, all the features of each development
These routines are characterized as follows:

- Basic routine use SDK interface to create the camera's configuration window, similar to the DirectShow interface device properties page. Using this method, you basically do not need to develop their own UI interface even without the camera carefully to understand the relevant SDK to use interface, it can greatly save your development and debugging time, and we are the parameters of the camera the situation of binary files saved on your development machine after a good debugging parameters via software interface to save the parameters into a file that can be posted to the target machine, eliminating the need for complex programming initialization parameters work, it is strongly recommended that you use ways for development.

- Advanced routines covering most of the SDK interfaces, demonstrates the use of conditions and methods of these interface functions using VC ++ 6.0 interface to make camera configuration attributes. If Basic routines can not fully meet the needs of your design, you can refer to Advanced routines.

- MultiCamera routine demonstrates how to develop multi-camera can be connected 1-4 units of the same model or different models even different interfaces (USB2.0 USB3.0 GIGE) the camera while previewing.

- OCX routine demonstrates how ActiveX controls provided by our camera development, the same with OXC control mode, also supports multiple cameras.

- UserDataTest routine demonstrates how to write or read the camera in a custom data that is stored in the camera's internal, remain valid after power. Can be used as ID, application data, bundled data read and write.
- TriggerAndStrobe routines, demonstrates the development of external trigger and flash interface.
- ImageFormat & Saving routines, capture SDK demonstrates how to set the data format and how to save image operation.
- SnapshotOnPreview routine that demonstrates how to preview high speed in a small resolution, to capture an image at full resolution.
- ROI routines, demonstrates that when preset resolution does not meet the project requirements, how to customize the camera image resolution. For some models, the camera supports simultaneous transmission of multi-zone ROI function, the camera hardware to complete the crop, the software end SDK automatically spliced into the whole picture. After the removal of the unwanted transfer area, can effectively improve the frame rate, for example, 500-megapixel camera USB2.0 After using this technique, at 500 million pixels no compression, frame rate can be raised to 15.
- GPIO routine that demonstrates how to operate the camera comes with input and output IO.

2.2.1 Basic of demo

These routines The routine located in the installation directory Demo / VC ++ / Basic folder. After the routine runs, the interface as shown below:

- Pause button. Pause for camera work.
• Settings button. Click on the camera configuration window. The configuration window is generated by the SDK, and the very formula manufactured. The routine is located in the installation directory Demo / VC ++ / Basic folder. After the routine runs, the interface as shown below:
• Snapshot button. Use the SDK to capture post-click interface to obtain an image and save it to a local file.
• About button. Click the pop-up descriptions for the information of the routine.
• bottom status bar. From left to right shows the current preview resolution, the display frame rate, frame rate capture.

2.2.2 Advanced of demo
The routine is located in the installation directory Demo / C ++ / Advanced folder. After the routine runs, the interface as shown below:

- title bar interface, display resolution, the display frame rate and frame rate capture.
- Exposure module, set exposure related control functions.
- Resolution module, set the resolution of related control functions, including a custom resolution (ROI).
- Trigger module, set the trigger mode control function. Continue showing continuous output mode, Software means the software trigger mode.
- Crosshair module, set the crosshairs control functions.
- Snapshot module, set the capture mode control functions, while demonstrating how scheduled automatic camera.
ISP Graphic module, a collection of graphics-related control functions, including mirroring, sharpening, noise reduction and other functions.

ISP Color module, a collection of images related to color control functions, including white balance, RGB three-channel digital gain, saturation control lights.

Lut module, searching through a collection of power conversion control function, SDK supports three look-up table conversion method, which are generated by adjusting the parameters of the LUT, the LUT using the camera preset and custom LUT table.

2.2.3 MultiCamera demo (using multiple cameras)

The routine is located in the installation directory Demo / VC ++ / MultiCamera folder that demonstrates how to use the SDK to develop a single computer at the same time take 1-4 cameras and display four cameras preview images. If you need to take more of the camera, please refer to the example of the method can be extended to support more than 32 cameras simultaneously.

2.2.4 OCX of demo(ActiveX)

The routine is located in the installation directory Demo / VC ++ / OCX folder that demonstrates how to use the camera OCX control for application development. OCX way you can also use multiple cameras at the same time, cases can take 1-2 cameras presentations, need to take more cameras simultaneously, please refer to the example of the method for expansion.

2.2.5 MultiExposure of demo(A camera double exposure , dynamic range effects)

The routine is located in the installation directory Demo / VC ++ / MultiExposure folder that demonstrates how to use a camera to achieve the next two channels, different exposure value and the brightness gain continuous preview mode. Using the same way, you can extend it to four or more preview channels to obtain images of different exposure effect. The figure is 1.3 million using a USB2.0 interface color camera, this double exposure mode, each channel can be up to 10.

Typical applications: shoot metal, glass and other reflective materials, this method can be used to obtain different luminance image analysis. When this method is extended, two or more can be obtained exposure.
2.2.6 ImageFormat & Saving of demo (Gray, RGB24, RGB32 Formatting)

The routine is located in the installation directory Demo / VC ++ / ImageFormat & Saving folder, SDK demonstrates how to set the output format of image processing and image files save operation.

Currently, our SDK support 8bit, 24bit, 32bit image formats, to meet the vast majority of applications; providing RAW format, BMP 8,24 bit depth format, PNG, JPG image format to save a total of five kinds of image files. RAW format is the format in which the camera raw output may be 8,10,12 and 16bit raw data; BMP 8 bit depth of the image files you need to configure the output format is 8bit mode SDK; save BMP 24 bit depth, JPG, PNG format needs SDK output format is 24bit mode.

Typical applications: 8-bit grayscale and monochrome output format, will increase the speed of image processing; color camera output RGB32 format, 4-byte address aligned to facilitate visual library with assembly instructions for hardware acceleration.

The DEMO run after the interface is shown below.
2.2.7 **TriggerAndStrobe of demo (Control trigger signal and the flash)**

The routine is located in the installation directory Demo / VC ++ / TriggerAndStrobe folder that demonstrates how to set up the camera and flash trigger signal, it is noted that the example of a model does not support external trigger invalid.

Currently, our camera supports four trigger signals, respectively, rising edge, falling edge, high level trigger, low level trigger, if you are using a mechanical switch, it is recommended that the level triggered mode, and set to shake time to filter out the interference signal mechanical switch transition, if it is an electronic switch, there is no limit; support for automatic and semi-automatic flash signal in two ways, which is automatically generated automatic signal waveform when the camera exposure; semi-automatic mode the flash converter is to control the level of the signal level of the terminal with the program interface.

Typical applications: flash sync output, camera flash is activated only at the moment of exposure; pin level programming manual control flash lights output to control the infrared light switch (lit or normally closed); the use of multiple cameras simultaneously trigger photographs; use triggers to take pictures at a specified time, such as when the button is pressed.

Run the routine interface as shown below.
2.2.8 UserDataTest of demo (To customize the camera to read and write data)

The routine is located in the installation directory Demo / VC ++ / UserDataTest folder, the camera shows how to write custom data and to read and modify the device name of the camera function.

Modify camera name, it will directly affect the enumerated list of the camera, after the name changes, while interface to access Directshow, Twain, Halcon, Labview and other cameras, you name will be modified, cured in the camera, permanent. The main function is to facilitate the distinction simultaneously when connected to multiple cameras, modify the name of the camera after each good, do not worry about the interface on which the camera is connected, even if the replacement of the host computer will not affect the name of the camera.

Since the definition of reading and writing data, it is in order to facilitate the development of secondary, program data stored in the camera; or write data to achieve some special programs and camera bindings.

Typical applications: camera calibration data read and write; read and write data to achieve specific binding associated software and hardware.

The interface as shown in the program runs as follows:
2.2.9 SnapshotOnPreview of demo (Small resolution preview high-speed, high-resolution camera)

The routine is located in the installation directory Demo / VC ++ / SnapshotOnPreview folder, play.
Shows how the preview in continuous use capture function to fetch a specified resolution of the image and save it as a BMP file (capture channel captured images in memory, can be treated additionally displayed, in this case, captured image file and then be saved).

Resolution capture, preview resolution settings and capture images preview image acquisition and acquisition, has a separate interface function, independently of each other; the other parameters, such as image brightness, color, etc., preview and capture channels share. After setting the resolution to capture images in continuous capture preview, interior SDK automatically switching operation, after a successful capture, automatically switch back to preview resolution.

Typical applications: small resolution, high-speed preview, for example BIN or SKIP preview resolution, frame rate can be increased by four times, smooth video is good; large capture resolution, obtained full picture, high-precision processing.

The routine operating results in the following figure, the preview resolution can be set in the "Camera Settings" :
2.2.10 RawTransTest of test (Offline RAW files into BMP, JPG file)

The demo is located in the installation directory Demo / VC ++ / RawTransTest folder, play
It shows how to use the SDK to save the good camera RAW photo files, convert BMP or JPG file.

RAW file conversion, you need to provide a corresponding camera profiles
(camera configuration files in the installation directory Camera / Configs folder, the
file is saved the camera parameter information at runtime). Through different camera
configuration file, the RAW photo files can be different camera accurately reduced to
BMP, JPG files.

Typical applications: single or multi-camera high-speed camera to save RAW files
to the SSD solid state drive, the latter will be converted into RAW BMP, JPG
processing, such as Panorama and other applications.

2.2.11 LineScan of demo (With Line scan mode)

For certain types of area scan cameras, such as the MV-U130M and
MV-UB130M, (130 megapixel black and white), supports output line ROI resolution,
for example 1280X1, At this time, the area camera as a lower performance the line
scan cameras to use, the line speed of about 1700 lines per second.

The demo is located in the installation directory Demo / VC ++ / LineScan folder,
does not support the resolution of the output of the linear ROI camera, being given the
run-time routine. After this routine runs, it reads the press line image data, and spliced
into a whole sub-picture after the show, so the routine for the shooting of the
cylindrical rolling objects.
2.2.12 ROI of demo (How to customize Region of interest)

As shown above, ROI routine demonstrates a 500-megapixel camera USB2.0, when opening multi-zone ROI function at 2592X1944 image size, via USB2.0 interface, you can achieve a frame rate of 14 per second. Not involved in the transmission part is filled with black, four can be cut ROI region can be dynamically modified.

Typical applications: When the field of view need to take very large, and when dispersed into a plurality of regions, consider using this feature, you can effectively save bandwidth, lower CPU utilization, while improving image transmission frame rate.

2.2.13 GPIO of demo(Only for GPIO models)
As shown above, GPIO routine demonstrates how to operate the camera comes with the GPIO port. When the input type IO status is changed, it will automatically update the interface; when operating in IO-output interface, the camera output IO level will be appropriate to change.

2.2.14 SaveFile of demo (A preview image of each successive save to disk)

As shown above, SaveFile routine demonstrates how to save the image of the high-speed operation, through the routine, you can directly preview of each frame image is saved to disk, used for short-speed, continuous shooting. Use this routine should be noted that the hard drive performance of the system must be good enough hard disk write bandwidth must be greater than the output bandwidth of the preview image. In MV-GE130GM camera, for example, the camera can output 1280X960 images per second (1.2 million pixels, 1.2M) 60 frames increases bandwidth is 60 X 1.2 = 72MB / s, if you choose to save the image as RAW format, the disk write bandwidth of 72MB bytes will be able to meet the requirements of each frame output from the camera can be saved, if the image is selected as BMP format, the disk write bandwidth requires 72MB X 3 = 216 MB / s, all images can be saved . When bandwidth is less than the required disk write, part of the image file can not be preserved phenomenon occurs. Also, do not recommend saving at high speed, select a format other than RAW and BMP images jpg, png format, compression encoding requires a lot of time, can cause congestion resulting in dropped frames, RAW format completely without any treatment, it is necessary to convert time is minimal, BMP format conversion compared to RAW format, slightly more complex, but the processing time is usually shorter than, without compromising performance. Choose
Save RAW format, and then with RawTransTest routine, you can convert the RAW data offline BMP, jpg or png format.

### 2.3 Debugging camera parameters

#### 2.3.1 How to set the exposure time (anti-flicker, no smear, dynamic range boost)

The camera exposure time is a very important parameter in many applications, set methods have to pay attention, not according to the application, exposure time settings are divided into the following areas:

- Implement anti-strobe effect. When the visual system, when a light source is not a direct way of lighting, when you preview the video, if the camera is rolling shutter mode, it is possible to see the light and dark stripes; if the camera is a global shutter mode, it is possible before and after the emergence of two very different image brightness changes, looks like a flashing feeling. To resolve this problem, you first need to know the frequency of the light source, in general, the exchange of the illumination source is 50 Hz or 60 Hz. If the light source is 50 Hertz, you will need the exposure time is set to an integer multiple of 10 ms (10 ms minimum); if it is 60 Hz source, you need to set the exposure time to an integer multiple of 8.3333 milliseconds. In addition, you can use the automatic exposure mode that we provide, and the anti-strobe function enabled.

- Smear-free pictures. In the water-line visual detection often require dynamic camera, namely the detection of objects in motion capture is unreasonable if the exposure time is set, it is possible to cause obvious smear, resulting in blurred images. To resolve this problem, you need the exposure time is set to a small value, typically requires less than 2 milliseconds, specifically to see the speed of the movement of objects, continue to reduce the exposure time to match. Due to reduced exposure time, the image brightness will drop, need to provide a stronger external fill light, and because of the rolling shutter mode the camera will produce distortion deformation in the vertical direction, therefore, the need to use a global shutter mode the camera will be no smear capture.

- Dynamic range improvement. The so-called dynamic range of the image is included in the "darkest" to "brightest" range. Dynamic range, the more rich layers can be represented. In the actual imaging, often appear when looking at the clear, dark dark has been unable to identify; time to see the dark, raising has been exposed, is not recognized. There are currently two ways to solve this problem, first select the hardware to enhance the wide dynamic range of the camera; the second is to use manually set the exposure time, and then with a gamma value and contrast adjustment to achieve the purpose of this method is the case of software to enhance the dynamic range , the specific operation method is to first manually set the exposure time, to be able to see the brightest place, and then, slowly adjust the gamma value to low, high contrast to slow the adjustment, the process of change of the observed image, until better results.
AE reference locale. By default, auto-exposure algorithm reference area is the image in the middle 1/2 of the secondary development or can call the relevant function in the SDK software interface to modify the AE reference area, after modification, other than when the reference area when the image portion changes, auto exposure algorithm does not update the camera exposure.

### 2.3.2 How to get better image color

Most of the industrial automation detecting color information of the object is not sensitive, so you would use black and white camera, but the microscope imaging, color classification and other areas, the camera's color reproduction is particularly important, adjusting camera parameters in terms of color, the following aspects need attention.

White / regional balance. Most of the light source itself is not pure white, through the illumination, the object is likely to appear the phenomenon of color cast, for example, under a yellow light, the camera image, the image is likely to be biased yellow, this case, you need to perform white balance correction, and its role is in all kinds of lighting color temperature, the image was originally a real white areas (but by the color temperature effects have cast), reduced to pure white, that is $R = G = B$. White balance is to do the so-called regional operation, you can specify a particular image region as a reference object of analysis, by default, the white balance of the reference range is the entire sub-picture.

Manual white balance / auto white balance. White balance operation, is divided into two kinds of manual and automatic mode. Manual white balance, also known as disposable or one-click white balance refers to the user when the need for color correction, manually click the button on the software, once white balance correction, obtain the correction coefficients, without for the next White at equilibrium, always use the same color correction coefficients; automatic white balance is analyzed according to the camera images in real time, dynamic modification of the correction coefficient. In general, industrial applications mainly manual white balance mainly to ensure that each correction coefficient imaged exactly the same.

manually adjust the red, green and blue color channels gain. When the operation has completed the manual white balance, the white area is reduced to think on your computer's white, that white areas of $R$, $G$, $B$ component completely equal, but that does not mean the current color makes it look very comfortable, According to the survey, Europeans prefer warm colors of the image (yellow), while Asians prefer the image of cool colors (bluish), this time, you can then modify the $R$, $G$, $B$ three channel gain manually to improve the image colors.

saturation adjustment. Different people, rich and gaudy color preference is not the same degree, can be improved by adjusting saturation, like the bright colors, increase the saturation setpoint; like pale hue, saturation is reduced setpoint.
2.3.3 How to improve image clarity

In general, the optical clarity of the image by the pointer of the lens, the focusing, the decision algorithm software aspects. If the optical pointer below the camera lens focus or unclear requirements, the choice of how high pixel camera will not help.

- matching lens and camera. The camera parameter table, you can see the size of the camera's optics, for example, 1 / 2.5 "target surface, 2.2 X 2.2 um pixel size, based on these parameters to select the appropriate lens, put to achieve the best results, select a specific manner not described here, please refer to the network of the relevant text file.

- sharpness adjustment. You can increase the sharpness value on the software interfaces to improve the clarity of the image. Note: while improving sharpness, clarity will be significantly increased, but image noise will also be highlighted.

2.3.4 BayerRestore Selection Algorithm (Contribute to a better image contour extraction)

For industrial color camera, under normal circumstances, not the RGB data from the camera to the host-side data format, but an array called the Bayer color data format, and the format required by a certain algorithm, computer vision algorithms can be reduced to recognition of RGB color data, different reduction algorithms differ in effectiveness and performance of our SDK provides a total of A1 to A5 Fifth Bayer conversion algorithms can be selected:

- default A2 algorithm to calculate the algorithm advantages, reducing the sharpness is good, but the color of the border at the edge prone to false color.

- For color camera contour extraction, we recommend the A3, A4 algorithm, which compared two algorithms sharpness algorithm A2 slight decline, but the edges of the image will be more smooth and continuous.

2.3.5 How to reduce CPU usage

- Through the following aspects to reduce CPU usage:
  - camera in black and white, black and white color in the camera without conversion, so you can save some CPU overhead. Image processing SDK and set the format of 8bit mode, the data thus obtained is 8-bit grayscale, it can take up less memory space and memory replication.
  - use the camera with hardware ISP functions such GIGE, USB3.0 camera.
  - to adjust the exposure time, LUT curve and other parameters, are prominent features of the test object at the same time, try to make some unwanted detection of very dark or very bright, so can greatly reduce vision algorithms running time.

2.3.6 The camera parameters are saved and loaded

SDK provides four sets of parameters for the secondary development of the user or directly, to save or load camera parameters. In our software platform, the camera
parameter is a binary file is saved on disk, save path is under software installation
directory / Camera / Configs, is .Config file extension.

- **three different parameter loading method**

Access camera parameters can be divided by type, by nickname, by serial
number in three ways, according to the default model is loaded, as shown below:

<table>
<thead>
<tr>
<th>相机配置参数存档</th>
<th>保存</th>
<th>恢复默认参数</th>
</tr>
</thead>
<tbody>
<tr>
<td>分组： A B C D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>保存当前配置参数到指定文件</td>
<td>从指定文件中加载参数</td>
<td></td>
</tr>
<tr>
<td>加载方式： 按型号</td>
<td>按昵称</td>
<td>按序列号</td>
</tr>
<tr>
<td>选中参数组时自动保存</td>
<td>关闭相机时自动保存</td>
<td></td>
</tr>
</tbody>
</table>

In MV-UB130M (130 Wan black and white frame memory camera) as an example:

1. When you select the model by loading four parameters are the file name
   MV-UB130M-Group0.Config, MV-UB130M-Group1.Config, MV-UB130M-Group2.Config, MV-UB130M-Group3.Config, Do not correspond to the software interface settings A, B, C, D four groups of parameters, this parameter loading mode, all of the same type on the computer connected to the camera, all four shared set of parameters;

   When you select Load Parameters nickname by the way, the file name four parameters were MV-UB130M # 0 - Group0.Config, MV-UB130M # 0-Group1.Config, MV-UB130M # 0-Group2.Config, MV-UB130M # 0-Group3.Config, corresponding to the software interface settings a, B, C, D four groups of parameters, it is worth noting that the camera can be modified nickname, you can directly modify the software interface can also by SDK software interface modification, after the nickname modifications are permanent and effective cure in the camera, replace the computer, or after the USB slot, its nickname unchanged, modified nickname, camera parameters corresponding to the file name will change, for example, the camera was renamed Camera1, the file name that four groups of parameters are Camera1-Group0.Config, Camera1-Group1.Config, Camera1-Group2.Config, Camera1-Group3.Config, this more flexible model parameters loaded when the computer only when connected to a camera, if you do not change the device nickname, nickname with each camera model is the same, we shared the four groups of parameters, if you want to load a different table in which a set of parameters, you can individually named this station camera, making it the nickname equipment separately from other models of the same area to the camera.

   When you select by serial loading, file name, four parameters were MVUB130M-XXXXX-XXXXX-Group0.Config, MVUB130M-XXXXX-XXXXX-Group1.Config, MVUB130M-XXXXX-XXXXX-Group2.Config, MVUB130M-XXXXX-XXXXX-Group3.Config, wherein (XXXXX-XXXXX unique serial number for each camera),
so this parameter loading, the parameters of each camera are separate, independently of each other.

In practice, you can according to different needs to choose a different way to load, according to model loading to ensure that all of the same model camera using the same parameters; press nickname loading, you can have the same parameters of the same type of camera, also you can specify a single station uses independent parameters; by ordinal loaded, each camera uses a separate argument.

modify the default parameters of the camera

When the user interface to the software configuration click on the "Restore default parameters" button, the camera will reset the parameters, the parameter is reset to a default state to prevent a user without knowing the camera parameters, the modified parameters appear a variety of "abnormal" situation.

Be restored when the default parameters, the parameters will be restored to factory condition, but the parameter if you do secondary development, it may be desirable when a customer clicks on the button, you return to the default parameter settings, not the factory, here, we also provide a way to modify the default parameters, so that end customers during the recovery preset parameters (after the first installation of the camera or the program, the first turn on the camera) can be your set parameters. The method is simple: use our demo software, after the camera parameter set, saved as a separate file, the file name is the model name + "- Default.Config", to MV-UB130M (130 Wan black and white frame memory camera) to example, save it to file arguments "MV-UB130M-Default.Config", you can simply existing Config file, renamed "MV-UB130M-Default.Config", then the file in the installation directory / Camera / Configs directory. Thus, the end-user recovery during the preset parameters will be restored to the "MV-UB130M-Default.Config" parameters in the file.
3 SDK data type definition

3.1 Structure is defined

- **tSdkCameraDevInfo**
  
  **prototype:**
  
  ```
  typedef struct {
      UINT uVendorID;            // Manufacturer ID
      UINT uProductID;           // product ID
      char acVendorName[32];     // Manufacturer Names
      char acProductSeries[32];  // Products
      char acProductName[32];    // Product name
      char acFriendlyName[64];   //Initialize the device a nickname used by the interface
      char acDevFileName[32];    // Driver Name
      char acFirmwareVersion[32]; // Version
      char acSensorType[32];     // Sensor type
      char acPortType[32];       // Interface Type
  } tSdkCameraDevInfo;
  ```
  
  Description: Camera Equipment Information

- **tSdkImageResolution**
  
  **prototype:**
  
  ```
  typedef struct {
      INT     iIndex;             // Index number, [0, N] indicates the default resolution (N is the default maximum number of resolution, generally not more than 20), OXFF represents a custom resolution (ROI)
      char    acDescription[32];  // The resolution description. When only the default resolution of the information is valid. Custom resolution can ignore the message
      UINT    uBinSumMode;        // BIN (summation) mode, can not exceed the scope of uBinSumModeMask tSdkResolutionRange
  } tSdkResolutionRange;
  ```
UINT uBinAverageMode;       // BIN (averaging) mode, can not exceed the scope of uBinAverageModeMask tSdkResolutionRange
UINT uSkipMode;            // SKIP whether the size of 0 disables SKIP mode, can not exceed the scope of uSkipModeMask tSdkResolutionRange
UINT uResampleMask;       // Hardware resampling shield
INT iHOffsetFOV;          // Field of view with respect to the acquisition and the vertical offset of the upper left corner of the maximum field of view Sensor
INT iVOffsetFOV;          // Field of view with respect to the acquisition field of view Sensor maximum horizontal offset of the upper left corner
INT iWidthFOV;            // The width of the acquisition field of view
INT iHeightFOV;           // Acquisition height field of view
INT iWidth;               // The width of the final output of the camera image
INT iHeight;              // The final height of the camera images output
INT iWidthZoomHd;         // Hardware scaling width, does not require the resolution of this action, this variable is set to 0.
INT iHeightZoomHd;        // Hardware scaling heights, does not require the resolution of this action, this variable is set to 0.
INT iWidthZoomSw;         // Swscaler width, does not require the resolution of this action, this variable is set to 0.
INT iHeightZoomSw;        // Swscaler height, does not require the resolution of this action, this variable is set to 0.
} tSdkImageResolution;

Description: The camera's resolution is described

➢ tSdkMediaType

prototype:
typedef struct
{   
    INT   iIndex;        // Format Type No
    char  acDescription[32]; // Description
}
UINT iMediaType;  // Corresponding to the image coding format
} tSdkMediaType;

Description: The camera output image data format

### tSdkIspCapacity

Prototype:

typedef struct {
    BOOL bMonoSensor;  // The model indicates that the camera is a black and white camera, if the camera is in black and white, the color-related functions can not be adjusted
    BOOL bWbOnce;      // The model indicates whether the camera supports manual white balance function
    BOOL bAutoWb;      // The model indicates whether the camera supports auto white balance function
    BOOL bAutoExposure; // The model indicates whether the camera supports auto-exposure function
    BOOL bManualExposure; // The model indicates whether the camera supports manual exposure
    BOOL bAntiFlick;   // The model indicates that the camera is supported by anti-strobe function
    BOOL bDeviceIsp;  // The model indicates the camera support function hardware ISP
    BOOL bForceUseDeviceIsp:// bDeviceIsp and bForceUseDeviceIsp TRUE simultaneously, it indicates mandatory only hardware ISP, can not be canceled.
    BOOL bZoomHD;     // Is the camera hardware support for image scaling output (only shrink).
} tSdkIspCapacity;

Description: Enable the information ISP module.

### tSdkCameraCapability

Prototype:
typedef struct 
{
    tSdkTrigger  *pTriggerDesc;          // Trigger Mode
    INT             iTriggerDesc;           // The number of trigger mode, the size of the array Trigger Desc
    tSdkImageResolution   *pImageSizeDesc;// Select the default resolution
    INT             iImageSizeDesc;        // The number of default resolution, the size of the array pImageSizeDesc
    tSdkColorTemperatureDes *pClrTempDesc;// The default color temperature mode for white balance
    INT             iClrTempDesc;
    tSdkMediaType     *pMediaTypeDesc;    // The camera output image format
    INT             iMediaTypeDesc;        // The number of types of camera output image format, the size of the array pMediaType Desc.
    tSdkFrameSpeed    *pFrameSpeedDesc;   // Adjustable frame type corresponds to the common interface and super high-speed three speed settings.
    INT             iFrameSpeedDesc;      // The number of types of frames can be adjusted, the size pFrameSpeedDesc array.
    tSdkPackLength    *pPackLenDesc;      // Transmission packet length, generally used for network equipment
    INT             iPackLenDesc;         // The number of sub-transmission lengths to choose from, the size of pPackLenDesc array.
    INT             iOutputIoCounts;       // Programming can be output IO number
    INT             iInputIoCounts;        // Programming can enter the number of IO
    tSdkPresetLut  *pPresetLutDesc;       // The camera preset LUT table
    INT             iPresetLut;            // The camera preset number
    INT             iUserDataMaxLen;        // It indicates that the camera is used to save the maximum length of the user data area. 0 indicates no.
BOOL bParamInDevice; // It indicates that the device supports read and write from the device parameter set. 1 support 0 not supported.

tSdkAeAlgorithm *pAeAlmSwDesc; // AE software algorithm description
int iAeAlmSwDesc; // The number of software AE algorithm

tSdkAeAlgorithm *pAeAlmHdDesc; // AE hardware algorithm description is NULL not support hardware AE
int iAeAlmHdDesc; // AE number of hardware algorithms, 0 to not support hardware AE

tSdkBayerDecodeAlgorithm *pBayeDecAlmSwDesc; // Software Bayer RGB data is converted to algorithmic description.
int iBayeDecAlmSwDesc; // Software Bayer RGB data is converted to the number of algorithms

tSdkBayerDecodeAlgorithm *pBayeDecAlmHdDesc; // Hardware Bayer RGB data is converted to algorithmic descriptions for NULL indicates no support
int iBayeDecAlmHdDesc; // Hardware Bayer RGB data is converted to algorithmic descriptions for NULL indicates no support

/ * The adjustment range defined image parameters for dynamically build UI */

tSdkExpose sExposeDesc; // The range of values of exposure
tSdkResolutionRange sResolutionRange; // Resolution Range Description
tRgbGainRange sRgbGainRange; // Image Digital Gain Range Description
tSaturationRange sSaturationRange; // Saturation Range Description
### Description

- **Gamma Range**
  - `tGammaRange sGammaRange;`  
  - **Description:**

- **Contrast Range**
  - `tContrastRange sContrastRange;`  
  - **Description:**

- **Sharpen Range**
  - `tSharpnessRange sSharpnessRange;`  
  - **Description:**

- **ISP capability description**
  - `tSdkIspCapacity sIspCapacity;`  
  - **Description:**

### Device Description information defined integration, this information can be used to dynamically build UI.

#### `tSdkFrameHead`

**prototype:**

```c
typedef struct {
    UINT uiMediaType; // Image Format
    UINT uBytes; // Total bytes
    UINT iHeight; // Image width
    UINT iWidth; // Image height
    BOOL bIsTrigger; // is trigger
    UINT uiTimeStamp; // The frame acquisition time, 0.1 ms units
    UINT uiExpTime; // Current exposure value of the image, in microseconds
    float fAnalogGain; // Multiple analog gain of the current image
    INT iGamma; // The frame image gamma setting value
    INT iContrast; // Set the contrast value of the image frame, and only when the LUT mode dynamic parameter to generate valid, at rest mode -1
    INT iSaturation; // Saturation setpoint of the frame image for the black and white camera meaningless as 0
    float fRgain; // Red digital gain multiples of the frame image processing, for black and white camera meaningless, 1
    float fGgain; // Green gain multiple digital image processing of the frame, for black and white
```
camera meaningless, 1 float fBgain; // Blue digital gain multiples of the frame image processing, for black and white camera meaningless, 1
}tSdkFrameHead;
Note: the image header information

- **tSdkFrame**

  prototype:
  typedef struct sCameraFrame
  {
    tSdkFrameHead head; // Header
    BYTE * pBuffer; // Data Area
  }tSdkFrame;
  Description: description of image frames

### 3.2 Parameter Type Definition

- **emSdkLutMode**

  prototype:
  
  typedef enum
  {
    LUTMODE_PARAM_GEN=0, // Dynamically generated LUT table by adjusting parameters
    LUTMODE_PRESET, // Use default LUT table
    LUTMODE_USER_DEF, // User-defined LUT table
  }emSdkLutMode;
  Description: Transform the way the image look-up table

- **emSdkParamTarget**

  prototype:
  typedef enum
{  
    PARAM_ON_PC = 0, // Save to your PC
    PARAM_ON_DEVICE = 1, // Stored in the camera
}emSdkParamTarget;

Explanation: The camera parameters stored objects

- **emSdkRunMode**

  prototype:
  typedef enum
  {
    RUNMODE_PLAY=0, // Normal preview, the captured image is displayed.
    RUNMODE_PAUSE, // pause
    RUNMODE_STOP, // The camera stops working.
  }emSdkRunMode;

  Description: Control the camera's video stream

- **emSdkDisplayMode**

  prototype:
  typedef enum
  {
    DISPLAYMODE_SCALE=0, // Zoom display mode
    DISPLAYMODE_REAL //1:1 Display Mode
  }emSdkDisplayMode;

  Explanation: An internal SDK interface display mode

- **emSdkRecordMode**

  Prototype :
  typedef enum
  {
    RECORD_STOP = 0, // Stop
    RECORD_START, // Recording
    RECORD_PAUSE, // Pause
  }emSdkRecordMode;
Description: Recording status

- **emSdkMirrorDirection**
  
  Prototype:

  ```c
  typedef enum {
    MIRROR_DIRECTION_HORIZONTAL = 0, // Mirror horizontally
    MIRROR_DIRECTION_VERTICAL, // Vertical Mirror
  } emSdkMirrorDirection;
  ```

  Description: Mirror image operation

- **emSdkFrameSpeed**
  
  Prototype:

  ```c
  typedef enum {
    FRAME_SPEED_LOW = 0,       // Slow Mode
    FRAME_SPEED_NORMAL,       // Normal mode
    FRAME_SPEED_HIGH,         // High-speed mode
    FRAME_SPEED_SUPER         // Ultra-high-speed mode
  } emSdkFrameSpeed;
  ```

  Note: The camera video frame rate

- **emSdkFileType**
  
  Prototype:

  ```c
  typedef enum {
    FILE_JPG = 1, // JPG
    FILE_BMP = 2, // BMP
    FILE_RAW = 4, // RAW: Bayer camera output file format
    FILE_PNG = 8  // PNG
  } emSdkFileType;
  ```

  Description: Save the file format type

- **emSdkLightFrequency**
typedef enum
{
    LIGHT_FREQUENCY_50HZ = 0,     //50HZ
    LIGHT_FREQUENCY_60HZ,         //60HZ
}emSdkLightFrequency;

Description: Anti-strobe flash's automatic exposure

➤ emSdkParameterMode

Prototype :

typedef enum
{
    PARAM_MODE_BY_MODEL = 0, // Loading parameters from the profile according to the camera model name, such as MV-U300
    PARAM_MODE_BY_NAME,      // The apparatus nickname (tSdkCameraDevInfo.acFriendlyName) Loading parameters from the profile, such as MV-U300, the nickname can be customized
    PARAM_MODE_BY_SN,        // Loading device according to a unique serial number from the profile parameters, serial number at the factory has been written to the device, each camera has a different serial number.
    PARAM_MODE_IN_DEVICE     // Loading parameters from the solid-state memory devices. Not all models support reading and writing from the camera parameter set is determined by tSdkCameraCapability.bParamInDevice
}emSdkParameterMode;

Description: PARAM_MODE_BY_MODEL: all the same type of camera parameters file sharing ABCD four groups. modify
A camera parameter file, it will affect the whole of the same model
Loading camera parameters.

PARAM_MODE_BY_NAME: All the same camera device name, file sharing ABCD four parameters.
By default, when a computer took only a certain type of camera,
The device name is the same, and you want to be able to load a camera
Different parameter file, you can modify the device name by the way
To let the load parameters specified file.

PARAM_MODE_BY_SN: camera according to their own unique serial number to load the ABCD four parameter file, 
Number at the factory has been cured in the camera, each camera's serial number Are not the same, this way, each camera parameter files are independent.

You can use according to their environment, the flexibility to use more than a few ways to load parameters. For example, 
MV-U300, for example, you want more than one of the model of camera parameters in group 4 are shared on your computer, then
Use PARAM_MODE_BY_MODEL way; if you want a table or where a few MV-U300 can 
Use your own parameter file and the rest of the MV-U300 again using the same parameters file, then use
PARAM_MODE_BY_NAME way; if you want each MV-U300 uses a different parameter file, then
Use PARAM_MODE_BY_SN way.
Parameter file exists installation directory under \ Camera \ Configs directory in config file name suffix.

➤ emSdkParameterTeam

```cpp
typedef enum
{
    PARAMETER_TEAM_DEFAULT = 0xff,
    PARAMETER_TEAM_A = 0,
    PARAMETER_TEAM_B = 1,
    PARAMETER_TEAM_C = 2,
    PARAMETER_TEAM_D = 3
}emSdkParameterTeam;
```

Explanation: The configuration parameters of the camera, is divided into A, B, C, D 4 group to be saved.
emSdkPropSheetMsg

prototype:
typedef enum
{
    SHEET_MSG_LOAD_PARAM_DEFAULT = 0, // After the parameters are reset to the default, triggering the message
    SHEET_MSG_LOAD_PARAM_GROUP, // Loads the specified parameter set, triggering the message
    SHEET_MSG_LOAD_PARAM_FROMFILE, // After loading parameters from the specified file, the message is triggered
    SHEET_MSG_SAVE_PARAM_GROUP // The current parameter set is saved, the message is triggered
} emSdkPropSheetMsg;

Description: Type SDK callback messages generated by the camera's configuration page

tSdkWhiteBalanceDes

prototype:
typedef struct
{
    INT iIndex; // Mode index number
    char acDescription[32]; // Description
} tSdkWhiteBalanceDes;

Description: The camera white balance mode description

tSdkFrameSpeed

prototype:
typedef struct
{
    INT iIndex; // Frame rate index number
    char acDescription[32]; // Description information
} tSdkFrameSpeed;

Description: The Description information the camera frame rate
**tSdkExpose**

Prototype:

typedef struct
{
    UINT     uiTargetMin;     // AE target minimum brightness
    UINT     uiTargetMax;     // AE target maximum brightness
    UINT     uiAnalogGainMin; // Minimum analog gain
    UINT     uiAnalogGainMax; // The maximum analog gain
    float    fAnalogGainStep; // Analog gain steps.
    UINT     uiExposeTimeMin; // In manual mode, the minimum number of line exposure
    UINT     uiExposeTimeMax; // In manual mode, the maximum number of rows impressions
} tSdkExpose;

Note: The camera exposure function scope definition

**tSdkTrigger**

Prototype:

typedef struct
{
    INT     iIndex;         // Mode index number
    char    acDescription[32];  // This mode of Description information
} tSdkTrigger;

Explanation: The trigger mode Description

**tSdkPackLength**

Prototype:

typedef struct
{
    INT     iIndex;         // Subcontracting size of the index number
    char    acDescription[32];  // Subcontracting size of the index number
} tSdkPackLength;

说明：传输分包大小描述
- **tSdkHardwareIO**
  Prototype:
  ```c
  typedef struct {
    INT    iIndex;           // IO number
    BOOL   bOutPut;          // IO properties, input or output
  } tSdkHardwareIO;
  ```
  Description: The camera programming IO description

- **tSdkPresetLut**
  Prototype:
  ```c
  typedef struct {
    INT    iIndex;           // Numbering
    char acDescription[32];  // Description information
  } tSdkPresetLut;
  ```
  Explanation: The table describes the default LUT

- **tSdkFrameStatistic**
  Prototype:
  ```c
  typedef struct {
    INT     iTotal;          // The current total number of frames acquired
    INT     iCapture;        // The number of valid frames of the current collection
    INT     iLost;           // The current number of dropped frames
  } tSdkFrameStatistic;
  ```
  Description: Frame rate statistics

- **tGammaRange**
  Prototype:
typedef struct
{
    INT iMin;  // Min
    INT iMax;  // Max
} tGammaRange;

Description: The gamma setting range

➢ **tContrastRange**

Prototype:

```c
typedef struct
{
    INT iMin;  // Min
    INT iMax;  // Max
} tContrastRange;
```

Description: The contrast setting range

➢ **tRgbGainRange**

Prototype:

```c
typedef struct
{
    INT iRGainMin;   // The minimum red gain
    INT iRGainMax;   // Red maximum gain
    INT iGGainMin;   // Minimum green gain
    INT iGGainMax;   // Green maximum gain
    INT iBGainMin;   // Blue minimum gain
    INT iBGainMax;   // Blue maximum gain
} tRgbGainRange;
```

Description: RGB three-channel digital gain setting range

➢ **tSaturationRange**

Prototype:

```c
typedef struct
{
    INT iMin;  // Min
```
INT iMax;          // Max
} tSaturationRange;

Description: The saturation limits set

➢ tSdkResolutionRange

Prototype :

typedef struct
{
    INT iHeightMax;              // The maximum height of the image
    INT iHeightMin;              // The minimum height of the image
    INT iWidthMax;               // The maximum width of the image
    INT iWidthMin;               // The minimum width of the image
    UINT uSkipModeMask;         // SKIP mode shield, is 0, that does not
                                  support SKIP. bit0 1, expressed support for the SKIP 2x2; bit1 1, expressed
                                  support for the SKIP 3x3 ....
    UINT uBinSumModeMask;       // BIN (Sum) mode shield is 0, that is not
                                  supported by BIN. bit0 1, expressed support for BIN 2x2; bit1 1, expressed
                                  support for BIN 3x3 ....
    UINT uBinAverageModeMask;   // BIN (averaging) mode shield is 0, that is
                                  not supported by BIN. bit0 1, expressed support for BIN 2x2; bit1 1, expressed
                                  support for BIN 3x3 ....
    UINT uResampleMask;         // Hardware resampling shield
} tSdkResolutionRange;

Description: The camera's resolution setting range

➢ emSdkRefWinType

Prototype :

typedef enum
{
    REF_WIN_AUTO_EXPOSURE = 0,
    REF_WIN_WHITE_BALANCE,
} emSdkRefWinType;

Description: Visual Reference window type
- **tSharpnessRange**

  Description: Sharpen setting range

- **SKIPMASK_xxx Macro definition**

  Description: Hardware SKIP, BIN, resampling 2X2
```c
#define MASK_7X7_SW   (1<<21)
#define MASK_8X8_SW   (1<<22)
#define MASK_9X9_SW   (1<<23)
#define MASK_10X10_SW (1<<24)
#define MASK_11X11_SW (1<<25)
#define MASK_12X12_SW (1<<26)
#define MASK_13X13_SW (1<<27)
#define MASK_14X14_SW (1<<28)
#define MASK_15X15_SW (1<<29)
#define MASK_16X16_SW (1<<30)
#define MASK_17X17_SW (1<<31)
```

Description: tSdk Resolution Range struct SKIP, BIN, mask value RESAMPLE mode

- **CAMERA_MEDIA_TYPE_MONOxx Macro definition**

  Prototyp :

  ```c
  #define CAMERA_MEDIA_TYPE_MONO1P
  (CAMERA_MEDIA_TYPE_MONO      |
   CAMERA_MEDIA_TYPE_OCCUPY1BIT | 0x0037)

  #define CAMERA_MEDIA_TYPE_MONO1P
  (CAMERA_MEDIA_TYPE_MONO      |
   CAMERA_MEDIA_TYPE_OCCUPY2BIT | 0x0038)

  #define CAMERA_MEDIA_TYPE_MONO1P
  (CAMERA_MEDIA_TYPE_MONO      |
   CAMERA_MEDIA_TYPE_OCCUPY4BIT | 0x0039)

  #define CAMERA_MEDIA_TYPE_MONO1P
  (CAMERA_MEDIA_TYPE_MONO      |
   CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0001)
  ```
#define CAMERA_MEDIA_TYPE_MONO8S
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0002)

#define CAMERA_MEDIA_TYPE_MONO10
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0003)

#define CAMERA_MEDIA_TYPE_MONO10_PACKED
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0004)

#define CAMERA_MEDIA_TYPE_MONO12
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0005)

#define CAMERA_MEDIA_TYPE_MONO12_PACKED
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0006)

#define CAMERA_MEDIA_TYPE_MONO14
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0025)

#define CAMERA_MEDIA_TYPE_MONO16
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0007)

Description: Image Format Definition

- **CAMERA_MEDIA_TYPE_BAYxxx Macro definition**

 Prototype:

#define CAMERA_MEDIA_TYPE_BAYGR8
(CAMERA_MEDIA_TYPE_MONO |
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0008)
#define CAMERA_MEDIA_TYPE_BAYRG8
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x0009)

#define CAMERA_MEDIA_TYPE_BAYGB8
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000A)

#define CAMERA_MEDIA_TYPE_BAYBG8
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY8BIT | 0x000B)

#define CAMERA_MEDIA_TYPE_BAYGR10_MIPI
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA.MEDIA_TYPE_OCCUPY10BIT | 0x0026)

#define CAMERA_MEDIA_TYPE_BAYRG10_MIPI
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0027)

#define CAMERA_MEDIA_TYPE_BAYGB10_MIPI
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0028)

#define CAMERA_MEDIA_TYPE_BAYBG10_MIPI
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY10BIT | 0x0029)

#define CAMERA_MEDIA_TYPE_BAYGR10
(CAMERA_MEDIA_TYPE_MONO | 
CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000C)
#define CAMERA_MEDIA_TYPE_BAYRG10
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000D)

#define CAMERA_MEDIA_TYPE_BAYGB10
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000E)

#define CAMERA_MEDIA_TYPE_BAYBG10
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x000F)

#define CAMERA_MEDIA_TYPE_BAYGR12
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0010)

#define CAMERA_MEDIA_TYPE_BAYRG12
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0011)

#define CAMERA_MEDIA_TYPE_BAYGB12
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0012)

#define CAMERA_MEDIA_TYPE_BAYBG12
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0013)

#define CAMERA_MEDIA_TYPE_BAYGR10_PACKED
(CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0026)
#define CAMERA_MEDIA_TYPE_BAYRG10_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0027)

#define CAMERA_MEDIA_TYPE_BAYGB10_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0028)

#define CAMERA_MEDIA_TYPE_BAYBG10_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0029)

#define CAMERA_MEDIA_TYPE_BAYGR12_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002A)

#define CAMERA_MEDIA_TYPE_BAYRG12_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002B)

#define CAMERA_MEDIA_TYPE_BAYGB12_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002C)

#define CAMERA_MEDIA_TYPE_BAYBG12_PACKED (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x002D)

#define CAMERA_MEDIA_TYPE_BAYGR16 (CAMERA_MEDIA_TYPE_MONO | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x002E)
Description: Bayer format code definitions

CAMERA_MEDIA_TYPE_RGB8xx Macro definition

Prototype:

```c
#define CAMERA_MEDIA_TYPE_RGB8
(CAMERA_MEDIA_TYPE_COLOR | 
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0014)
```

```c
#define CAMERA_MEDIA_TYPE_BGR8
(CAMERA_MEDIA_TYPE_COLOR | 
CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0015)
```

```c
#define CAMERA_MEDIA_TYPE_RGBA8
(CAMERA_MEDIA_TYPE_COLOR | 
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0016)
```

```c
#define CAMERA_MEDIA_TYPE_BGRA8
(CAMERA_MEDIA_TYPE_COLOR | 
CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x0017)
```

```c
#define CAMERA_MEDIA_TYPE_RGB10
(CAMERA_MEDIA_TYPE_COLOR | 
CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0018)
```
#define CAMERA_MEDIA_TYPE_BGR10 (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0019)

#define CAMERA_MEDIA_TYPE_RGB12 (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001A)

#define CAMERA_MEDIA_TYPE_BGR12 (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x001B)

#define CAMERA_MEDIA_TYPE_RGB16 (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0033)

#define CAMERA_MEDIA_TYPE_RGB10V1_PACKED (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001C)

#define CAMERA_MEDIA_TYPE_RGB10P32 (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY32BIT | 0x001D)

#define CAMERA_MEDIA_TYPE_RGB12V1_PACKED (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY36BIT | 0X0034)

#define CAMERA_MEDIA_TYPE_RGB565P (CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0035)
```c
#define CAMERA_MEDIA_TYPE_BGR565P
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0X0036)

Description: RGB format code definitions

➤ CAMERA_MEDIA_TYPE_YUV411_8 xxxx Macro definition

Prototype:
```c
#define CAMERA_MEDIA_TYPE_YUV411_8_UYYVYY
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x001E)

#define CAMERA_MEDIA_TYPE_YUV422_8_UYVY
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x001F)

#define CAMERA_MEDIA_TYPE_YUV422_8
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0032)

#define CAMERA_MEDIA_TYPE_YUV8_UYV
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0020)

#define CAMERA_MEDIA_TYPE_YCBCR8_CBYCR
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003A)

#define CAMERA_MEDIA_TYPE_YCBCR422_8
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003B)

#define CAMERA_MEDIA_TYPE_YCBCR422_8_CBYCRY
   (CAMERA_MEDIA_TYPE_COLOR | 
   CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0043)


#define CAMERA_MEDIA_TYPE_YCBCR411_8_CBYYCRYY
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003C)

#define CAMERA_MEDIA_TYPE_YCBCR601_8_CBYCR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x003D)

#define CAMERA_MEDIA_TYPE_YCBCR601_422_8
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x003E)

#define CAMERA_MEDIA_TYPE_YCBCR601_422_8_CBYCRY
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0044)

#define CAMERA_MEDIA_TYPE_YCBCR601_411_8_CBYYCRYY
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x003F)

#define CAMERA_MEDIA_TYPE_YCBCR709_8_CBYCR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0040)

#define CAMERA_MEDIA_TYPE_YCBCR709_422_8
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0041)

#define CAMERA_MEDIA_TYPE_YCBCR709_422_8_CBYCRY
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY16BIT | 0x0045)
#define CAMERA_MEDIA_TYPE_YCBCR709_411_8_CBYYCRYY
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY12BIT | 0x0042)

Description: YUV format code definitions.

### CAMERA_MEDIA_TYPE_RGBXX_PLANAR Macro definition

Prototype:

```c
#define CAMERA_MEDIA_TYPE_RGB8_PLANAR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY24BIT | 0x0021)
```

```c
#define CAMERA_MEDIA_TYPE_RGB10_PLANAR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0022)
```

```c
#define CAMERA_MEDIA_TYPE_RGB12_PLANAR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0023)
```

```c
#define CAMERA_MEDIA_TYPE_RGB16_PLANAR
(CAMERA_MEDIA_TYPE_COLOR | CAMERA_MEDIA_TYPE_OCCUPY48BIT | 0x0024)
```

Description: RGB planar format code definitions.

### 3.3 The return value is defined interfaces (error code)

```c
#define CAMERA_STATUS_SUCCESS 0
// Successful operation
#define CAMERA_STATUS_FAILED -1
// Operation failed
#define CAMERA_STATUS_INTERNAL_ERROR -2
// Internal error
#define CAMERA_STATUS_UNKNOWN -3
```
// Unknown mistake
#define CAMERA_STATUS_NOT_SUPPORTED -4
// This feature is not supported
#define CAMERA_STATUS_NOT_INITIALIZED -5
// Initialization is not completed
#define CAMERA_STATUS_PARAMETER_INVALID -6
// Invalid argument
#define CAMERA_STATUS_PARAMETER_OUT_OF_BOUND -7
// Parameter out of range
#define CAMERA_STATUS_UNENABLED -8
// Not enabled
#define CAMERA_STATUS_USER_CANCEL -9
// User manually canceled, such as roi panel click Cancel to return
#define CAMERA_STATUS_PATH_NOT_FOUND -10
// The registry is not found in the corresponding path
#define CAMERA_STATUS_SIZE_DISMATCH -11
// Obtain image data length and defined size mismatch
#define CAMERA_STATUS_TIME_OUT -12
// Time-out error
#define CAMERA_STATUS_IO_ERROR -13
// Hardware IO error
#define CAMERA_STATUS_COMM_ERROR -14
// Communication error
#define CAMERA_STATUS_BUS_ERROR -15
// Bus Error
#define CAMERA_STATUS_NO_DEVICE_FOUND -16
// Total found no equipment
#define CAMERA_STATUS_NO_LOGIC_DEVICE_FOUND -17
// Logic devices found
#define CAMERA_STATUS_DEVICE_IS_OPENED -18
// The device has been opened
#define CAMERA_STATUS_DEVICE_IS_CLOSED -19
// Equipment has been closed
#define CAMERA_STATUS_DEVICE_VEDIO_CLOSED -20
// Without opening the device video, call recording related function, if the video
camera is not open, then return back to the error.
#define CAMERA_STATUS_NO_MEMORY -21
// There is not enough system memory
#define CAMERA_STATUS_FILE_CREATE_FAILED -22
// Create a file failed
#define CAMERA_STATUS_FILE_INVALID -23
// Invalid file format
#define CAMERA_STATUS_WRITE_PROTECTED -24
// Write protection, can not be written
#define CAMERA_STATUS_GRAB_FAILED -25
// Data acquisition failure
#define CAMERA_STATUS_LOST_DATA -26
// Data is lost, incomplete
#define CAMERA_STATUS_EOF_ERROR -27
// No frame is received terminator
#define CAMERA_STATUS_BUSY -28
// Busy (the previous operation is still in progress), the operation can not be
performed
#define CAMERA_STATUS_WAIT -29
// Wait (operating condition is not satisfied), you can try again
#define CAMERA_STATUS_IN_PROCESS -30
// Progress has been operated
#define CAMERA_STATUS_IIC_ERROR -31
// IIC transmission errors
#define CAMERA_STATUS_SPI_ERROR -32
// SPI transmission errors
#define CAMERA_STATUS_USB_CONTROL_ERROR -33
// USB control transmission errors
#define CAMERA_STATUS_USB_BULK_ERROR -34
// USB BULK transfer error
#define CAMERA_STATUS_SOCKET_INIT_ERROR -35
// Network transmission kit failed to initialize
#define CAMERA_STATUS_GIGE_FILTER_INIT_ERROR -36
Network Camera kernel filter driver initialization fails, check whether the correct driver installed or reinstalled.

#define CAMERA_STATUS_NET_SEND_ERROR -37
// Network data transmission error

#define CAMERA_STATUS_DEVICE_LOST -38
// Lost connection with the network camera, the heartbeat detection timeout.

#define CAMERA_STATUS_DATA_RECV_LESS -39
// The number of bytes received less than requested.

#define CAMERA_STATUS_FUNCTION_LOAD_FAILED -40
// Loading from a file fails

#define CAMERA_STATUS_CRITICAL_FILE_LOST -41
// Necessary to run the program file is missing.

#define CAMERA_STATUS_SENSOR_ID_DISMATCH -42
// Firmware, and it does not match, because downloaded the wrong firmware.

#define CAMERA_STATUS_OUT_OF_RANGE -43
// Parameter is out of range.

#define CAMERA_STATUS_REGISTRY_ERROR -44
// Setup Registration error. Please install the application or run the installation directory Setup/Installer.exe

#define CAMERA_STATUS_ACCESS_DENY -45
// No Access. When you specify the camera is already in use by another program, and then apply for access to the camera, it will return to the state. (A camera can not be multiple programs simultaneously access)

#define CAMERA_STATUS_CAMERA_NEED_RESET -46
// Indicates that the camera needs to reset to normal use, then let the camera power restart or reboot the operating system, it can be used normally.

#define CAMERA_STATUS_ISP_MOUDLE_NOT_INITIALIZED -47
// ISP module is not initialized

#define CAMERA_STATUS_ISP_DATA_CRC_ERROR -48
// Data validation errors

#define CAMERA_STATUS_MV_TEST_FAILED -49
// Data test failed

#define CAMERA_AIA_PACKET_RESEND 0x0100
// The frame needs to be retransmitted

#define CAMERA_AIA_NOT_IMPLEMENTED 0x8001
// Device does not support the command

#define CAMERA_AIA_INVALID_PARAMETER 0x8002
// Illegal command parameter

#define CAMERA_AIA_INVALID_ADDRESS 0x8003
// Address inaccessible

#define CAMERA_AIA_WRITE_PROTECT 0x8004
// Object Access cannot write

#define CAMERA_AIA_BAD_ALIGNMENT 0x8005
// Access address is not aligned in accordance with the requirements of

#define CAMERA_AIA_ACCESS_DENIED 0x8006
// No access

#define CAMERA_AIA_BUSY 0x8007
// Command is being processed

#define CAMERA_AIA_DEPRECATED 0x8008
// 0x8008-0x800B 0x800F The directive has been abandoned

#define CAMERA_AIA_PACKET_UNAVAILABLE 0x800C
// Invalid package

#define CAMERA_AIA_DATA_OVERRUN 0x800D
// Data overflow, usually received more data than necessary

#define CAMERA_AIA_INVALID_HEADER 0x800E
// Does not match the packet header in certain regions and protocol
#define CAMERA_AIA_PACKET_NOT_YET_AVAILABLE 0x8010
// Subcontracting image data is not ready, multi-shot mode for application access timeout

#define CAMERA_AIA_PACKET_AND_PREV_REMOVED_FROM_MEMORY 0x8011
// You need access to subcontracting does not exist. Used for the retransmission data not already in the buffer

#define CAMERA_AIA_PACKET_REMOVED_FROM_MEMORY 0x8012
// CAMERA_AIA_PACKET_AND_PREV_REMOVED_FROM_MEMORY

#define CAMERA_AIA_NO_REF_TIME 0x0813
// No reference clock source. When used for time synchronization command execution

#define CAMERA_AIA_PACKET_TEMPORARILY_UNAVAILABLE x0814
// Since the channel bandwidth issues, current subcontracting is temporarily unavailable, we need to access later

#define CAMERA_AIA_OVERFLOW 0x0815
// Device-side data overflow, usually queue is full

#define CAMERA_AIA_ACTION_LATE x0816
// Effective command over a specified time

#define CAMERA_AIA_ERROR 0x8FFF
// error
4 SDK interface function description (C / C ++ VB Delphi C # General)

4.1 This section describes the SDK prototype and description of each interface functions for C / C ++, VB, Delphi, this manual can be used directly in the name of the function call, C # is used in packaged adapter class MVSDK.MvApi call. For example, MVSDK.MvApi.CameraSdkInit (1) is initialized, call the function of the other interface, only you need to add a prefix to MVSDK.MvApi.

4.2 CameraSdkInit

Prototype :

```c
MVSDK_API CameraSdkStatus CameraSdkInit
(
    int     iLanguageSel
);
```

Function: SDK initialization

Parameter Description: iLanguageSel: Select Interface Language 1: Chinese, 0: English. The other does not support.

Return value: on success, returns CAMERA_STATUS_SUCCESS = 0; otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: Before calling any other SDK interface, you must call this interface to initialize. This function is only run during the entire process needs to be called once.

Example :

```c
/* Initialize the SDK for the Chinese Interface*/
CameraSdkInit(1);
```
4.3 CameraInit

Prototype:

```c
MVSDK_API CameraSdkStatus CameraInit
(
    tSdkCameraDevInfo* pCameraInfo,
    int emParamLoadMode,
    int emTeam,
    CameraHandle* pCameraHandle
);
```

Function: initialize camera.

Parameter Description:

- pCameraInfo: The camera enumeration description information obtained from CameraEnumerateDevice.
- emParamLoadMode: loading parameters used to initialize the camera. -1 Indicates that the parameter loading the last exit. The remaining value of the reference emSdkParameterMode type definition.
- emTeam: initialization parameter set used. -1 Indicates the last exit load parameter set.
- * PCameraHandle: After the camera handle index, the initialization is successful, the indicator

Returns a valid handle to the camera, calling other cameras

When the relevant user interface, you need to pass the handle, the main

Used to distinguish between multiple cameras.

Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h

Note: After initialization is successful, the camera can call any other relevant operation interface.

Example:

```c
/* Initialize the camera, load parameter set last exit */
tSdkCameraDevInfo sCameraList[10];
```


INT iCameraNums = 10;

INT status;

if (CameraEnumerateDevice(sCameraList, &iCameraNums) == CAMERA_STATUS_SUCCESS && iCameraNums > 0)
{
    status = CameraInit(&sCameraList[0], -1, -1, &m_hCamera);

    if (status == CAMERA_STATUS_SUCCESS)
    {
        // Camera successful initialization
    }
}

4.4 CameraInitEx

Prototype :

MVSDK_API CameraSdkStatus CameraInitEx(

    int iDeviceIndex,
    int iParamLoadMode,
    int emTeam,
    CameraHandle* pCameraHandle
);

Function: initialize camera.

Parameter Description:

iDeviceIndex      The camera's index number, starting from 0.
iParamLoadMode    Loading parameters used when the camera is initialized. -1
                 Indicates that the parameter loading the last exit.

                 Is represented by the models loaded
                PARAM_MODE_BY_MODEL
                 It is represented by sequence number to load
                PARAM_MODE_BY_SN
                 It is represented by the nickname loaded
                PARAM_MODE_BY_NAME
See details in open CameraDefine.hemSdkParameterMode definition.

emTeam Initialization parameter set to use when. -1 Indicates the last exit load parameter set.
Handle pointer pCameraHandle camera, after initialization is successful, the pointer
The camera returns a valid handle when calling other camera-related user interface, you need to pass the handle, it is mainly used to distinguish between multiple cameras.

Return value: on success, returns CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: After initialization is successful, the camera can call any other relevant operation interface.

Example:
/* Initialize the camera, load parameter set last exit*/
tSdkCameraDevInfo sCameraList[10];
INT iCameraNums = 10;
INT status;
if (CameraEnumerateDeviceEx(sCameraList,&iCameraNums) > 0)
{
    status = CameraInitEx(0,-1,-1,&m_hCamera);
    if (status == CAMERA_STATUS_SUCCESS)
    {
        // Camera successful initialization
    }
}

4.5 CameraInitEx2

Prototype:
MVSDK_API CameraSdkStatus
CameraInitEx2
(
    char*             CameraName,
    CameraHandle*   pCameraHandle
);

**Function Description**: Camera initialization.

**Parameter Description**:
- CameraName: Camera name, a string of 0 characters at the end.
- pCameraHandle: Handle pointer camera, after initialization is successful, the pointer

Returns a valid handle to the camera, when you call other camera-related user interface, you need to pass the handle, it is mainly used to distinguish between multiple cameras.

**Return value**: on success, returns CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: After initialization is successful, the camera can call any other relevant operation interface.

**Example**:
```c
/* Initialization named Camera1 camera requires prior use our software to initialize the camera name to Camera1 success*/
tSdkCameraDevInfo   sCameraList[10];
INT                 iCameraNums = 10;
INT                 status;
if ((iCameraNums == CameraEnumerateDeviceEx(sCameraList,&iCameraNums) )> 0)
{
    status = CameraInitEx2("Camera1",-1,-1,&m_hCamera);
    if (status == CAMERA_STATUS_SUCCESS)
    {
```
4.6 CameraDisplayInit

Prototype:

```c
MVSDK_API CameraSdkStatus CameraDisplayInit(
    CameraHandle hCamera,
    HWND hWndDisplay
);
```

Function: Initializes SDK internal display module

Parameter Description:
- hCamera: camera handle, obtained by the CameraInit function.
- hWndDisplay: Display the window handle, usually window m_hWnd members.

Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

Note: Before calling CameraDisplayRGB24 must first call the function to initialize. If you are in the secondary development, using its own way an image is displayed (do not call CameraDisplayRGB24), you do not need to call this function.

Example:

```c
/* Initialize the display interface, display interface encapsulated using the SDK */
CameraDisplayInit(m_hCamera, m_cPreview.GetSafeHwnd())
```

4.7 CameraSetDisplayMode

Prototype:

```c
MVSDK_API CameraSdkStatus
```
CameraSetDisplayMode
(
    CameraHandle hCamera,
    int mode
);

Function: Setting mode display.
Parameter Description:
hCamera: camera handle, obtained by the CameraInit function.
mode: Display mode, DISPLAYMODE_SCALE or DISPLAYMODE_REAL, see in particular the definition of CameraDefine.h in emSdkDisplayMode
Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.
Note: You must initialize CameraDisplayInit been called before calling this function.
Example:

4.8 CameraSetDisplayOffset

Prototype:
MVSDK_API CameraSdkStatus
CameraSetDisplayOffset
(
    CameraHandle hCamera,
    int iOffsetX,
    int iOffsetY
);

Function: start offset value is set to be displayed.
Parameter Description:
hCamera: camera handle, obtained by the CameraInit function.
iOffsetX: Offset of X coordinate.
iOffsetY: offset Y coordinates.

Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: when DISPLAYMODE_REAL valid only when the display mode. Such as the display size of the control is 320X240, and the size of the image is 640X480, then when iOffsetX = 160, iOffsetY = 120 is displayed when the area is the center of the image 320X240 position. You have been called CameraDisplayInit initialized before calling this function.

Example:

4.9 CameraSetCallbackFunction

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetCallbackFunction
(
    CameraHandle hCamera,
    CAMERA_SNAP_PROC pCallbackFunction,
    PVOID pContext,
    CAMERA_SNAP_PROC *pCallbackOld
);
```

Function: Set the image captured by the callback function.

Parameter Description:

hCamera: camera handle, obtained by the CameraInit function.
pCallbackFunction: callback function pointer.
pContext: additional argument to the callback function when the callback function is called the additional parameters are passed, it can be NULL. Carry additional information used for multiple cameras.
* PCallbackOld: to save the current callback function. It can be NULL.

Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**note:**

Example:

```c
/* */
CameraSetCallbackFunction(m_hCamera,GrabImageCallback,(PVOID)this,NUL L)
```

### 4.10 CameraUnInit

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraUnInit
(  
    CameraHandle hCamera
);
```

**Function:** Anti-initialize the camera. Release resources.

**Parameter Description:** hCamera camera handle, obtained by the CameraInit function.

**Return value:** on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```c
/* Anti-initialize the camera */
CameraUnInit(m_hCamera)
```
4.11 CameraPlay

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraPlay
(
    CameraHandle hCamera
);
```

**Function:** Let the SDK in work mode, begin to receive images from the camera to send.

**Parameter Description:** hCamera: camera handle, obtained by the CameraInit function.

**Return value:** on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

**Note:** If the current mode of the camera is triggered, you need a trigger frame is received later will be updated image.

**Example:**

```c
/* The camera began collecting images */
CameraPlay(m_hCamera);
```

4.12 CameraPause

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraPause
(
    CameraHandle hCamera
);
```
**Function:** Let SDK into suspend mode, does not receive image data from the camera, but the camera will send a command to make the output pause, release of transmission bandwidth.

**Parameter Description:** `hCamera`: camera handle, obtained by the CameraInit function.

**Return value:** on success, returns `CAMERA_STATUS_SUCCESS` (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

**Note:** Pause mode, you can configure the parameters of the camera, with immediate effect.

**Example:**
```c
/* The camera began collecting images */
CameraPause (m_hCamera);
```

### 4.13 CameraStop

**Prototype:**
```c
MVSDK_API CameraSdkStatus
CameraStop
(
    CameraHandle hCamera
);
```

**Function:** Let SDK brought to a standstill, typically called when uninitialized.

**Parameter Description:** `hCamera`: camera handle, obtained by the CameraInit function.

**Return value:** on success, returns `CAMERA_STATUS_SUCCESS` (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

**Note:** This function is called, the camera parameters can no longer be configured.

**Example:**
/* The camera began collecting images */
CameraStop (m_hCamera);

4.14 CameraCreateSettingPage

Prototype:

```
MVSDK_API CameraSdkStatus
CameraCreateSettingPage
(
    CameraHandle hCamera,
    HWND hParent,
    Char *pWinText,
    CAMERA_PAGE_MSG_PROC pCallbackFunc,
    PVOID pCallbackCtx,
    UINT uReserved
);
```

Function: Create the properties of the camera configuration window.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **hParent**: Handle the main application window. It can be NULL.
- ***pWinText**: String pointer, window's title bar.
- **pCallbackFunc**: When the callback function message window when the corresponding event occurs, pCallbackFunc points will be called, for example, to switch parameters like operation, when pCallbackFunc is called back at the entrance of the parameters specified in the message type. This makes it easy to synchronize between the interface and the UI we generate your own development. This parameter can be NULL.
- **pCallbackCtx**: Additional argument to the callback function. It can be NULL. Callback Ctx will be called back when
pCallbackFunc, as one of the parameters passed. You can use this parameter to do some flexible judgment.

uReserved : Set aside. It must be set to 0.

Return value: on success, returns CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: this function is called, the internal SDK will help you create a good camera configuration window, eliminating the need for you to re-develop the interface of camera time. It is strongly recommended that you use this function to let the SDK created the configuration window for you.

Example :

```

```

4.15 CameraCustomizeResolution

Prototype :

MVSDK_API CameraSdkStatus
CameraCustomizeResolution
(

    CameraHandle hCamera,
    tSdkImageResolution *pImageCustom
);

Function: Open a custom resolution panel, and by visual way to configure a custom resolution.

Parameter Description:
hCamera: camera handle, obtained by the CameraInit function. Camera name
*pImageCustom: pointer to return a custom resolution.
Return value: on success, returns CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```c
/* User-defined resolution */
CameraCustomizeResolution(m_hCamera,&sImageSize)
```

4.16 CameraShowSettingPage

Prototype:

```c
MVSDK_API CameraSdkStatus CameraShowSettingPage(
    CameraHandle hCamera,
    BOOL bShow
);
```

**Function:** Set the camera properties configuration window display state.

**Parameter Description:**

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `Show`: TRUE, display. FALSE, hidden.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:** You must first call CameraCreateSettingPage camera properties configuration window is successfully created, you can call this function for display.

Example:

```c

```
4.17 CameraSetActiveSettingSubPage

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetActiveSettingSubPage
(
    CameraHandle hCamera,
    INT index
);
```

**Function:** Set the camera configuration startup page window.

**Parameter Description:**
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `index`: Index chart page.

**Return Value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

**Note:** multiple sub-pages constituting the camera configuration window, this function can be set for the start of the current state which sub-page is displayed in the forefront.

**Example:**

4.18 CameraGetInformation

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetInformation
(
    CameraHandle hCamera,
    char **pbuffer
);
```
**Function Description** : Get the description of the camera

**Parameter Description** :

- **hCamera** : Camera handle, obtained by the CameraInit function.
- **pbuffer** : Pointing the camera Description information pointer to a pointer.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example** :

```

```

### 4.19 CameraImageProcess

**Prototype** :

```c
MVSDK_API CameraSdkStatus CameraImageProcess (
    CameraHandle hCamera,
    BYTE    *pbyIn,
    BYTE    *pbyOut,
    tSdkFrameHead    *pFrInfo
);
```

**Function Description** : Camera raw image data obtained is processed superimposed saturation, gain and color correction, noise reduction and other treatment effect, and finally get the image data RGB888 format.

**Parameter Description**:

- **hCamera** : Camera handle, obtained by the CameraInit function.
- **pbyIn** : The input image data buffer address and can not be NULL.
*pbyOut : After processing the image output buffer address, and can not be NULL.
*pFrInfo : After the header information of the input image processing is complete, the header information in the image format uiMediaType will change.

return value: When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example :

/* */
CameraImageProcess(m_hCamera, pbyBuffer, m_pbImgBuffer, &tFrameHead);

4.20 CameraImageProcessEx

Prototype :

MVSDK_API CameraSdkStatus CameraImageProcessEx
(
    CameraHandle          hCamera,
    BYTE*                pbyIn,
    BYTE*                pbyOut,
    tSdkFrameHead*      pFrInfo,
    UINT                 uOutFormat,
    UINT                 uReserved
);

Function Description: Camera raw image data obtained is processed superimposed saturation, gain and color correction, noise reduction and other treatment effect, and finally get the image data RGB888 format.

Parameter Description :
hCamera Camera handle, obtained by the CameralInit function.
pbyIn The input image data buffer address and can not be NULL.
pbyOut     The input image data buffer address and can not be NULL.
pFrInfo    After entering header information of the image processing is complete, the header information
uOutFormat After processing the image output format, which can be
            CAMERA_MEDIA_TYPE_MONO8, CAMERA_MEDIA_TYPE_RGB, CAMERA_MEDIA_TYPE_BGR, CAMERA_MEDIA_TYPE_RGBA8
            one of them.
pbyIn      Corresponding buffer size must match the specified format and OutFormat phase.
uReserved  Reserved parameter must be set to 0

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

4.21 CameraDisplayRGB24

Prototype :
MVSDK_API CameraSdkStatus CameraDisplayRGB24
(
    CameraHandle hCamera,
    BYTE *pbyRGB24,
    tSdkFrameHead *pFrInfo
);

Function Description : Display image.
Parameter Description:
    hCamera : Camera handle, obtained by the CameraInit function.
    *pbyRGB24 : Image data buffer, RGB888 format.
    *pFrInfo : Image header information.
return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: You must initialize CameraDisplayInit been called before calling this function.

Example :

/* Calling the display function displays the image */
CameraDisplayRGB24(pThis->m_hCamera, pThis->m_pFrameBuffer, pFrameHead);

4.22 CameraSetDisplaySize

Prototype :

MVSDK_API CameraSdkStatus CameraSetDisplaySize

(CameraHandle hCamera, INT iWidth, INT iHeight);

Function: Sets the display size of the control.

Function parameter description :

  hCamera : Camera handle, obtained by the CameraInit function.
  iWidth : width
  iHeight : height

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: You must initialize CameraDisplayInit been called before calling this function.

Example :
CameraSetDisplaySize(m_hCamera,rect.right - rect.left,rect.bottom - rect.top)

4.23 CameraGetImageBuffer

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraGetImageBuffer(
    CameraHandle hCamera,
    tSdkFrameHead* pFrameInfo,
    BYTE** pbyBuffer,
    UINT wTimes
);
```

**Function Description:** Obtaining an image data.

**Parameter Description:**

- `hCamera` : Camera handle, obtained by the CameraInit function.
- `*pFrameInfo` : Image header information pointer.
- `**pbyBuffer` : Point data buffer pointer image. As a result of zero-copy mechanisms to improve efficiency, hence the use of a pointer to the index.
- `wTimes` : Capture images timeout. Milliseconds. In wTimes time yet to obtain an image, the function returns a timeout message.

**return value** : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:** In order to improve efficiency, SDK during image capture using zero-copy mechanism, CameraGetImageBuffer actually get is a kernel buffer address, after a successful call to this function must be freed by calling CameraReleaseImageBuffer CameraGetImageBuffer the resulting buffer, so that the kernel continue to use the buffer.
Example:

```c
/* */
CameraGetImageBuffer(pThis->m_hCamera,&sFrameInfo,&pbyBuffer,1000);
```

4.24 CameraGetImageBufferEx

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraGetImageBufferEx
(
    CameraHandle hCamera,
    INT*           piWidth,
    INT*           piHeight,
    UINT           wTimes
);
```

**Function:** get an image data. Image data is different from the Camera Get Image Buffer function, which obtained the RGB format. Subsequent need to call Camera Image Process function and function Camera Release ImageBuffer.

**Parameter Description:**

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `piWidth`: Int pointers, return width of the image.
- `piHeight`: Int pointers, returning the image height Times: capture images timeout. Milliseconds.
- `wTimes`: Not yet been obtained within the image, the function returns a timeout message.

**Return Value:** When successful, the image data buffer returns the first address fails, the return NULL or 0.

4.25 CameraReleaseImageBuffer

**Prototype:**

```c
MVSDK_API CameraSdkStatus
```
CameraReleaseImageBuffer
(
    CameraHandle hCamera,
    BYTE *pbyBuffer
);

**Function Description:** Released by the Camera Get Image Buffer buffer is obtained.

**Parameter Description:**
- hCamera: Camera handle, obtained by the CameraInit function.
- *pbyBuffer: Buffer address by the Camera Get Image Buffer obtained.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**
/* */
CameraReleaseImageBuffer(pThis->m_hCamera,pbyBuffer);

### 4.26 CameraGetFrameStatistic

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraGetFrameStatistic
(
    CameraHandle       hCamera,
    tSdkFrameStatistic *psFrameStatistic
);
```

**Function Description:** Get the camera frame rate of the received statistics, including error frames and frame loss situation.

**Parameter Description:**
- hCamera: Camera handle, obtained by the CameraInit function.
*psFrameStatistic : Pointer to return statistics.

**Return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

### 4.27 CameraSetNoiseFilter

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraSetNoiseFilter
(
    CameraHandle hCamera,
    BOOL bEnable
);
```

**Function Description:** Set the image noise reduction module is enabled.

**Parameter Description:**
- hCamera : Camera handle, obtained by the CameraInit function.
- bEnable : TRUE, enabled; FALSE, prohibited.

**Return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

### 4.28 CameraGetNoiseFilterState

**Prototype:**

```c
```
MVSDK_API CameraSdkStatus
CameraGetNoiseFilterState
(
    CameraHandle hCamera,
    BOOL *pEnable
);

**Function Description:** Get enabled image noise reduction module.

**Parameter Description:**
- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***pEnable**: Indicators, return status. TRUE, it is enabled.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:**

**Example:**

4.29 CameraInitRecord

**Prototype:**
MVSDK_API CameraSdkStatus
CameraInitRecord
(
    CameraHandle hCamera,
    int iFormat,
    char *pcSavePath,
    BOOL b2GLimit,
    DWORD dwQuality,
    int iFrameRate
);

80
Function Description: Initiate a video.

Parameter Description:

- hCamera: Camera handle, obtained by the CameraInit function.
- iFormat: Video format, currently supports only compression and MSCV two ways. 0: no compression; 1: MSCV compresses.
- *pcSavePath: Video file path.
- b2GLimit: If the automatic segmentation is TRUE, files larger than 2G.
- dwQuality: Video quality factor, the bigger, the better the quality. Range 1-100.
- iFrameRate: Video frame rate. Recommended settings than the actual frame rate capture large, so it will not leak frames.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.30 CameraStopRecord

Prototype:

MVSDK_API CameraSdkStatus CameraStopRecord(  
    CameraHandle hCamera
);  

Function Description: The end of this video.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.
**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**: 

```
```

### 4.31 CameraPushFrame

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraPushFrame(
    CameraHandle hCamera,
    BYTE *pbyImageBuffer,
    tSdkFrameHead *pFrInfo
);
```

**Function**: The data is stored in a video stream.

**Parameter Description**:

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pbyImageBuffer`: Image data buffer must be RGB format.
- `*pFrInfo`: Image header information.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: You must call CameraInit Record to call the function.

After CameraStopRecord call, you can not call the function. Since our header information carries the time stamp information of image acquisition, so can record precise time synchronization, the frame rate without being affected by instability.
4.32 CameraSaveImage

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSaveImage(
    CameraHandle hCamera,
    LPCTSTR lpszFileName,
    BYTE *pbyImageBuffer,
    tSdkFrameHead *pFrInfo,
    BYTE byFileType,
    BYTE byQuality
);
```

Function Description: Save the image buffer data into image files.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **lpszFileName**: Save the picture file full path.
- **pbyImageBuffer**: Image data buffer.
- **pFrInfo**: Image header information.
- **byFileType**: Save the image format. See CameraDefine.h in the range emSdkFileType type definitions. Currently supports BMP, JPG, PNG, RAW four formats. Which means that the original RAW data output from the camera, save the RAW file format requirements pbyImageBuffer and pFrInfo data obtained by the CameraGetImageBuffer, but without CameraImageProcess converted into BMP format; the other hand, if you want to save as BMP, JPG or PNG format, and pFrInfo pbyImageBuffer after the data is
RGB format CameraImageProcess process. Specific usage can refer to the Advanced routines.

byQuality : Quality factor to save the image only when saved as JPG format This parameter is valid, the range of 1-100. The rest can be written 0 format.

return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note :

Example :

4.33 CameraSetMirror

Prototype :

MVSDK_API CameraSdkStatus CameraSetMirror (CameraHandle hCamera, INT iDir, BOOL bEnable);

Function Description : Set the image mirrored. Mirroring is divided into two horizontal and vertical directions.

Parameter Description :

hCamera : Camera handle, obtained by the CameraInit function.

iDir : Mirroring direction to be obtained. 0: horizontal direction; 1: a vertical direction.

bEnable : TRUE, enabling the mirror; FALSE, prohibit the mirror
**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:


4.34 CameraGetMirror

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraGetMirror(
    CameraHandle hCamera,
    INT iDir,
    BOOL *pbEnable
);
```

**Function Description**: Get a mirror image of the state.

**Parameter Description**:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iDir**: Mirroring direction to be obtained. 0: horizontal direction; 1: a vertical direction.
- ***pbEnable**:

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**note**:

**Example**:
4.35 CameraSetMonochrome

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetMonochrome(
    CameraHandle hCamera,
    BOOL bEnable
);
```

Function Description: Converted to black and white color setting function is enabled.

Parameter Description:
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `bEnable`: TRUE, shows a color image into black and white.

ReturnValue: When successful, return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.36 CameraGetMonochrome

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetMonochrome(
    CameraHandle hCamera,
    BOOL *pbEnable
);
```
Function Description: Get black and white color conversion function is enabled status.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.

*pbEnable: pointer. Returns TRUE opens the color image is converted to black and white images.

return value: When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.37 CameraSetInverse

Prototype:

MVSDK_API CameraSdkStatus CameraSetInverse

(CameraHandle hCamera,
BOOL bEnable);

Function Description: Setting color image color inversion function is enabled. (negative)

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.

bEnable: TRUE, indicates turning color image flip function, you can get a similar negative film effect.

return value: When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note:
4.38 CameraGetInverse

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetInverse(
    CameraHandle hCamera,
    BOOL *pbEnable
);
```

Function Description: Obtain image color inversion function is enabled.

Parameter Description:
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pbEnable`: Pointer, the function returns the enable state.

Return Value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note:

Example:

4.39 CameraSetImageResolution

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetImageResolution(
```

CameraHandle hCamera,
tSdkImageResolution *pImageResolution
);

**Function Description** : Set Preview resolution.

**Parameter Description** :

  hCamera : Camera handle, the function obtained by the CameraInit.
  *pImageResolution : Structure pointer, used to set the current resolution.

**return value** : When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example** :


### 4.40 CameraGetImageResolution

**Prototype** :

```c
MVSDK_API CameraSdkStatus CameraGetImageResolution
(
    CameraHandle hCamera,
    tSdkImageResolution *psCurImageResolution
);
```

**Prototype** : Get the current preview resolution.

**Parameter Description** :

  hCamera : Camera handle, obtained by the CameraInit function.
  *psCurImageResolution : Structure pointer, used to return the current resolution.
**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```

```

### 4.41 CameraGetMediaType

**Prototype**:

```
MVSDK_API CameraSdkStatus
CameraGetMediaType
(
    CameraHandle hCamera,
    INT *piMediaType
);
```

**Function Description**: Get the current camera output format index numbers for the original data.

**Parameter Description**:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **piMediaType**: Pointer used to return the current format type index. Property obtained by the camera CameraGetCapability, in tSdkCameraCapability structure in pMediaTypeDesc members, holds an array of formats supported by the camera, piMediaType points to the index number, that is, the array index number. pMediaTypeDesc [*piMediaType] .iMediaType said current encoding format. The coding see CameraDefine.h the [image format definition] section.
return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.42 CameraSetMediaType

Prototype:

MVSDK_API CameraSdkStatus CameraSetMediaType(
    CameraHandle hCamera,
    INT iMediaType
);

Function Description: Setting the current camera output format index numbers for the original data.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.

*piMediaType: Pointer used to return the current format type index. Property obtained by the camera CameraGetCapability, in tSdkCameraCapability structure in pMediaTypeDesc members, holds an array of formats supported by the camera, piMediaType points to the index number, that is, the array index number. pMediaTypeDesc [*piMediaType] .iMediaType said current encoding format. The coding see CameraDefine.h the [image format definition] section.
return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.43 CameraSetAeState

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetAeState(
    CameraHandle hCamera,
    BOOL bState
);
```

Function Description: Set the camera exposure mode. Automatically or manually.

Parameter Description:

- hCamera: Camera handle, obtained by the CameraInit function.
- bState: TRUE, Enable automatic exposure; FALSE, stop auto exposure.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
**4.44 CameraGetAeState**

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraGetAeState
(
    CameraHandle hCamera,
    BOOL *pbAeState
);
```

**Function Description:** Get the current camera exposure mode.

**Parameter Description:**

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pbAeState`: Indicators for the return to the automatic exposure enable state.

**Return Value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```c
```

**4.45 CameraSetAeTarget**

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraSetAeTarget
(
    CameraHandle hCamera,
    INT iAeTarget
);
```

**Function Description:** AE brightness setting target.
Parameter Description:

hCamera : Camera handle, the function obtained by the CameraInit function.
iAeTarget : Brightness target value.

return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: The setting range by the Camera GetCapability function to obtain.

Example :

4.46 CameraGetAeTarget

prototype :

MVSDK_API CameraSdkStatus CameraGetAeTarget (CameraHandle hCamera, INT *piAeTarget);

Function Description : Attain AE brightness target value.

Parameter Description :

hCamera : Camera handle, obtained by the CameraInit function.
*piAeTarget : Pointer, return target.

return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example :
4.47 CameraSetExposureTime

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraSetExposureTime
(
    CameraHandle     hCamera,
    double           fExposureTime
);
```

**Function Description:** Set the exposure time. Microseconds.

**Parameter Description:**

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **fExposureTime**: Exposure time in microseconds.

**Return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:** For CMOS sensor, which is a unit of exposure calculated by row, therefore, the exposure time is not adjustable in the microsecond. But will follow the entire line to choose. After calling this function to set the exposure time, it is recommended to obtain the value and then transferred CameraGetExposureTime actually set.

**Example:**

4.48 CameraGetExposureTime

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraGetExposureTime
(
    CameraHandle     hCamera,
    double           *pfExposureTime
);
```
Function Description: Get exposure time. Microseconds.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- pfExposureTime: Pointer, return the exposure time in microseconds.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

4.49 CameraGetExposureLineTime

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetExposureLineTime(
    CameraHandle hCamera,
    double *pfLineTime)
```

Function Description: Get a line exposure time.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h

Example:
4.50 CameraSetAnalogGain

Prototype:

MVSDK_API CameraSdkStatus CameraSetAnalogGain(
    CameraHandle hCamera,
    INT usAnalogGain
);

Function Description: Set the camera's image analog gain value.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.

usAnalogGain: 设定的模拟增益值。

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: This value is multiplied by the camera attribute structure CameraGetCapability obtained sExposureDesc.fAnalogGainStep, get the actual image signal magnification.

Example:

4.51 CameraGetAnalogGain

Prototype:

MVSDK_API CameraSdkStatus CameraGetAnalogGain(
    CameraHandle hCamera,
    INT *pusAnalogGain
);
Function Description: To obtain an analog gain value of the image signal.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- *pusAnalogGai: Pointer, returns the current value of the analog gain.

return value: On success, returns CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note:

Example:

4.52 CameraSetAeWindow

Prototype:
MVSDK_API CameraSdkStatus CameraSetAEWindow
(
    CameraHandle hCamera,
    int iHOff,
    int iVOff,
    int iWidth,
    int iHeight
);

Function Description: Set the AE reference window

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- iHOff: Abscissa of the upper left corner of the window
- iVOff: Ordinate the upper left corner of the window
- iWidth: Width of the window
- iHeight: Height of the window
return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: 1/2 the size of the window is set to the middle of each resolution. It can change the resolution and follow along with the change; if iHOff, iVOff, iWidth, window position range iHeight determined beyond the current scope of the resolution, the size of the window automatically centered 1/2

Example:

4.53 CameraGetAEWindow

Prototype:

MVSDK_API CameraSdkStatus CameraGetAEWindow
(
    CameraHandle    hCamera,
    INT*            piHOff,
    INT*            piVOff,
    INT*            piWidth,
    INT*            piHeight
);

Function Description: 获得自动曝光参考窗口的位置。

Parameter Description:

hCamera：Camera handle, obtained by the CameraInit function.

piHOff：Pointer return to the window location of the upper left corner abscissa value.

piVOff：Pointer return to the window position the upper left corner ordinate values.
piWidth：Index, returns the width of the window.
piHeight：Index return to the window height.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:**

**Example:**

### 4.54 CameraSetSharpness

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraSetSharpness
(
    CameraHandle hCamera,
    INT iSharpness
);
```

**Function Description:** Sharpening parameter setting process image.

**Parameter Description:**

- **hCamera**：Camera handle, obtained by the CameraInit function.
- **iSharpness**：Sharpening parameters. Camera GetCapability obtained by the scope of generally [0,100], 0 disables sharpening.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**
4.55 CameraGetSharpness

Prototype:

```c
 MVSDK_API CameraSdkStatus CameraGetSharpness(
   CameraHandle hCamera,
   INT *piSharpness
);
```

Function Description: Get the current value of the sharpening settings.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **piSharpness**: Pointer, returned sharpening setting value currently set.

Return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```
4.56 CameraSetOnceWB

Prototype:

```c
 MVSDK_API CameraSdkStatus CameraSetOnceWB(
   CameraHandle hCamera
);
```
**Function Description**: In the manual white balance mode, call the function once the white balance. Time is in effect upon receipt of the next frame of image data.

**Parameter Description**: Camera: The camera handle, obtained by the Camera Init function.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```
```

### 4.57 CameraSetOnceBB

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraSetOnceBB(
    CameraHandle    hCamera
);
```

**Function Description**: Perform a black balance operation.

**Parameter Description**: hCamera: Camera handle, obtained by the CameraInit function.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```
```
4.58 CameraSetLutMode

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraSetLutMode(
    CameraHandle hCamera,
    INT emLutMode
);
```

**Function Description:** Set the camera look-up table conversion mode NAT mode.

**Parameter Description:**

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **emLutMode**: LUTMODE_PARAM_GEN represented dynamically generated by gamma and contrast parameters LUT table.
  - LUTMODE_PRESET that the use of pre-LUT table.
  - LUTMODE_USER_DEF that the use of user-defined LUT table.

LUTMODE_PARAM_GEN defined reference CameraDefine.h in emSdkLutMode type.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```c
```

4.59 CameraGetLutMode

**Prototype:**
MVSDK_API CameraSdkStatus CameraGetLutMode
(
    CameraHandle hCamera,
    INT *pemLutMode
);

**Function Description**: Get the camera look-up table conversion mode NAT mode

**Parameter Description**:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***pemLutMode**: Pointer, returns the current LUT mode. Significance and CameraSetLutMode in emLutMode same parameters.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

### 4.60 CameraSelectLutPreset

**Prototype**:

MVSDK_API CameraSdkStatus CameraSelectLutPreset
(
    CameraHandle hCamera,
    int iSel
);

**Function Description**: Select the default LUT table LUT mode. You must use CameraSetLutMode
The LUT mode as the default mode.

**Parameter Description:**

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iSel**: Table index number. Number of tables from the Camera GetCapability obtained.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```

```

### 4.61 CameraGetLutPresetSel

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraGetLutPresetSel (CameraHandle hCamera, int* piSel);
```

**Function Description**: Gets the default number LUT LUT indexing mode.

**Parameter Description**:

- **hCamera**: Gets the default number LUT LUT indexing mode.
- **piSel**: Pointer, the return table index number.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```

```
### 4.62 CameraSetCustomLut

**Prototype:**

```
MVSDK_API CameraSdkStatus CameraSetCustomLut
(
    CameraHandle            hCamera,
    BYTE*                   pLut
);
```

**Function Description:** Set up a custom LUT table.

**Parameter Description:**

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **pLut**: Pointer to address LUT table. LUT table size is 256 bytes, respectively, the code color channel from 0 to 256 corresponding map value.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: You must use CameraSetLutMode the LUT mode to Custom mode.

**Example**:

```
```

### 4.63 CameraGetCustomLut

**Prototype**:

```
MVSDK_API CameraSdkStatus CameraGetCustomLut
(
    CameraHandle            hCamera,
    BYTE*                   pLut
);
```
Function Description: Get a custom LUT tables currently in use.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.
pLut: Pointer used to return a custom LUT tables currently in use. LUT table size is 256 bytes, respectively, the code color channel from 0 to 256 corresponding map value. Custom LUT table mode values are 0-255. It represents 1: 1 linear image without changing the original value.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note:

Example:

4.64 CameraGetCurrentLut

Prototype:

MVSDK_API CameraSdkStatus
CameraGetCurrentLut
(
    CameraHandle hCamera,
    BYTE* pLut
):

Function Description: Get the camera's current LUT table can be invoked in any LUT mode.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.
pLut：Pointer, used to return the camera's current LUT table.

return value：When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.65 CameraSetWbMode

Prototype:

```cpp
MVSDK_API CameraSdkStatus CameraSetWbMode(
    CameraHandle hCamera,
    BOOL bAuto
);
```

Function Description：Set the camera white balance mode. Divided into manual and automatic mode.

Parameter Description:

- hCamera：Camera handle, obtained by the CameraInit function.
- bAuto ：TRUE, the command enables the automatic mode. FALSE, it means using the manual mode, by calling CameraSetOnceWB to a white balance.

return value：When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h

Example:
4.66 CameraGetWbMode

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraGetWbMode(
    CameraHandle hCamera,
    BOOL *pbAuto
);
```

**Function Description:** Get the current white balance mode.

**Parameter Description:**
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pbAuto`: Pointer, returns TRUE if the automatic mode, FALSE manual mode.

**Return Value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**


4.67 CameraIsWbWinVisible

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraIsWbWinVisible(
    CameraHandle hCamera,
    BOOL *pbShow
);
```
**Function Description** : Get the white balance display state of the window.

**Parameter Description:**
- hCamera : Camera handle, obtained by the CameraInit function.
- *pbShow : Pointer, returns TRUE, it indicates that the window is visible.

**return value** : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example :**

```
4.68 CameraGetWbWindow

**Prototype :**
MVSDK_API CameraSdkStatus CameraGetWbWindow
(
    CameraHandle hCamera,
    INT* PiHOff,
    INT* PiVOff,
    INT* PiWidth,
    INT* PiHeight
);
```

**Function Description** : Obtaining the position of the reference window WB.

**Parameter Description:**
- hCamera : Camera handle, the function obtained by the CameraInit.
- PiHOff : Indicators, reference back to the top left corner of the window abscissa.
- PiVOff : Indicators, reference back to the top left corner of the window ordinate.
- PiWidth : Index, returns the width of the reference window.
PiHeight：Index Returns the height of the reference window.

Return value：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```

```

4.69 CameraSetWbWindow

Prototype:

```
MVSDK_API CameraSdkStatus CameraSetWbWindow(
    CameraHandle hCamera,
    INT iHOff,    
    INT iVOff,    
    INT iWidth,   
    INT iHeight
);
```

Function Description：Setting the reference position of the window WB.
Parameter Description:

- hCamera：Camera handle, the function obtained by the CameraInit function.
- iHOff：Reference to the upper left corner of the window abscissa
- iVOff：Ordinate the upper left corner of the reference window
- iWidth：The width of the reference window
- iHeight：Height reference window

Return value：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note：
4.70 CameraImageOverlay

Prototype:

```c
MVSDK_API CameraSdkStatus CameraImageOverlay(  
    CameraHandle hCamera,  
    BYTE *pRgbBuffer,  
    tSdkFrameHead *pFrInfo
);
```

**Function Description:** Superimposed on the image data input crosshairs, white balance reference window, auto exposure reference window and other graphics. Visible only to cross the line and can be superimposed on the reference window.

**Parameter Description:**

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pRgbBuffer`: Image data buffer.
- `*pFrInfo`: Image header information.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**
4.71 CameraSetCrossLine

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraSetCrossLine(  
  CameraHandle hCamera,  
  int iLine,  
  INT x,  
  INT y,  
  UINT color,  
  BOOL bVisible  
);
```

**Function Description:** Setting parameter specifies the crosshairs.

**Parameter Description:**

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iLine**: To set represents the first of several cross state lines. The range [0,8], a total of nine.
- **x**: The horizontal crosshair center position value.
- **y**: Ordinates crosshair center position.
- **color**: Color crosshairs, format (R | (G << 8) | (B << 16))
- **bVisible**: It displays the status of crosshairs. TRUE, it indicates displayed.

**Return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: Only set to show crosshair state after calling CameraImageOverlay will be superimposed on the image.

**Example:**

\[\text{...}\]
4.72 CameraGetCrossLine

Prototype :

```c
MVSDK_API CameraSdkStatus
CameraGetCrossLine
(
    CameraHandle     hCamera,
    int               iLine,
    INT*             *px,
    INT*             *py,
    UINT*            *pcolor,
    BOOL*            *pbVisible
);
```

Function Description : Crosshairs for the specified state.

Parameter Description:

- **hCamera** : Camera handle, obtained by the CameraInit function.
- **iLine** : It indicates the status of the first few to get the crosshairs. The range [0,8], a total of nine.
- **px** : Pointer, returned to the center position of the horizontal crosshair.
- **py** : Pointer, returned to the center position of the horizontal crosshair.
- **pcolor** : Pointer, returns the color of the crosshairs, the format (R | (G << 8) | (B << 16)).
- **pbVisible** : Pointer, returns TRUE, it indicates that the crosshairs visible.

return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example :
4.73 CameraSetGain

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetGain(
    CameraHandle hCamera,
    int iRGain,
    int iGGain,
    int iBGain
);
```

Function Description: Set image digital gain.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iRGain**: Gain value of the red channel.
- **iGGain**: Gain value for the green channel.
- **iBGain**: Gain the blue channel.

Return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: The camera body is set by the scope attribute structure CameraGetCapability obtained sRgbGainRange member statements. The actual magnification is set value / 100.

Example:

```
```

4.74 CameraGetGain

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetGain
```
Function Description: Obtaining a digital image processing gain

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***piRGain**: Pointer, return the digital gain value for the red channel.
- ***piGGain**: Pointer to return the green channel digital gain value.
- ***piBGain**: Pointer, the return value of a digital gain of blue channel.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```c
4.75 CameraSetGamma

Prototype:

MVSDK_API CameraSdkStatus
CameraSetGamma
(
    CameraHandle hCamera,
    int iGamma
);

Function Description: Gamma setting LUT dynamically generated mode.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
iGamma: To set the Gamma value.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: Set value will be immediately stored in the internal SDK, but only when the camera is dynamic parameters generated LUT mode to take effect. Refer CameraSetLutMode function description section.

**Example**: 

```
```

### 4.76 CameraGetGamma

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraGetGamma(
    CameraHandle hCamera,
    int *piGamma
);
```

**Function Description**: Gamma value LUA obtain dynamically generated mode. Refer CameraSetGamma of functionality description.

**Parameter Description**:

- hCamera: Camera handle, obtained by the CameraInit function.
- *piGamma: Pointer, returns the current value of Gamma.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**:

**Example**:

```
```
4.77 CameraSetSaturation

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetSaturation(
    CameraHandle hCamera,
    INT iSaturation
);
```

Function Description: Set the saturation of the image processing.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- iSaturation: camera handle, obtained not the camera init function.

Return value: When successful, the return CAMERA_STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

Note: Black and white camera is invalid.

Example:

4.78 CameraGetSaturation

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetSaturation(
    CameraHandle hCamera,
    INT *piSaturation
);
```
**Function Description**: Get saturation image processing.

**Parameter Description**:
- hCamera: Camera handle, obtained by the CameraInit function.
- *piSaturation: Pointer, returns the current saturation value of image processing.

**Return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

4.79 CameraSetContrast

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraSetContrast(
    CameraHandle hCamera,
    INT iContrast
);
```

**Function Description**: Set contrast value LUT dynamically generated mode.

**Parameter Description**:
- hCamera: Camera handle, obtained by the CameraInit function.
- iContrast: Set contrast value.

**Return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: Set value will be immediately stored in the internal SDK, but only when the camera is dynamic parameters generated LUT mode to take effect. Refer CameraSetLutMode function description section.
4.80 CameraGetContrast

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetContrast(
    CameraHandle hCamera,
    INT *piContrast
);
```

Function Description: LUA contrast value obtained dynamically generated mode.

Parameter Description:
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*piContrast`: Pointer, returns the current contrast value.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```
```

4.81 CameraSetFrameSpeed

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetFrameSpeed(
```

```
```
CameraHandle hCamera,
INT iFrameSpeedSel
);

**Function Description**: Set the camera output image frame rate.

**Parameter Description**:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iFrameSpeedSel**: Select the frame rate mode index number, information structure range from 0 to CameraGetCapability obtained iFrameSpeedDesc - 1

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```
4.82 CameraGetFrameSpeed

**Prototype**:
MVSDK_API CameraSdkStatus
CameraGetFrameSpeed
(
    CameraHandle hCamera,
    INT *piFrameSpeedSel
);

**Function Description**: Get the camera frame rate of the output image selection index.

**Parameter Description**:
- **hCamera**: Camera handle, obtained by the CameraInit function.
*piFrameSpeedSel : Pointer back to select the frame rate mode index.

**return value :** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note:**

**Example :**

---

### 4.83 CameraSetAntiFlick

**Prototype :**

```c
MVSDK_API CameraSdkStatus CameraSetAntiFlick(
    CameraHandle hCamera,
    BOOL bEnable
);
```

**Function Description :** Anti-strobe function enabled automatic exposure setting.

**Parameter Description:**

- **hCamera :** Camera handle, obtained by the CameraInit function.
- **bEnable :** TRUE, turn on anti-strobe function; FALSE, turn off the feature.

**return value :** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note :** For manual exposure mode is invalid.

**Example :**
4.84 CameraGetAntiFlick

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetAntiFlick
(
    CameraHandle hCamera,
    BOOL *pbEnable
);
```

Function Description: Anti-strobe function enabled N get automatic exposure.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- *pbEnable: Pointer, returns the enable state of the feature.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.85 CameraGetLightFrequency

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetLightFrequency
(
    CameraHandle hCamera,
    int *piFrequencySel
);
```

Function Description: Get auto exposure, eliminate flicker frequency selection.
**Parameter Description:**

hCamera : Camera handle, obtained by the CameraInit function.

*piFrequencySel : Pointer to return the selected index number. 0: 50HZ 1: 60HZ

**return value** : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example :**

```
```

### 4.86 CameraSetLightFrequency

**Prototype :**

```c
MVSDK_API CameraSdkStatus CameraSetLightFrequency(
    CameraHandle hCamera,
    int iFrequencySel
);
```

**Function Description** : Set AE eliminate flicker frequency.

**Parameter Description:**

hCamera : Camera handle, obtained by the CameraInit function.

iFrequencySel : 0:50HZ , 1:60HZ

**return value** : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example :**

```
```
4.87 CameraSetTransPackLen

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetTransPackLen
(
    CameraHandle hCamera,
    INT iPackSel
);
```

Function Description: Setting the size of the camera sub-transmission of image data. (Obsolete this way, SDK has changed the way auto-negotiation subcontractors, GIGE when using the camera, please enable NIC Jumbo frame feature enabled, SDK will automatically enable the largest subcontractor to optimize performance)

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iPackSel**: Select the length of the sub-index number. Subcontracting length attribute structure can get the camera body pPackLenDesc members expressed, iPackLenDesc member indicates the maximum number of optional sub-mode.

Return Value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:


4.88 CameraGetTransPackLen

Prototype:

```c
MVSDK_API CameraSdkStatus
```

CameraGetTransPackLen
(
    CameraHandle hCamera,
    INT    *piPackSel
);

**Function Description**: Obtain current transmission sub-size camera selection index.

**Parameter Description**:  
- hCamera: Camera handle, obtained by the CameraInit function.
- *piPackSel: Pointer, returns the current size of the selected sub-index.

**Return Value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:  

4.89 CameraWriteSN

**Prototype**:  
MVSDK_API CameraSdkStatus CameraWriteSN
(
    CameraHandle hCamera,
    BYTE    *pcSN,
    INT     iLevel
);

**Function Description**: Set the camera’s serial number. Our company is divided into three camera numbers. 0 The factory default camera number,
grade 1 and grade 2 left secondary development. Each level numbers are 32 bytes.

**Parameter Description:**
- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***pcSN**: Buffer number.
- **iLevel**: To set the number of levels, only 1 or 2.

**Return Value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**:

**Example**:

```c
CameraReadSN(hCamera, *pcSN, iLevel);
```

### 4.90 CameraReadSN

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraReadSN(
    CameraHandle hCamera,
    BYTE *pcSN,
    INT iLevel
);
```

**Function Description**: The camera reads the number specified level

**Parameter Description**:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***pcSN**: Buffer number.
- **iLevel**: Reference level to read. Only 1 and 2.

**Return Value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.
4.91 CameraSaveParameter

Prototype:

```
MVSDK_API CameraSdkStatus
CameraSaveParameter
(
    CameraHandle              hCamera,
    INT                       iTeam
);
```

**Function Description**: Save the current camera parameters to the specified parameter set. The camera provides the A, B, C, D four groups space to save the parameters.

**Parameter Description**:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iTeam**: PARAMETER_TEAM_A Save to group A, PARAMETER_TEAM_B Save to group B, PARAMETER_TEAM_C Save to group C, PARAMETER_TEAM_D Save to group D,

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```
```

4.92 CameraLoadParameter

Prototype:
MVSDK_API CameraSdkStatus
CameraLoadParameter
(
    CameraHandle hCamera,
    INT iTeam
);

**Function Description**: Load designated group to the camera.

**Parameter Description**:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iTeam**: PARAMETER_TEAM_A Save to group A,
  PARAMETER_TEAM_B Save to group B,
  PARAMETER_TEAM_C Save to group C,
  PARAMETER_TEAM_D Save to group D,

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

### 4.93 CameraReadParameterFromFile

**Prototype**:

```c
MVSDK_API CameraSdkStatus
CameraReadParameterFromFile
(
    CameraHandle hCamera,
    char *sFileName
);
```
**Function Description**: Loading parameters from the parameters specified on the PC profile. Our camera parameters, stored on the PC for the .config suffix file, located in the installation under Camera\Configs folder.

**Parameter Description**:
- hCamera: Camera handle, obtained by the CameraInit function.
- *sFileName: The full path of the parameter file.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```

```

4.94 CameraGetCurrentParameterGroup

**Prototype**:
MVSDK_API CameraSdkStatus CameraGetCurrentParameterGroup
(
    CameraHandle hCamera,
    INT *piTeam
);

**Function Description**: Get the current selected parameter set.

**Parameter Description**:
- hCamera: Camera handle, obtained by the CameraInit function.
- *piTeam: Pointer to return the currently selected parameter set.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```

```
4.95 CameraEnumerateDevice

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraEnumerateDevice
(
    tSdkCameraDevInfo *pDSCameraList,
    INT *piNums
);
```

**Function Description:** Enumerates devices, and building a device list.

**Parameter Description:**

- *piNums*: The number of indicators of the device, when an incoming call to pDSCameraList is made. The number of elements in the array, the function returns, save the actual number of devices found.

**Return Value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition in CameraStatus.h error codes.

**Note:** Before calling CameraInit, you must call this function to obtain information about the device. PiNums point value must be initialized, no more than the number of array elements pDSCameraList, or they may cause memory overflow.

**Example:**


4.96 CameraGetCapability

**Prototype:**

```c
```
Function Description: Get the camera characterization structure.

Parameter Description:

hCamera: Camera handle, obtained by the CameraInit function.

*pCameraInfo: Pointer to return the camera structure characterization. tSdkCameraCapbility defined in CameraDefine.h.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.97 CameraSetTriggerCount

Prototype:

MVSDK_API CameraSdkStatus CameraSetTriggerCount

CamroversetTriggerCount

( CameraHandle hCamera,
  INT iCount

);
iCount：A trigger acquisition frames

**return value**：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the error code definitions CameraStatus.h.

**Note**：Software trigger and hardware trigger mode are valid. The default is 1, ie, a trigger signal acquisition an image.

Example:

4.98 CameraGetTriggerCount

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraGetTriggerCount(
    CameraHandle hCamera,
    INT *piCount
);
```

**Function Description**：Get a trigger frames.

**Parameter Description**:

- hCamera：Camera handle, obtained by the CameraInit function.
- *piCount：Pointer to return a trigger acquisition frames.

**return value**：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.99 CameraGetTriggerDelayTime

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetTriggerDelayTime
(
    CameraHandle hCamera,
    UINT *puDelayTimeUs
);
```

Function Description: Get the current setting of a hard trigger delay time.

Parameter Description:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- ***puDelayTimeUs**: Pointer, returns the delay time in microseconds.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Note: Only part of the model of camera support this function.

Example:

```
4.100 CameraSetTriggerDelayTime

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetTriggerDelayTime
(
    CameraHandle hCamera,
    UINT uDelayTimeUs
);
```
**Function Description**: Trigger delay time setting hardware trigger mode, in microseconds.

**Parameter Description**:
- `hCamera`: Camera handle, obtained by the `CameraInit` function.
- `uDelayTimeUs`: Trigger delay. In microseconds.

**return value**: When successful, the return `CAMERA STATUS_SUCCESS (0)`; otherwise it returns a non-zero value of the error codes, refer to the definition `CameraStatus.h` error codes.

**Note**: 仅部分型号的相机支持该功能。

**Example**:

```c
```

### 4.101 CameraSoftTrigger

**Prototype**:

```c
MVSDK_API CameraSdkStatus CameraSoftTrigger(
    CameraHandle hCamera
);
```

**Function Description**: Perform a soft trigger.

**Parameter Description**: `hCamera`: Camera handle, obtained by the `CameraInit` function.

**return value**: When successful, the return `CAMERA STATUS_SUCCESS (0)`, otherwise it returns a non-zero value of the error codes, refer to the definition `CameraStatus.h` error codes.

**Example**:

```c
```
4.102 CameraSetTriggerMode

Prototype:
MVSDK_API CameraSdkStatus
cameraSetTriggerMode(
    CameraHandle hCamera,
    INT iTriggerModeSel
);  

Function Description: Set the camera's trigger mode.
Parameter Description:
  hCamera: Camera handle, obtained by the CameraInit function.
  iTriggerModeSel: Mode Select index number. You can set the pattern
                  acquired by CameraGetCapability function. Please refer to
                  the definitions CameraDefine.h in tSdkCameraCapability.
                  Typically, 0 represents the continuous acquisition mode;
                  1 represents a software trigger mode; 2 represents a hardware
                  trigger mode.
return value: When successful, the return CAMERA_STATUS_SUCCESS (0);
              otherwise it returns a non-zero value of the error codes, refer to the
              definition CameraStatus.h error codes.

Example:

4.103 CameraGetTriggerMode

Prototype:
MVSDK_API CameraSdkStatus
cameraGetTriggerMode(
);
Function Description: Get the camera trigger mode

Parameter Description:

  hCamera: Camera handle, obtained by the CameraInit function.
  *piTriggerMode: Pointer to return the currently selected camera trigger mode index number.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.104 CameraSnapToBuffer

Prototype:

MVSDK_API CameraSdkStatus CameraSnapToBuffer
(
    CameraHandle hCamera,
    tSdkFrameHead *pFrameInfo,
    BYTE **pbyBuffer,
    UINT uWaitTimeMs
);

Function Description: Capture an image into a buffer.

Parameter Description:

  hCamera: Camera handle, obtained by the CameraInit function.
  *pFrameInfo: Pointer return header information image.
**pbyBuffer**: A pointer to a pointer to the return address the image buffer.

`uWaitTimeMs`: Timeout, in milliseconds.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

#### 4.105 CameraGetResolutionForSnap

**Prototype**:

```c
MVSDK_API CameraSdkStatus
CameraGetResolutionForSnap
(
    CameraHandle            hCamera,
    tSdkImageResolution *pImageResolution
);
```

**Function Description**: Select the index number to obtain a resolution capture mode.

**Parameter Description**:

- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pImageResolution`: Pointer to return to capture mode resolution.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.106 CameraSetResolutionForSnap

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetResolutionForSnap
(
    CameraHandle hCamera,
    tSdkImageResolution *pImageResolution
);
```

**Function Description:** Set the resolution at the camera output image capture modes.

**Parameter Description:**
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `*pImageResolution`: If `ImageResolution->Width` and `ImageResolution->Height` is 0, then set to follow the current preview resolution. To capture the image resolution and the current setting will be the same as the preview resolution.

**return value:** When successful, the return `CAMERA_STATUS_SUCCESS` (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```c

```

4.107 CameraSetParameterTarget

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetParameterTarget
(
    CameraHandle hCamera,
    tSdkImageParameter *pImageParameter
);
```
Function Description : Audience set parameters access.

Parameter Description:

hCamera : Camera handle, obtained by the CameraInit function.

iTarget : Object parameter access. PARAM_ON_PC represents read and write parameters to save files from a PC, PARAM_ON_DEVICE read parameter file from the camera. Reference CameraDefine.h in emSdkParamTarget type definitions.

return value : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example :
**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Note**: 

**Example**:

4.109 **CameraSetParameterMask**

**Prototype**:

MVSDK_API CameraSdkStatus
CameraSetParameterMask
(
    CameraHandle hCamera,
    UINT uMask
);

**Function Description**: Set the mask parameter access. When loading and saving parameters will be determined based on the shield of each module parameters is loaded or saved.

**Parameter Description**:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **uMask**: shield. Reference Camera Define.h in emSdkPropSheetMask type definition.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

---
4.110 CameraRstTimeStamp

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraRstTimeStamp
(
    CameraHandle hCamera
);
```

Function Description: Reset image acquisition time stamp, starting from 0.

Parameter Description: hCamera: Camera handle, obtained by the CameraInit function.

Return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

```

```

4.111 CameraSaveUserData

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSaveUserData
(
    CameraHandle hCamera,
    UINT uStartAddr,
    BYTE *pbData,
    int ilen
);
```

Function Description: Save the user-defined data to the camera's non-volatile memory. The maximum length of the user data area of each camera models may not support the same. The length information can be obtained from the characterization of the device.
Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **uStartAddr**: Starting address, starting with 0. **Note**: Address must be 64-byte aligned.
- **pbData**: Data buffer pointer to return the read data.
- **ilen**: Read data length, len + StartAddr area must be less than the maximum length of the user.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.112 CameraLoadUserData

Prototype:

```c
MVSDK_API CameraSdkStatus CameraLoadUserData
    (CameraHandle hCamera,
    UINT uStartAddr,
    BYTE *pbData,
    int ilen);
```

Function Description: Read user-defined data from the camera's non-volatile memory. The maximum length of the user data area of each camera models may not support the same. The length information can be obtained from the characterization of the device.

Parameter Description:

- **hCamera**: Camera handle, obtained by the CameraInit function.
- **uStartAddr**: Starting address, starting with 0. **Note**: Address must be 64-byte aligned.
- **pbData**: Data buffer pointer to return the read data.
ilen: Data buffer pointer to return the read data.

return value: When successful, the return CAMERA STATUS_SUCCESS (0);
otherwise it returns a non-zero value of the error codes,
refer to the definition CameraStatus.h error codes.

Example:

4.113 CameraGetFriendlyName

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetFriendlyName
(
    CameraHandle    hCamera,
    char *           pName
);
```

Function Description: Read the camera's nickname, nickname default camera is automatically generated in the model name + # 0,1,2 means, such as Camera MV-U500 # 0 and Camera MV-U500 # 1, represents a computer connected on the same 2 500-megapixel camera, a camera for a separate set of nicknames, you can call CameraSetFriendlyName function, for example, where the first camera to My camera 1, then the two names on the MV-U500 are My camera 1 and camera MV-U500 # 1.

Parameter Description:
- hCamera: Camera handle, obtained by the CameraInit function.
- pName: Back camera nickname first string buffer address, buffer size requires more than 32 bytes

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.114 CameraSetFriendlyName

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetFriendlyName
(
    CameraHandle    hCamera,
    char*             pName
);
```

**Function Description:** Setting a camera nickname, after setting success nickname cured within the camera, this function can easily distinguish between multiple cameras.

**Parameter Description:**
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `pName`: Nickname pointing the camera to set the string buffer first address, nickname

  Must be less than 31 characters, the string is set to "auto", or "Automatic generation "indicates the default nickname.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

4.115 CameraSdkGetVersionString

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSdkGetVersionString
(
    char*       pVersionString
);
```
Function Description: Read user-defined data from the camera's non-volatile memory. The maximum length of the user data area of each camera models may not support the same. The length information can be obtained from the characterization of the device.

Parameter Description: pVersionString Back SDK version string buffer length is greater than the required 32 words.

return value: Back SDK version description string.

Example:

### 4.116 CameraGetEnumInfo

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraGetEnumInfo
(
    CameraHandle        hCamera,
    tSdkCameraDevInfo*  pCameraInfo
);
```

Function Description: Get a specified device enumeration information.

Parameter Description: hCamera Camera handle, obtained by the CameraInit function.

pCameraInfo Pointer, enumerations return information about the device.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

### 4.117 CameraSetI0State

Prototype:

```c
MVSDK_API CameraSdkStatus
```
CameraSetIOState
(
    CameraHandle hCamera,
    INT iOutputIOIndex,
    UINT uState
);

Function Description: IO setting specifies the level of the state, IO output type IO, 
IO cameras reserve programmable output is determined by the number of 
tSdkCameraCapability iOutputIoCounts.

Parameter Description:
    hCamera Camera handle, obtained by the CameraInit function.
    iOutputIOIndex IO index number, starting from 0.
    uState To set status 1 is high, 0 low

Return value: When successful, the return CAMERA STATUS_SUCCESS (0); 
otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.118 CameraGetIOState

Prototype:

MVSDK_API CameraSdkStatus
CameraGetIOState
(
    CameraHandle hCamera,
    INT iInputIOIndex,
    UINT* puState
);

Function Description: IO setting specifies the level of the state, IO is imported IO, 
IO cameras reserve programmable output is determined by the number of 
tSdkCameraCapability iInputIoCounts.
**Parameter Description:**

- **hCamera:** Camera handle, obtained by the CameraInit function.
- **iInputIOIndex:** IO index number, starting from 0.
- **puState:** Pointer, returns IO status, 1 is high, 0 low.

**Return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```plaintext```
```
```

### 4.119 CameraSetIspOutFormat

**Prototype:**

```c
MVSDK_API CameraSdkStatus CameraSetIspOutFormat(CameraHandle hCamera, UINT uFormat);
```

**Function Description:** Set output format CameraImageProcess image processing functions, support

- CAMERA_MEDIA_TYPE_MONO8
- CAMERA_MEDIA_TYPE_RGB8
- CAMERA_MEDIA_TYPE_RGBA8
- CAMERA_MEDIA_TYPE_BGR8
- CAMERA_MEDIA_TYPE_BGRA8

(defined in CameraDefine.h in) five kinds, corresponding to 8-bit grayscale images and 24-bit RGB, 32 the RGBA Wei, 24 BGR, 32 Wei BGRA format, the secondary development may be required visual library Select the output format, such as the need BGR format under OPENCV, but with QT development, you need to RGB format.

**Parameter Description:**

- **hCamera:** Camera handle, obtained by the CameraInit function.
- **uFormat:** To set the format provided, which can be

  ```c
  CAMERA_MEDIA_TYPE_MONO8
  ```
CAMERA_MEDIA_TYPE_RGB8、
CAMERA_MEDIA_TYPE_RGBA8、
CAMERA_MEDIA_TYPE_BGR8、
CAMERA_MEDIA_TYPE_BGRA8 其中之一。

**return value**：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```c
4.120 CameraGetIspOutFormat

**Prototype**:

MVSDK_API CameraSdkStatus
CameraGetIspOutFormat
(
    CameraHandle    hCamera,
    UINT*           puFormat
);

**Function Description**：Obtain Camera Get Image Buffer function image processing output format, which can be CameraSetIspOutFormat set.

**Parameter Description**: hCamera Camera handle, obtained by the CameraInit function.

puFormat It returns the currently set format, and its value is
    CAMERA_MEDIA_TYPE_MONO8 and
    CAMERA_MEDIA_TYPE_RGB8、
    CAMERA_MEDIA_TYPE_RGBA8、
    CAMERA_MEDIA_TYPE_BGR8、
    CAMERA_MEDIA_TYPE_BGRA8 one of them.

**return value**：When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.
Example :

### 4.121 CameraGetErrorString

**Prototype :**

MVSDK_API char*

CameraGetErrorString

( CameraSdkStatus  iStatusCode

);

**Function Description**: Get the error code corresponding to the description string.

**Parameter Description**: iStatusCode error code. (As defined in the Camera Status.h)

**return value**: When successful, enter the error code corresponding to the first string address.

Example :

### 4.122 CameraReConnect

**Prototype :**

MVSDK_API CameraSdkStatus

CameraReConnect

( CameraHandle  hCamera,

);

**Function Description**: Reconnect the device for USB, reconnect the device after GIGE unexpectedly dropped.

**Parameter Description**: hCamera Camera handle, obtained by the CameraInit function.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.
4.123 CameraConnectTest

Prototype:

```c
MVSDK_API CameraSdkStatus CameraConnectTest(
    CameraHandle    hCamera,
);
```

**Function Description**: Test the connection of the camera, for detecting whether the camera dropped.

**Parameter Description**:  
- `hCamera` Camera handle, obtained by the CameraInit function.

**Return value**:  
- 成功时，返回 `CAMERA_STATUS_SUCCESS (0)`，It represents not dropped; otherwise it returns a non-zero value of the error codes, refer to the definition `CameraStatus.h` error codes.

Example:

```
```

4.124 CameraSetTriggerDelayTime

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetTriggerDelayTime(
    CameraHandle    hCamera,
    UINT            uDelayTimeUs
);
```

**Function Description**: Trigger delay time setting hardware trigger mode, in microseconds. When the trigger signal is coming hard after a specified time delay, this function sets the delay time by calling this function.

Example:
delay, then start collecting images. Only part of the model of camera support this function. For details, please view product specifications.

**Parameter Description:**
- **hCamera** Camera handle, obtained by the CameraInit function.
- **uDelayTimeUs** Hard trigger delay. In microseconds.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```

```

### 4.125 CameraGetTriggerDelayTime

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraGetTriggerDelayTime
(
    CameraHandle  hCamera,
    UINT*          puDelayTimeUs
);
```

**Function Description:** Get the current setting of a hard trigger delay time.

**Parameter Description:**
- **hCamera** Camera handle, obtained by the CameraInit function.
- **puDelayTimeUs** Pointer, returns the delay time in microseconds.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example:**

```

```
4.126 CameraSetStrobeMode

Prototype:

```c
MVSDK_API CameraSdkStatus
CameraSetStrobeMode
(
    CameraHandle    hCamera,
    INT              iMode
);
```

Function Description: Setting STROBE signal IO pin terminals. The flash control signal can be done, you can also do an external mechanical shutter control.

Parameter Description:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- **iMode**: When the STROBE_SYNC_WITH_TRIG_AUTO and trigger signal synchronization, triggered when the camera exposure automatically generate STROBE signal. In this case, the effective polarity can be set (CameraSetStrobePolarity).
  - When is STROBE_SYNC_WITH_TRIG_MANUAL, the synchronization and the trigger signal, trigger, STROBE delay specified time (CameraSetStrobeDelayTime), then for a specified time Pulse (CameraSetStrobePulseWidth), effective polarity can be set (CameraSetStrobePolarity).
  - When is STROBE_ALWAYS_HIGH time, STROBE signal constant is high, ignoring other settings
  - When is STROBE_ALWAYS_LOW time, STROBE signal constant is low, ignoring other settings

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.127 CameraGetStrobeMode

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetStrobeMode(
    CameraHandle    hCamera,
    INT*             piMode
);
```

**Function Description**: Or STROBE signal setting mode current.

**Parameter Description**:
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `piMode`: Pointer to return
  - `STROBE_SYNC_WITH_TRIG_AUTO`,
  - `STROBE_SYNC_WITH_TRIG_MANUAL`,
  - `STROBE_ALWAYS_HIGH` or
  - `STROBE_ALWAYS_LOW`.

**Return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

4.128 CameraSetStrobeDelayTime

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetStrobeDelayTime(
    CameraHandle    hCamera,
    UINT              uDelayTimeUs
);
```
Function Description: When STROBE signal is STROBE_SYNC_WITH_TRIG, through the function sets the relative trigger delay time.

Parameter Description: hCamera     Camera handle, obtained by the CameraInit function.
                        uDelayTimeUs    Relative trigger delay time of the signal, the unit is us. It can be 0, but cannot be negative.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.129 CameraGetStrobeDelayTime

Prototype:
MVSDK_API CameraSdkStatus CameraGetStrobeDelayTime(
    CameraHandle    hCamera,
    UINT*           upDelayTimeUs
);

Function Description: When STROBE signal is STROBE_SYNC_WITH_TRIG, obtained by the function of its relative trigger delay time.

Parameter Description: hCamera     Camera handle, obtained by the CameraInit function.
                        upDelayTimeUs    Pointer, returns the delay time in us.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.130 CameraSetStrobePulseWidth

Prototype:

```c
MVSDK_API CameraSdkStatus CameraSetStrobePulseWidth(
    CameraHandle    hCamera,
    UINT            uTimeUs
);
```

**Function Description**: When STROBE signal is STROBE SYNC WITH TRIG, set its pulse width by the function.

**Parameter Description**: 
- `hCamera`: Camera handle, obtained by the CameraInit function.
- `uTimeUs`: Pulse width in time us.

**Return Value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

4.131 CameraGetStrobePulseWidth

Prototype:

```c
MVSDK_API CameraSdkStatus CameraGetStrobePulseWidth(
    CameraHandle    hCamera,
    UINT*            upTimeUs
);
```

**Function Description**: When STROBE signal is STROBE SYNC WITH TRIG, obtained by the pulse width function.

**Parameter Description**: 
- `hCamera`: Camera handle, obtained by the CameraInit function.
return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.132 CameraSetStrobePolarity

Prototype:

```
MVSDK_API CameraSdkStatus CameraSetStrobePolarity (
    CameraHandle hCamera,
    INT uPolarity
);
```

Function Description: When STROBE signal is STROBE_SYNC_WITH_TRIG, set the active level through the polar function. The default is active high, when the trigger signal arrives, STROBE signal is pulled high.

Parameter Description: hCamera Camera handle, obtained by the CameraInit function.

uPolarity STROBE signal polarity is active-low 0, 1 is high

Ping valid. The default is active high.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.133 CameraGetStrobePolarity

Prototype:
MVSDK_API CameraSdkStatus
CameraGetStrobePolarity
(
    CameraHandle    hCamera,
    INT*             upPolarity
);

Function Description : Effective polarity current camera STROBE signal. The default
is active high.

Parameter Description:
    hCamera Camera handle, obtained by the CameraInit function.
    upPolarity Pointer, returns the currently active STROBE signal polarity.

return value : When successful, the return CAMERA STATUS_SUCCESS (0);
otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example :

4.134 CameraSetExtTrigSignalType

Prototype :

MVSDK_API CameraSdkStatus
CameraSetExtTrigSignalType
(
    CameraHandle    hCamera,
    INT              iType
);

Function Description : Set the camera external trigger signal type. The upper edge,
lower edge, or high and low mode.

Parameter Description:
    hCamera Camera handle, obtained by the CameraInit function.
**iType** External trigger signal type, return value: Reference Camera Define.h in emExtTrigSignal type definition.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.135 CameraGetExtTrigSignalType

**Prototype:**

```c
MVSDK_API CameraSdkStatus
CameraGetExtTrigSignalType
(
    CameraHandle    hCamera,
    INT*             ipType
);
```

**Function Description:** Get the camera's current external trigger signal type.

**Parameter Description:**
- **hCamera** Camera handle, obtained by the CameraInit function.
- **ipType** Pointer, return external trigger signal type, return value: Reference Camera Define.h in emExtTrigSignal type definition.

**return value:** When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:
4.136 CameraSetExtTrigDelayTime

Prototype:

MVSDK_API CameraSdkStatus CameraSetExtTrigDelayTime
(
    CameraHandle hCamera,
    UINT uDelayTimeUs
);

Function Description: Setting the external trigger signal delay time, the default is 0 microseconds. When the value is not 0 uDelayTimeUs set, the camera after receiving external trigger signal, the delay uDelayTimeUs microseconds after the image capture.

Parameter Description:

hCamera Camera handle, obtained by the CameraInit function.

uDelayTimeUs Delay time in microseconds. The default is 0.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.137 CameraGetExtTrigDelayTime

Prototype:

MVSDK_API CameraSdkStatus CameraGetExtTrigDelayTime
(
    CameraHandle hCamera,
    UINT* upDelayTimeUs
);

Example:
**Function Description**: Get set external trigger signal delay time, the default is 0 microseconds.

**Parameter Description**: `hCamera` Camera handle, obtained by the CameraInit function.

`upDelayTimeUs` Pointer, return delay time.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:

```
4.138 CameraSetExtTrigJitterTime

Prototype:
MVSDK_API CameraSdkStatus
CameraSetExtTrigJitterTime
(
    CameraHandle hCamera,
    UINT uTimeUs
);

**Function Description**: Set the camera's external trigger signal debounce time, only when the external trigger mode select high or low trigger debounce time to take effect. The default is 0, in microseconds, maximum 150 milliseconds.

**Parameter Description**: `hCamera` Camera handle, obtained by the CameraInit function.

`uTimeUs` Microseconds.

**return value**: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

**Example**:
4.139 CameraGetExtTrigJitterTime

Prototype:

MVSDK_API CameraSdkStatus
CameraGetExtTrigJitterTime(
    CameraHandle    hCamera,
    UINT*            upTimeUs
);

Function Description: Get the camera set up outside the trigger debounce time, the default is 0. microseconds.

Parameter Description:
- **hCamera**: Camera handle, obtained by the CameraInit function.
- **upTimeUs**: Pointer to return debounce time setting.

return value: When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

Example:

4.140 CameraGetExtTrigCapability

Prototype:

MVSDK_API CameraSdkStatus
CameraGetExtTrigCapability(
    CameraHandle    hCamera,
    UINT*            puCapabilityMask
);


**Function Description** : Camera Properties mask obtain external trigger.

**Parameter Description:**
- `hCamera` Camera handle, obtained by the CameraInit function.
- `puCapabilityMask` Pointer, returns the camera external trigger characteristic mask, mask macro reference CameraDefine.h in EXT_TRIG_MASK_open header definition.

**return value** : When successful, the return CAMERA STATUS_SUCCESS (0); otherwise it returns a non-zero value of the error codes, refer to the definition CameraStatus.h error codes.

### 4.141 CameraAlignMalloc

**Prototype** :

```c
MVSDK_API BYTE*
CameraAlignMalloc
(  
    int size,
    int align
);
```

**Function Description** : Application address aligned memory block. With hardware acceleration for algorithms required aligned block of memory.

**Parameter Description:**
- `size` memory size
- `align` aligned address, if you need 16-byte aligned, it is set to 16

**return value** : When successful, return the application to the first memory address. On failure, returns NULL.

### 4.142 CameraAlignFree

**Prototype** :

```c
MVSDK_API VOID
CameraAlignFree
```


BYTE* membuffer

Function Description: Released by the Camera Align Malloc application address aligned memory block.

Parameter Description: membuffer CameraAlignMalloc returned by the memory address.

return value:
5 SDKInterface function by function explanation

In order to facilitate the user to quickly find a way to develop a feature, we have some of the common methods are summarized.

5.1 Camera Initialization and deinitialization

For camera workflow, we use the first enumeration, initialization, and then exit the program before the anti-initialize the camera.

- camera enumeration we provide two functions, CameraEnumerateDevice and CameraEnumerateDeviceEx. CameraEnumerateDevice will return the number of cameras and detailed descriptive information, including information in the manner described in the structure of the array is returned, such as the camera name, model, serial number, interface type and other details will be listed; CameraEnumerateDeviceEx will only return only one camera number.

- initialize the camera's we offer CameraInit, CameraInitEx, CameraInitEx2 these three functions. CameraInit CameraEnumerateDevice needs and supporting the use of, and CameraInitEx and CameraInitEx2 CameraEnumerateDeviceEx you need and supporting the use of. CameraInit need to pass information from the camera structure CameraEnumerateDevice obtained; CameraInitEx simply pass the camera's serial number ID can be, for example, initialize the first a camera, passing 0, the second camera is afferent 1, and so on; CameraInitEx2 is pass in the name of the camera, for example, by providing the tools we advance the camera name to "Camera1", when CameraInitEx2 calls, incoming "Camera1" can, for the case of multiple cameras simultaneously, this method can effectively build one to one relationship, but be aware that each name must be changed to a different camera, ensure uniqueness.

- Anti-initialization function, CameraInit. Either way initialization take anti initialization call CameraUnInit can.

The following is the use of Camera EnumerateDevices and CameraInit typical program flow:

tSdkCameraDevInfo sCameraList [10];
int iCameraNums;
int status;

iCameraNums = 10;
// Enumerate cameras, up to 10 return to the description of the camera
if (CameraEnumerateDevice (sCameraList, & iCameraNums) != CAMERA_STATUS_SUCCESS || iCameraNums == 0)
{
    return FALSE;
}
// If only one camera, iCameraNums will be modified CameraEnumerateDevice internal one.
    if ((status = CameraInit (& sCameraList [0], -1, -1, & m_hCamera)) != CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
    }

// Call before the program exits
CameraUnInit (m_hCamera);

The following is the use of CameraEnumerateDeviceEx and CameraInitEx
typical program flow:

int  iCameraNums;
iCameraNums = CameraEnumerateDeviceEx();
if (iCameraNums == 0) return FALSE;

    if ((status = CameraInitEx(0,-1,-1,&m_hCamera)) != CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
    }

//Before calling the program exitsCameraUnInit(m_hCamera);

The following is the use of CameraEnumerateDeviceEx and CameraInitEx2
typical program flow:

int  iCameraNums;
iCameraNums = CameraEnumerateDeviceEx();
if (iCameraNums == 0) return FALSE;

//This routine, use the "Camera 1" name initialization, the camera must first tool was renamed "Camera1"
    if ((status = CameraInitEx2("Camera1",&m_hCamera)) != CAMERA_STATUS_SUCCESS)
    {
        return FALSE;
    }
//Before calling the program exits
CameraUnInit(m_hCamera);

5.2 Saving and loading the camera parameters

Camera parameters are stored in the computer to a binary situation exists. Divided into A, B, C, D four groups on the interface. Loading is also divided by the models, by nicknames, and by serial number in three ways, as shown below:

- CameraSetParameterMode, set the parameters for saving manner. Refer to Chapter 4 Functions explained.
CameraSaveParameter, save the parameters to a specified set of parameters. Refer to Chapter 4 Functions explained.

CameraLoadParameter, Parameter specifies the group loaded. Refer to Chapter 4 Functions explained.

CameraSaveParameterToFile, to save current camera parameters to the specified file. Refer to Chapter 4 Functions explained.

CameraReadParameterFromFile, loaded camera parameters from the specified file. Refer to Chapter 4 Functions explained.

5.3 The camera takes images (Fig initiative taken or callback mode)

After initialization is complete, you can begin to take the camera diagram work. Taken in accordance with FIG manner, our SDK is divided into active and passive (callback) two.

active in image mode:
After completing the camera initialization, you can use the active CameraGetImageBuffer or CameraGetImageBufferEx3 function reads the image, the difference between the two is, CameraGetImageBuffer function obtained is the original RAW data buffer, use CameraImageProcess function RAW data is converted to the specified data format, After using the buffer pointer was finished, to call CameraReleaseImageBuffer release buffer usage rights, need to explain here is CameraReleaseImageBuffer just released by the CameraGetImageBuffer obtained the right to use the data buffer and does not apply and release the memory repeatedly, without worrying about efficiency question; CameraGetImageBufferEx3 functions are directly specified image data format, the specific format is determined by the input parameters, it can be 8-bit, 16-bit grayscale, or 24-bit, 32-bit color data format, and function calls CameraGetImageBufferEx3 after not need to call CameraReleaseImageBuffer. Two functions is common in image timeout can be set, for example, set 1000 ms timeout, then in 1000 ms, if no valid image, the function will be blocked, the thread is suspended until more than 1000 msec or reading a valid image, so the software architecture design, the user can create a thread dedicated image acquisition, image acquisition and has only need to set a reasonable time-out can be, or need to capture the image when calling a function, to obtain an image.

Here is CameraGetImageBuffer sample code:

tSdkFrameHead sFrameInfo;
BYTE* pbyBuffer;
CameraSdkStatus status;
/* pbyBuffer Automatically apply to a good memory, to store the original RAW data from the internal SDK*/
if (CameraGetImageBuffer(m_hCamera,&sFrameInfo,&pbyBuffer,1000) == CAMERA_STATUS_SUCCESS)
{
/* m_pFrameBuffer You need to apply good memory pb Buffer converted image data stored in the m_pFrameBuffer, the default will be the image data BRG24bit*/
    CameraImageProcess(m_hCamera, pbyBuffer, m_pFrameBuffer,&sFrameInfo);
    CameraReleaseImageBuffer(m_hCamera,pbyBuffer);
}

Here is CameraGetImageBufferEx3 sample code:

int width,height;//Image width and height
unsigned int uTimeStamp;//Image timestamp
BYTE*pImageData;
pImageData = CameraAlignMalloc(1280*1024*3,16);//It assumes 1.3 megapixel color camera

if(CameraGetImageBufferEx3(m_hCamera, pImageData ,1,&width,&height,&uTimeStamp,1000)
  == CAMERA_STATUS_SUCCESS)
{
    /*Successfully captured images, visual processing...... */
    /* pImageData Storing the color image data RGB24 format, width and height will return the image width and height, uTimeStamp will return to the image timestamp timeout, 1000 * 

    CameraAlignFree(pImageData);  //Practical application, when the program is initialized to pImageData allocate a memory, the memory is released again in the final exit, the intermediate process can be repeated using the buffer.

- Passive in image mode (callback)
  If you need to use a callback function in image processing, you need to set a good map reading callback function after the camera is initialized
  After CameraInit, call the following function settings callback.

  m_pFrameBuffer = CameraAlignMalloc(1280*1024*3,16);//It assumes 1.3 megapixel color camera
  CameraSetCallbackFunction(m_hCamera,GrabImageCallback,(PVOID)0,NULL);
  Wherein GrabImage Callback defined as follows:
  void _stdcall GrabImageCallback(CameraHandle hCamera, BYTE *pFrameBuffer, tSdkFrameHead* pFrameHead,PVOID pContext)
  {
  
  169
CameraSdkStatus status;
//Converts RAW data into image data for the specified format. The default into image data BGR24 format.
status = CameraImageProcess(hCamera, pFrameBuffer, m_pFrameBuffer,pFrameHead);
//After the conversion is successful, the use of data m_pFrame Buffer in subsequent processing.
}

5.4 using the display controls to preview images

In order to simplify the user's secondary development process, our internal SDK encapsulates the interface functions can easily preview image, the prerequisite is to use VS relevant interface development tools such as C #, VB, Delphi, MFC and the like.

After the camera is initialized, call the following code to initialize the display portion of the work.

Camera Display Init incoming hWnd window handle. Different development tools, there are different ways to get a handle hWnd displayed control.

Camera SetDisplaySize set the size of the display area, in pixels. Usually you can set to show the actual size of the control.

Camera SetDisplayMode setting mode display, there are two display modes and zoom tiled. The default display is the zoom. The image is scaled to the appropriate size is displayed on the display controls.

CameraSetDisplayOffset set the offset value display coordinates. When the size of the display is smaller than the dimensions of the image controls and CameraSetDisplayMode set tile mode, you can set the offset value of an area of the image displayed by the function.

Detailed usage over several functions, we offer a special routine, refer MindVision \ Demo \ VC ++ \ CameraDisplay routine.

5.5 Adjust the camera image brightness (exposure time set)

When the camera is shipped, in order to facilitate the demonstration effect, the default is automatic exposure, that is to say, the camera according to the brightness of ambient light, the camera automatically adjusts the exposure time and analog gain value to obtain the optimum image brightness. However, this automatic adjustment mode is not suitable for industrial applications requiring a few function calls to set the exposure and gain manually, to stabilize the brightness of the image to fit the needs of software algorithms, for example, in some cases, the user may want most of the image area overexposure to reduce interference image.
● CameraSetAeState, set the camera exposure mode is divided into manual and automatic modes.

● CameraSetExposureTime, set the camera's exposure time in microseconds. Note that the exposure time, the lower the camera's frame rate, for example, the exposure time is set to 500 ms, then one second imaging camera up to 2 times, the image looks will be more Caton. Therefore, in order to reduce the time taken diagram, the exposure time should be as low as possible, of course, low exposure time, you need an external light source to increase the light, otherwise the image will be very dark.

● CameraSetAnalogGain, set the analog gain value of the camera, the size of this value will only affect the image brightness, but will not affect the image frame rate, because only a circuit amplification factor, but will enhance the image of the background noise after the value increases, the pursuit of quality applications, analog gain should be set to a minimum.

Note, CameraSetExposureTime and CameraSetAnalogGain be set manually on the premise that by CameraSetAeState Set the camera to manual exposure mode, otherwise it is set invalid!

5.6 switch to a different resolution and a custom resolution (ROI function)

ROI explain the step value

5.7 Setting the camera's contrast, gamma, saturation, sharpness and other parameters ISP

● CameraSetGamma, Set the gamma value of the camera, the smaller the value of gamma, brightness will increase the value of smaller pixels, making the overall brightness of the CP; the larger the gamma value, will reduce the value gray pixels of high brightness, making the overall brightness becomes smaller. The default gamma value of 1.0.

● CameraSetContrast, set the camera's contrast value, the greater the contrast, the darker the area make the picture black and white more white areas, so that the image looks black and white is very clear, in a number of visual processing can effectively capture the outline; the other hand, if the contrast small, will be grays, it looks hazy. Contrast The default is 100, the minimum can be 1, up to 200.
• CameraSetSaturation, set saturation. The more concentrated the greater color saturation; whereas the more light, saturation. If set to 0, the image is completely without color, and is equivalent to the black and white camera. The default value is 100. The adjustment range is 0-200.

• CameraSetGain, gain setting value R, G, B three color channels.

• CameraSetSharpness, set the level of sharpening the image. The higher the sharpness, the better the image clarity, but also the greater the noise; conversely lower the sharpness of the image hazy sense is strong, but the noise is very low, it is very smooth. The default value is 0, that is, no sharpness enhancement effect.

5.8 color cameras switch to black and white camera

CameraSetMonochrome function can be color images into black and white, the effect is equivalent to the saturation of the image is set to 0, so the function can also be used CameraSetSaturation set saturation to zero to get a black and white gray scale image.

Note: Whether black and white or color cameras, the factory default is the output image data BGR24 format for black and white camera BGR24, B = G = R. If you only need to handle 8-bit grayscale image, call the following code after the initialization, the image CameraImageProcesss function output is the 8-bit grayscale.

//Initialize the camera
CameraInit....

//Setting Camera Image Process function output format is 8-bit grayscale
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO8);

//Subsequent to take part in Fig.
if(CameraGetImageBuffer(m_hCamera,&sFrameInfo,&pbyBuffer,1000) == CAMERA_STATUS_SUCCESS)
{
    CameraImageProcess(m_hCamera, pbyBuffer, m_pFrameBuffer,&sFrameInfo);//
    m_pFrameBuffer.Output buffer address pointed to, it holds the 8-bit grayscale image format up.
}

Or directly CameraGetImageBufferEx function, you can get to 8-bit grayscale images.
5.9 multiple cameras simultaneously, how to establish a correspondence between the camera

When the vision system using multiple cameras at the same time, the software needs to confirm to confirm the above good correspondence between the camera and inspection station between, but the camera is often scan order is not fixed, and the power-on sequence of the camera has relations, and therefore need some special software end identification. We recommend the following two ways to bind:

- In accordance with the camera's serial number. Before the camera is shipped has a unique serial number encoded. The serial number in the enumeration phase of the camera can be read by CameraEnumerateDevice function can scan to the number of cameras and detailed description array of structures of type tSdkCameraDevInfo, tSdkCameraDevInfo in acSn members on the preservation of the maximum 32 characters the serial number of the string section. You can determine the serial number of the software to select the appropriate tSdkCameraDevInfo array CameraInit call back according to customize the camera's name. By default, the name of the camera with the camera on the electrical changes in the order, such as the first power of the camera to the end of the model name plus # 0, for example, MV-GE500M # 0 and MV-GE500M # 1 denote system 2 camera name access, if the camera's power-up sequence has changed, then the original number 0 camera is likely to become the No. 1 camera. So we provide editing tools, you can manually edit the custom name for each camera. The custom name and serial number, is the enumeration phase can be enumerated, but also by CameraEnumerateDevice function can scan to the number of cameras and detailed description array of structures of type tSdkCameraDevInfo, tSdkCameraDevInfo in acFriendlyName members on the preservation of the maximum 32-byte device custom name. Follow choosing tSdkCameraDevInfo array basis of the name CameraInit calls. Since modifying the way the camera follows a defined name, nickname after an input device, click the Modify button, and then turn off the power to restart the camera once, custom name will come into effect, it will be permanently cured to
the inside of the camera.

Custom name modify the camera
Modify the name of the camera after a good, fast initialization may be performed by CameraInitEx2 function. Suppose the two cameras was renamed "Camera1" and "Camera2" code is as follows:

```c
//Scan camera, if there are two , iCameraCounts = 2
int iCameraCouts = CameraEnumerateDeviceEx();
//Initialization custom name "Camera1" camera, if not find the corresponding name of the camera, the initialization fails.
CameraInitEx2 ("Camera1",&hCamera1);
//Initialization custom name as "Camera2" camera, if not find the corresponding name of the camera, the initialization fails.
CameraInitEx2 ("Camera2",&hCamera2);
```

5.10 Flip and mirror image rotation 5.10

- Enable Flip image horizontally, CameraSetMirror(hCamera,0,1);
- Prohibit horizontal image flip (default), CameraSetMirror(hCamera,0,0);
- Enable the image flipped vertically,, CameraSetMirror(hCamera,1,1);
- Prohibited image flipped vertically (default)CameraSetMirror(hCamera,1,0);
- Image rotation 0 degrees (default), CameraSetRotate(hCamera,0);
- Rotate the image 90 degrees (default), CameraSetRotate(hCamera,1);
The image is rotated 180 degrees (default), CameraSetRotate(hCamera,2);
- Image rotated 270 degrees (default), CameraSetRotate(hCamera,3);

Note that after image rotation, image width and height dimensions will swap, for example, it turned out to be after 800X600, rotated 90 degrees, that is, the size of 600X800, when the process should pay attention to, otherwise it will cause an error, the image looks very confusing.

### 5.11 text is superimposed on the image function

In order to facilitate customers to overlay text on an image information suggests that we encapsulate a function CameraDrawText quick call. For usage of this function, we specialize in providing a routine located \Demo \ VC ++ under DrawText. Note that, CameraDrawText function overlay text is superimposed on the image content directly, in order not to affect visual processing, visual processing should be placed at the end of the process and then call (call before the image display function).

### 5.12 Save pictures and video features

- Image saving function. Use Camera Save Image, the image can be saved as one of RAW, PNG, JPG, BMP24bit, BMP8bit of.
- A typical process:

  1. Use CameraGetImageBuffer function takes the RAW image data;
  2. Use Camera Image Process RAW function will be converted into BGR24 or 8-bit grayscale format. If you want to save the RAW data format, skip this step.
  4. CameraSaveImage function to image data obtained CameraImageProcess function to save as JPG, BMP or PNG one of them. If RAW data is saved using CameraGetImageBuffer obtained RAW data buffer.

Examples of the image saved located \Demo \ VC ++ \ ImageFormat Saving

- Recording function. CameraInit Record function initializes the first video. CameraPushFrame a coded image. CameraStopRecord stop recording, the end of the write file operation.

Examples of video located \Demo \ VC ++ \ Record.
5.13 Setting the camera image output bit depth

By default, the camera industry is generally transmitted raw data 8bit format, in some applications a wide dynamic uplink, the user can change the camera functions CameraSetMediaType output of raw data bit depth. Currently we support the bit depth of 12, 16 bits have three options.

For example Camera SetMediaType (Camera, 0); selects the first seed, 8bit format Camera SetMediaType (Camera, 1); selecting the second species is 12bit, Camera SetMediaType (Camera, 2); third choice is 16bit.

Note: Not all models support 12 and 16 bit output mode. If CameraSetMediaType returns non-zero, it means the setting failure.

5.14 Setting pixel format of the image (8-bit grayscale, 24,32,48-bit color)

Unlike 5.13, said here pixel format, the final image obtained by the pixel format CameraImageProcess function, rather than the camera pixel format of the output of the original RAW.

Most types of industrial cameras can output 12bit or 16bit original RAW image, the color camera, if a color component with 8bit image detail is lost out of the low, so we provide RBG48bit color image format to meet the users to do high dynamic, high-precision visual analysis. Red, green, and blue color channels, each channel using 16bit to represent a pixel requires six bytes. When the camera supports 12bit or 16bit original RAW format, you can use 48bit color image format. 12bit RAW data format, the image is converted to 48bit color when low 4bit each channel will be filled with 0.

After the camera is initialized by calling CameraSetIspOutFormat to set up. camera set ISP out format (H camera, camera_Media_type_mono8);
//Once set, the output image is Camera Image Process 8bit grayscale image, and a pixel occupies one byte, in order of priority.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_MONO16);
//Once set, the output image is Camera Image Process 16bit grayscale image, and a pixel occupies two bytes, in order of priority.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGR8);
//Once set, the output image CameraImageProcess is 24bit BGR color image, and a pixel occupies 3 bytes, followed by blue, green, red this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGR16);
//Once set, the image is a color image 48bit BGR CameraImageProcess output a pixel occupies 6 bytes, followed by blue 16bit, 16bit green, red 16bit this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGRA8);
//Once set, the output image CameraImageProcess is 32bit BGRA color image, and a pixel occupies 3 bytes, followed by blue, green, red, Alpha this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_BGRA16);
// Once set, the output image CameraImageProcess is 64bit BGRA color image, and a pixel occupies 8 bytes, followed by blue 16bit, green 16bit, red 16bit, Alpha16bit this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGB8);

// Once set, the output image CameraImageProcess is 24bit RBG color image, and a pixel occupies 3 bytes, followed by red, green and blue so arranged.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGBA16);

// Once set, the output image CameraImageProcess is 48bit RBG color image, and a pixel occupies 6 bytes, followed by red 16bit, 16bit green, blue 16bit this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGBA8);

// Once set, the output image CameraImageProcess is 32bit RBGA color image, and a pixel occupies four bytes, followed by red 8bit, green 8bit, blue 8bit, Alpha8bit this arrangement.
CameraSetIspOutFormat(hCamera, CAMERA_MEDIA_TYPE_RGBA16);

For 16 bit (1 channel 2 bytes) of data, we use the little-endian mode memory, 1 byte is low 8bit, byte 2 is a high 8bit.

Usage of this function, you can refer to the routine: \Demo \ VC ++ \ ImageFormat & Saving

5.15 pairs of the original RAW image processing

In general, industrial cameras are the original RAW transmission of data, for black and white camera is grayscale image data for color cameras, image data is Bayer format.
CameraGetImageBuffer function, is the original RAW data obtained, the default is 8bit if needed 12,16bit format, please refer to 5.13 method. Note CameraGetImageBuffer CameraReleaseImageBuffer needs and supporting the use of.

5.16 camera network interface using the API to dynamically set IP, gateway, subnet mask

When using the network camera, you must first set up the camera and the network card to connect to the same subnet segment, so as to carry out normal
communications. We offer a dedicated Gigabit Ethernet IP setting tool (located in the software installation directory TOOLS folder).

At the same time users can also CameraGigeSetIp function dynamically modify the IP camera parameters in the program. Please refer to the specific usage routines: Demo\VC++\GigeConfig.

5.17 Set the camera's frame rate

By CameraSetFrameSpeed function, the camera can dynamically set the output frame rate. CameraSetFrameSpeed is to set the speed gear, we will be divided into high, medium, low and other several models of different types of cameras without high school low speed mode corresponds to a specific frame rate is not the same.

CameraSetFrameSpeed ( hCamera, 0 ) ;//It is set to the low-speed mode.

CameraSetFrameSpeed ( hCamera, 1 ) ;//Set to medium speed mode.

CameraSetFrameSpeed ( hCamera, 2 ) ;//Set to high-speed mode.

Usually the camera at the factory, the default rate is the highest, if desired deceleration run, you can call CameraSetFrameSpeed deceleration function.

In addition, some camera models support only two speed mode, some support 4, the number of specific support mode, you can CameraGetCapability function to give a description tSdkCameraCapbility structure of the camera, tSdkCameraCapbility structure of iFrameSpeedDesc members, indicating that the current the number of camera models supported speed mode.

5.18 Trigger soft or hard (external) triggering

- Set the camera to take continuous graph pattern. The factory default camera is in continuous mode to take diagram, can also CameraSetTriggerMode ( hCamera, 0 ); switched to continuous mode to take FIG.

- Mode ( hCamera, 1 ); switch to take the soft trigger mode Fig. After entering this mode, the camera image capture and stop transmitting. Only when the user invokes CameraSoftTrigger ( hCamera ) once, the camera will capture the image once sent up. After finished entering the camera will wait until the next time the user calls

- Mode ( hCamera, 1 ); switch to take the soft trigger mode Fig. After entering this mode, the camera image capture and stop transmitting. Only when the user invokes CameraSoftTrigger ( hCamera ) once, the camera will capture the image once sent up. After finished entering the camera will wait until the next time the user invokes CameraSoftTrigger ( hCamera ).
Set the camera to take a hard trigger mode (hCamera, 2); switches to take a hard trigger mode (hCamera, 2). After entering this mode, the camera image capture and stop transmitting. Only when the user inputs a pulse on the trigger terminal on the camera housing, the camera will capture the image once sent up. After finished the camera will enter a waiting state until the next trigger pulse is received.

If you need a trigger multiple images, you can CameraSetTriggerCount function set, the default is a trigger to get a picture. If you need a trigger to get N images, you can call CameraSetTriggerCount (hCamera, N) is set.

5.19 To signal external trigger mode to shake down

In order to avoid the interference of external signals to trigger an impact, especially when using a mechanical external trigger signal to the camera, there will be a lot of signal interference and jitter, likely to cause false positives. So you can use CameraSetExtTrigJitterTime function, set the debounce time to increase the capacity of the system to carry interference. The default time is 0, it means do not shake in the electronic switching signal is clean when not to shake the work. Set debounce time, it will introduce trigger delay, for example, 10 ms debounce time, the trigger delay will be delayed at least 10 milliseconds.

5.20 External trigger mode Set the trigger delay time

If the trigger signal is issued to the camera needs some time delay before taking pictures, you can use the trigger delay mode. Typical applications, such as using a proximity switch as a trigger source, but when the proximity switch signals the objects are not in place or is still in an unstable mode jitter, this time need a delay before triggering photographs. CameraSetTriggerDelayTime function is used to set the external trigger delay, in microseconds.

5.21 detection camera dropped and automatically reconnect

In some extreme conditions, the camera may be at risk of dropping such as USB camera in the host computer when the power supply is unstable, or vibration is relatively large, loose USB port and so on.

By default, the Division I USB and Gigabit Ethernet cameras have dropped automatically reconnect function, internal SDK has integrated this functionality. If you need real-time monitoring dropped the case, you can call the CameraConnectTest function periodically query.

CameraConnect Test function to search.
5.22 to read and write the serial number of the camera

The camera comes with three serial number, a serial number which is read-only, can not be modified, the factory had already fixed up. Secondary and tertiary serial number can be read freely. The serial number of each level is 32 bytes.

CameraReadSN, Read the serial number.

CameraWriteSN, Write the serial number.

Usage of this function refer to the installation directory \ Demo \ VC ++ \ UserDataTest routine.

5.23 to read and write custom data in the camera

All our models of cameras, all with a piece of data storage space, users can freely read and write. And after power is saved without loss of data. The camera, the size of the storage space of each model is not the same, the specific size can be obtained by CameraGetCapability function, iUserDataMaxLen tSdkCameraCapability structure indicates the maximum number of bytes stored in the camera.

CameraSaveUserData, write a piece of data to the camera. After power is still preserved.

CameraLoadUserData, read a piece of data from the camera.

Usage of this function refer to the installation directory \ Demo \ VC ++ \ UserDataTest routine.

5.24 Get a string describing the error code corresponding to the information

CameraGetErrorString

For example CameraInit function returns an error code is -45, then
char * s = CameraGetErrorString (-45) will get a pointer to the string, if the printf function to print out the words, it will show:
"Forbidden to specify the camera is already in use by another program, and then apply for access to the camera will return to the state (a camera can not be used by multiple programs simultaneously access).";
6 GigE Vision and USB3 Vision’s XML file defines interpretation

For a standard protocol interface development Vision customer, device enumeration phase, the camera needs to resolve the corresponding XML configuration file, this section we offer a special file for explanation, please refer to the installation directory under the Document folder "USB3, GigE Vision interface development and XML parsing .pdf "
7 Halcon Development Guidance

Halcon is the world’s most versatile machine vision software. Halcon users around the world from a flexible framework for rapid development of image analysis and machine vision benefit program. Halcon has provided more than 1,100 kinds of libraries with outstanding Performance of the controller, such as fuzzy analysis, morphology, pattern matching, 3D correction. Halcon supports multiple operating systems.

Our camera interface has been optimized specifically for Halcon settings can be well supported Halcon. Halcon has been installed in the system, then we run the camera software installation package MindVision Platform Setup (x.x.x.x) .exe installer, the installer will automatically detect the installation path Halcon, and its corresponding settings. At present, our camera development kits support Halcon8, Halcon9, Halcon10, Halcon11, Halcon12 32-bit and 64-bit versions.

After using the USB cable to connect the camera to a PC, you can begin to develop, and the following instructions for MV-U300 camera models as an example, when the development of other types of cameras, only need to modify the model name.

7.1 HDevelop in development

Start Halcon, click on the "assistant" menu, select "Open New Image Acquisition", shown in Figure 7.1.

![HDevelop Image acquisition interface](image)

7.1.1

Figure 7.1 Halcon Image acquisition interface

1. Click "image acquisition interfaces (IP)", in the drop-down list, locate MindVision. As shown in Figure 7.2. In Figure 7.2 1. Click "Connect" tab, the window in Figure 7.3. In Figure 7.3, we can see that our MV-U300 camera. If both multiple MindVision camera, select the camera you want to access the device list.
2. Click Figure 7.3 "Connecting (n)" button, after a successful connection, the "Parameters" tab is activated, shown in Figure 7.4. At this point you can choose the color space and bit depth.

- Our interface supports RGB24 and Gray two color spaces when selecting RGB24, the image output is RGB888 format, suitable for color cameras,
when you select Gray, the output image format is 8-bit grayscale for black and white camera.

- 8, 10, 12-bit bit depth support three kinds (only partially supported by the camera. 10, 12-bit bit depth, no frame buffer series only supports 8-bit depth).

Figure 7.4 Halcon Successful connection MV-U300

- parameter settings. Click on "Parameters" tab page button to switch to the parameter configuration page, as shown in Figure 7.5.
- bits_per_channel: You can select the bit depth per channel under;
- grab_timeout: Capture timeout, in milliseconds, the default is 200 milliseconds timeout; note that within the timeout period, if the image does not come, acquisition thread will be blocked until the time-out or read to the end of the effective image frame. Can be set by set_framegrabber_param (AcqHandle, 'grab_timeout', 200) Mining FIG timeout milliseconds.
- camera_play: When clicked, start or continue image capture function;
- camer_pause: Click, pause the image capture function;
- show_camera_settings: When clicked, display MV-U300 camera's configuration page, as shown in Figure 5.6. In halcon program, you can display the camera's configuration page by set_framegrabber_param (AcqHandle, 'show_camera_settings', 1).
- software_trig: After clicking, if the camera is in the soft trigger mode, the trigger the camera capture an image and sent to the host. If the camera does not work in software trigger mode, the button click is
invalid. In halcon program, you can set_framegrabber_param (AcqHandle, 'software_trig', 1) is sent to the camera a soft trigger command.

- **shutter**, Setting the camera's exposure time in microseconds.
- **contrast**, Setting the camera's contrast. In the range of 0-200, the default is 150.
- **gamma**, Setting the camera's gamma value in the range of 0-1000, the default is 50, corresponding to the gamma factor of 0.5.
- **exposure_mode**, Select the exposure mode. 0 indicates manual mode; 1 indicates the automatic mode. Only when exposure_mode set to 0, shutter and gain adjustment parameters to take effect.

- **trigger_mode**, select the camera mode. 0 indicates continuous acquisition mode; 1 represents a soft trigger mode; 2 represents an external hardware trigger mode. In the program, you can set_framegrabber_param (AcqHandle, 'trigger_mode', 2) let the camera into the external trigger mode, this time as long as the external input signal to trigger the camera on pins, the camera will begin to capture an image transmission to PC, without triggering signal, grab_image_async (Image, AcqHandle, -1) function returns a timeout.

- **frame_speed**, select camera acquisition speed, 0 represents the slowest, 1 medium, 2 denotes the highest. Only in the highest part of the camera speed mode. Acquisition slower, the longer the camera's maximum exposure time if takes a long time exposure, set frame_speed set to 0. halcon program, you can change the camera acquisition speed by set_framegrabber_param (AcqHandle, 'frame_speed', 0) call.

- **resolution**, Select the default resolution. 0 means select the first default resolution, 1 select the default resolution of the second, and so on. Dynamically switches.

- **color_temperature**, Select the color temperature mode (black and white camera for change parameter is invalid). 0 indicates that the color temperature of about 2700K, 1 represents the color temperature of 4200K, 2 represents the color temperature of about 5500K, 3 indicates a color temperature of about 6500K.

- **red_gain**, Digital gain setting red channel. In the range of 0-4 for color fine-tuning. For black and white camera, this parameter is meaningless.

- **green_gain**, Set digital gain green channel. In the range of 0-4 for color fine-tuning. For black and white camera, this parameter is meaningless.

- **blue_gain**, Set digital gain green channel. In the range of 0-4 for color fine-tuning. For black and white camera, this parameter is meaningless.
- **saturation**, the camera color saturation. Valid only for color cameras, ranging from 0 to 200, is set to 0, the image completely lose color and become black and white images, set to 200, the most beautiful color, the default is 100.
- **gain**, set the camera's analog gain, the parameters and shutter parameters together, determines the brightness of the image. Only when exposure_mode set to 0, gain shutter manually and parameters to adjust.
- **color_space**, the output image format. Select Gray, the output will be 8 grayscale images, select RGB24, the output 24-bit color images. Black and white camera can also choose RGB24 format, but R = G = B; color camera can also choose 8-bit grayscale image output.
- **color_space**, format image output. Select Gray, the output will be 8 grayscale images, select RGB24, the output 24-bit color images. Black and white camera can also choose RGB24 format, but R = G = B; color camera can also choose 8-bit grayscale image output.
- **GPO0,GPO1,GPO2,GPO3** This corresponds to 4 4 output type IO status on the camera control, set_framegrabber_param (AcqHandle, 'GPO0', 1) is set to OUTPUT IO state number 0 is high, set_framegrabber_param (AcqHandle, 'GPO0', 0) set number 0. OUTPUT IO status is low; set_framegrabber_param (AcqHandle, 'GPO1', 1) is set to OUTPUT IO No. 1 status is high, set_framegrabber_param (AcqHandle, 'GPO1', 0) is set numbered 1 the OUTPUT IO status is low; the other numbered IO and so on.
- **GPI0,GPI1**, This corresponds to 4 4 output type IO status on the camera control, set_framegrabber_param (AcqHandle, 'GPO0', 1) is set to OUTPUT IO state number 0 is high, set_framegrabber_param (AcqHandle, 'GPO0', 0) set number 0. OUTPUT IO status is low; set_framegrabber_param (AcqHandle, 'GPO1', 1) is set to OUTPUT IO No. 1 status is high, set_framegrabber_param (AcqHandle, 'GPO1', 0) is set numbered 1 the OUTPUT IO status is low; the other numbered IO and so on.
Figure 7.5 Halcon The camera parameter settings page

For image acquisition, Halcon image acquisition into a single acquisition and real-time acquisition in two ways. Figure 7.4 Click the "collection (S)" button to conduct a single acquisition, within the post-acquisition image now Halcon window, as shown in Figure 7.7; Figure 7.4 Click the "Live (v)" button for continuous acquisition, Halcon It will continuously display the image was captured.
Figure 7.6 Halcon Displayed in the camera settings window

Figure 7.7 Halcon The acquired images

7.2 C/C++, VB, C# development conducted Halcon

If you need to use C / C++, VB, C# and other languages Halcon development, only you need to generate the appropriate code to the project and the HDevelop. Specific approach is: In Figure 7.5, click the "Code Generation" tab, as shown in Figure
7.8:

In 7.8, you can make the appropriate configuration, or click on "Insert Code" button, you can get HDevelop can compile and run the code in Figure 7.9.

Figure 7.8 Halcon The code generation configuration
Figure 7.9 Code is automatically generated HDevelop

Next, the code is shown in Figure 7.9 is converted to C / C ++, VB, Delphi, C # project. Click "File" menu, select "Export", as shown in Figure 7.10.

And then in the Export dialog box pops up, set up, as shown in Figure 7.11, the drop-down list, select the type of language that you want to export, then "Export" button. After the export is complete, you can set up in your path, find the project file corresponding to the language, the compiler can directly open the run.

Figure 7.10 Export other development languages
7.3 Simultaneous use of multiple camera development

When connecting two cameras simultaneously, in order to better distinguish the correspondence between each of the cameras, we recommend using the following processes:

1. Use our demo software, modify the camera equipment nickname, for example, the first name of the camera modified to Camera1, the second name of the camera modified to Camera2, modification is completed, no matter which of the 2 cameras connected to USB or network interface, which will be the name Camera1 and Camera2, remain valid after replacing the computer. (Modify camera device name is shown in Figure 5.12)

2. Halcon device initialization, respectively, fill in the device name "Camera1" and "Camera2", as follows: 
   
   ```
   open_framegrabber ('MindVision', 1, 1, 0, 0, 0, 0, 'progressive', 8, 'Gray', -1, 'false', 'auto', 'Camera1', 0, -1, AcqHandle)
   open_framegrabber ('MindVision', 1, 1, 0, 0, 0, 0, 'progressive', 8, 'Gray', -1, 'false', 'auto', 'Camera2', 0, -1, AcqHandle).
   ```

![Figure 7.11 Export settings](image)
Figure 7.12  Modify camera name
8 The LABVIEW Development Guidance

In Labview by NI MAX can we provide the tools and routines based DLL calls two ways for development.

8.1 Secondary Development BY LABVIEW

1. First, connect the camera and then run after NI MAX. Can under "Devices and Interfaces" of NI-IMAQdx Devices, find our camera, as shown in Figure 8.1. Double-click, you can get a preview of the camera as shown in Figure 8.2 (after Figure 8.2 need to click the Grab button above to preview).

2. In Figure 8.2, the resolution can be switched.

3. Click the Camera Attributes Figure 8.2 label, the camera can be set other parameters. As shown in Figure 8.3. Under this interface, due to the agreement that only allows you to set some parameters of the camera, set other parameters, you can through our demo software, adjusted, preserved (camera parameters can be saved into a file, no matter which you use ways to develop the parameter file can be loaded automatically valid), then NI MAX open, the same effect.

![NI MAX Equipment List](image)

Figure 8.1 NI MAX Equipment List
Figure 8.2 Preview Interface
8.2 Base developed through DLL calls

The routine way of the installation directory DEMO / Labview / useDLL folder, demonstrates how to call the SDK DLL file (MVCAMSDK.dll) for development. The routine has the following characteristics:

1. Support monochrome and color cameras automatically recognize the camera in black and white using grayscale image Figure U8 format for display; a color camera Figure U32 format color image display.
2. Support the use of multiple cameras simultaneously developed. CameraInitEx only need to call the interface to the first parameter is incremented. Enter 0 initialize the first camera, the second initialization input 1 indicates the camera, all the rest of the VI source code can be copied directly.
3. The routine can be applied to the Division I all camera models, regardless of the size of the resolution, trigger mode, how to transfer interfaces, the routine can be automatically recognized without the need to manually modify any of the variables.
4. The parameters of the camera supports saving and loading (file mode). Modify good camera parameter file, this routine can also be loaded by other software, enter the code to reduce the manual workload.

8.3 Using multiple the camera in the LABVIEW

The routine way of the installation directory DEMO / Labview / directory under TwoCameras, this routine is also based DLL methods, demonstrates how to use multiple cameras in Labview. Routine is given a method using two cameras simultaneously, two cameras can be of the same type, or may be of different types, or even a plurality of different camera interfaces, such as a USB2.0,1 a USB3.0,1 GIGE a camera, you can use this approach to develop multi-camera. When using two or more cameras at the same development, please refer to the routine way to extend and number camera SDK is currently limited to 64, when the need to use more than 64 cameras at the same time, please contact our Technical Support You can further expand the number of multi-camera support.

8.4 Multi-camera distinguish LABVIEW

When developing when using multiple cameras, often require correspondence between each camera, different cameras will be used to perform different tasks. Division Multiple camera There are many ways, through the camera a unique serial number, the name of the camera, the camera from a variety of ways to define data. In Labview, we offer the following two ways to distinguish between multiple cameras:

- the use of custom data. We provide an interface, you can write custom data in the camera, you can use these data to distinguish between different cameras. However, this method must be initialized after the camera has to be read to the custom data, so Labview programming must be in the camera after initialization is complete, and then through a custom interface to obtain data to determine which one camera, then make the appropriate branch processing. (Routine already provides read-write Camera Custom data)

- use a custom device name of the function. The process and section 5.3 Halcon distinguish multi-camera similar. The first step, as shown in Figure6.4, respectively 2 camera equipment into the name Camera1 and Camera2. The second step, in Labview, call CameraInitEx2 interface to initialize the camera's, were passed in the first argument string "Camera1" and "Camera2". After modification, the name was cured to the inside of the camera, permanent, without interfaces, computer replacement of. The routine is located in the installation directory DEMO / Labview / TwoCamerasEx directory.
Figure 8.4 Modify camera nickname
9 When integrated the camera installation files released

When you integrate our camera, the camera when you need to be packaged and released the SDK documentation, you can follow the following ways:

1, the file needs to be packaged under our camera installation directory has the following folders:

- ./SDK directory, which includes all of the SDK files.
- ./Drivers directory that contains the camera's kernel driver file.
- ./Demo directory that contains the camera's development DEMO, that you can publish as needed, or no release Demo program.
- ./Camera/Configs directory under the corresponding camera profiles. The directory on your computer to save your current configuration parameters of the camera, you can publish it packaged together, the parameter file will be loaded automatically. If you do not release the camera profile, use the camera for the first time, will use the default parameters, and generates a file in the Camera / Configs directory. Please send the file and you develop into exe program under the same directory.

2, release mode. You can choose to publish the following three ways.

- use our platform installation package installed directly. MindVision Platform setup (version number) .exe, but the package will contain company information.
- use the tools we provide. In the Setup folder, we offer a program called "Installer.exe" program, whose role is all the files SDK and Drivers directory for installation. The SDK, Drivers, Setup folder copied to the target machine, and then run the Setup folder in the "Installer.exe" to automatically complete the installation, as shown in Figure (MVDCP.exe and Measure.exe we provide presentation software and measurement software, you can not publish, after running Installer.exe, MVDCP and measure software can properly run):
manually or write your own installer. First, copy the Drivers directory to the target machine and then manually install the kernel device driver (USB camera when the camera into the computer, you will be prompted to install the driver, select the appropriate INF file directory under Drivers can, Gige cameras need to manually run Drivers\Gige under MvDriverInstall.exe program can complete the installation of the kernel driver); and the SDK (if it is X64 systems, copy the SDK / X64 folder of all files (Note: the folder of files, not folders) to any directory of the target machine (assuming C: \ TEST directory), then you are a good secondary development of executable files into the C: \ TEST directory, you can run directly, without having to enter another installation.
10 Technical Support

You have any questions about the use and development process, please contact our technical support as soon as possible. We recommend that you try to communicate with us by e-mail, fax and other written form, and attach a detailed description of the issue, the cross-sectional Figure and other information, which will help us to quickly resolve your problem.

Your success is our success, your satisfaction is our satisfaction. Any suggestions that you are back on our recognition and encouragement. We look forward to your feedback, we will strive to do better!