QHYCCD SDK Introduction Manual

Ⅰ.SDK 介绍

QHYCCD SDK is a C++ library， it contains all control functions,you can use these functions to control camera.You can develop software via SDK,when do develop,you need use different method to call SDK APIs.Also,you need note software bits to choose right bits SDK,32 bits software use 32 bits SDK,64 bits software use 64 bits SDK.

You can download SDK from our official website:https://www.qhyccd.cn/html/prepub/log.html#!log.md

The page recorded newest stable version SDK.The download file is a zip file,it includes header files and library files,x86 folder include 32 bits SDK,x64 folder include 64 bits SDK,ini file is configure file,you can use it to configure camera function.

In addition to qhyccd.dll, the compressed package contains several additional files, which are qhyccd.exp, qhyccd.lib, ftd2xx.dll, tbb.dll, winUSb.dll, msvcp90. DLL and msvcr90. DLL.Qhyccd. exp is the SDK export library file, most of the time is not used, can be ignored;Qhyccd. lib is a statically linked library that needs to be used when developing in a statically linked way.Ftd2xx. DLL is the library of FTD chip, which can be ignored because it is not used in most cases.Tbb.dll is a C# development library that can be placed in the same directory as qhyccd.dll if necessary.Msvcr90. DLL and msvcp90. DLL are VC++ runtime libraries that can be copied to the program directory if the program indicates that it is missing.

Ⅱ.API Introduction

Most API functions in SDK will determine whether the function is successfully executed by returning a value that is a macro defined in qhyccderr.h. The return value is usually QHYCCD_SUCCESS or QHYCCD_ERROR, with values of 0 and -1 respectively.Some functions have no return value or other type of return value. These functions are specified later.

1.uint32_t InitQHYCCDResource(void);

Function Introduction:

Initialize SDK resources, return QHYCCD_SUCCESS if the function succeeds.Call only once at the beginning of the program, not multiple times, which may cause the program to crash.

Sample Code:

```
uint32_t ret = QHYCCD_ERROR;
ret = InitQHYCCDResource();
if(ret == QHYCCD_SUCCESS)
{
    printf("Initialize QHYCCD resource success.\n");
```

```
}else{
	printf("Initialize QHYCCD resource fail.\n");
}
```

2.uint32_t ScanQHYCCD(void);

Function Introduction:

Scan the connected QHYCCD camera, and return the number of devices scanned after completion.

Sample Code:

```
int num = 0;
num = ScanQHYCCD();
if(num > 0)
{
	printf("%d cameras has been connected\n",num);
}else{
	printf("no camera has been connected\n");
}
```

3.uint32_t GetQHYCCDId(uint32_t index,char *id);

Parameter Introduction:

index: QHYCCD Device index value in the device list. The value range is determined by the return value of the number of cameras;

id: The variable used to store the camera ID;

Function Introduction:

The obtained ID is stored in the character array. This ID can be used to open the camera device and get the camera handle. QHYCCD_SUCCESS is returned on success.The ID of each camera consists of the camera model and serial number.For example, qHY183C-C915484FA76EA7552, the front QHY183C is the camera model, and the back C915484FA76EA7552 is the camera serial number.Each camera has its own unique serial number, even the same model of camera will use a different serial number, and the serial number is normally fixed.

Sample Code:

```
int i,ret;
char id[32] = {0};
for(i = 0;i < camNum;i++) //camNum is the return value of ScanQHYCCD
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("Found connected camera,the id is %s\n",id);
	}
	else
	{
```

```
            printf("some errors occered!(%d %d)\n",i,ret);
        }
}
```

4.uint32_t OpenQHYCCD(char *id);

Parameter Introduction:

id: The camera ID can be obtained by using the GetQHYCCDId function.

Function Introduction:

You can open the camera by using the camera ID and return to the device handle of the camera. You can control the camera function by using this handle.If the handle is not empty, the function is successfully executed.

Sample Code:

```
qhyccd_handle *camhandle = NULL;
camhandle = OpenQHYCCD(id);
if(camhandle != NULL)
{
    printf("Open QHYCCD success!\n");
}else{
    printf("Open QHYCCD failed!\n");
}
```

5.uint32_t CloseQHYCCD(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Turn off the camera and disconnect the camera.QHYCCD_SUCCESS is returned successfully.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = CloseQHYCCD(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Close camera success.\n");
}else{
    printf("Close camera failed.");
}
```

6.uint32_t ReleaseQHYCCDResource(void);

Function Introduction:

Release the camera's resources, and return QHYCCD_SUCCESS if the function succeeds.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = ReleaseQHYCCDResource();
if(ret == QHYCCD_SUCCESS)
```

```
{
	printf("Release QHYCCD resource success.\n");
}else{
	printf("Release QHYCCD resource failed.\n");
}
```

7.uint32_t GetQHYCCDNumberOfReadMode(qhyccd_handle *h,uint32_t *numModes);

Parameter Introduction:

handle: Device handle of the camera;

numModes: a variable used to store the number of camera readout modes;

Function Introduction:

Get the number of readout modes of the camera. Each camera has at least one readout mode. Different readout modes have different names.For details, please contact our technical support to understand the differences and characteristics of different readout modes for different cameras.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t ret, numModes;
ret = GetQHYCCDNumberOfReadMode(camhandle, &numModes);
if(ret = QHYCCD_SUCCESS)
{
	printf(“Set stream mode success!\n” );
}else{
	printf(“Set stream mode failed!\n” );
}
```

8.uint32_t GetQHYCCDReadModeName(qhyccd_handle *h, uint32_t modeNumber, char* name);

Parameter Introduction:

handle: Device handle of the camera;

modeNumber: Number of the camera readout mode, its value range depends on the number of readout modes;

name: stores a variable that reads the schema name;

Function Introduction:

Gets the name of the read mode.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t ret = QHYCCD_ERROR;
char name[80] = { 0 };
for(int i = 0; i < numModes; i ++)
```

```
{
	ret = GetQHYCCDReadModeName(camhandle, i, name);
	if(ret == QHYCCD_ERROR)
	{
		printf("Current read mode is %d,its name is %s.\n", i, name);
	}
}
```

9.uint32_t GetQHYCCDReadModeResolution(qhyccd_handle *h, uint32_t modeNumber, uint32_t* width, uint32_t* height);

Parameter Introduction:

handle: Device handle of the camera;

modeNumber: Number of the camera readout mode, its value range depends on the number of readout modes;

width: a variable used to store the width of the image in a readout mode;

height: a variable used to store the height of the image in a readout mode;

Function Introduction:

Used to obtain the image size of the camera in different readout modes. The image size of most cameras is fixed, but individual cameras will use different resolutions in different readout modes, such as QHY42PRO, QHY2020, QHY294PRO.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t ret = QHYCCD_ERROR;
uint32_t width,height;
for(int i = 0; i < numModes; i ++)
{
	ret = GetQHYCCDReadModeResolution(camhandle, i, width, height);
	if(ret == QHYCCD_ERROR)
	{
		printf("Current read mode is %d,its resolution is %d x %d.\n", width, height);
	}
}
```

10.uint32_t SetQHYCCDReadMode(qhyccd_handle *h, uint32_t modeNumber);

Parameter Introduction:

handle: Device handle of the camera;

modeNumber: Indicates the number of the read mode. Its value range depends on the number of read modes.

Function Introduction:

Set the read mode of the camera, return QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t ret = QHYCCD_ERROR;
ret = SetQHYCCDReadMode(camhandle, 0);
if(ret == QHYCCD_SUCCESS)
```

```
{
    printf("Set Read Mode successfully.\n");
}
```

11.uint32_t SetQHYCCDStreamMode(qhyccd_handle *handle, uint8_t mode);

Parameter Introduction:

handle: Device handle of the camera;

mode: Working mode of the camera. When the value is 0, it is single frame mode, and when the value is 1, it is continuous mode.

Function Introduction:

Set the working mode of the camera. You can set single frame or continuous mode.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
//setup single mode
int ret = QHYCCD_ERROR;
ret = SetQHYCCDStreamMode(camhandle,0);
if(ret = QHYCCD_SUCCESS)
{
    printf("Set stream mode success!\n" );
}else{
    printf("Set stream mode failed!\n" );
}
//setup live mode
ret = SetQHYCCDStreamMode(camhandle,1);
if(ret = QHYCCD_SUCCESS)
{
    printf("Set stream mode success!\n" );
}else{
    printf("Set stream mode failed!\n" );
}
```

12.uint32_t InitQHYCCD(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Initialize camera parameters and SDK resources. Different cameras require different initialization times. QHYCCD_SUCCESS is returned when the initialization is successful.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_SUCCESS)
{
```

```
    printf("Init QHYCCD success!\n");
}else{
    printf("Init QHYCCD failed!\n");
}
```

13.uint32_t GetQHYCCDFWVersion(qhyccd_handle *handle, uint8_t *buf);

Parameter Introduction:

handle: Device handle of the camera;

buf: stores a variable of the firmware version number.

Function Introduction:

To obtain the version of the camera driver (firmware), the version is a date, such as 18-3-30, if the function is successfully executed, return QHYCCD_SUCCESS.At present, QHYCCD cameras are divided into WINUSB and CYUSB cameras. There are some differences in the calculation methods of firmware versions of these two cameras. Please refer to the following example codes for details.

Sample Code:

```
int ret = QHYCCD_ERROR;
unsigned char fwv[32];
ret = GetQHYCCDFWVersion(camhandle,fwv);
if(ret == QHYCCD_SUCCESS)
{
    if((fwv[0] >> 4) <= 9)
    {
        printf('Version:20%d-%d-%d', ( (fwv[0]>>4)+0x10,fwv[0]&~0xf0,fwv[1]); //WINUSB cameras
    }
    else
    {
        printf('Version:20%d-%d-%d',fwv[0]>>4,fwv[0]&~0xf0,fwv[1]); //CYUSB cameras
    }
}
```

14.uint32_t GetQHYCCDSDKVersion(uint32_t *year, uint32_t *month, uint32_t *day, uint32_t *subday);

Parameter Introduction:

year: The year of release date;

month: The month of release date;

day: The day of release date;

subday: The subversion;

Function Introduction:

Obtain the SDK version number, that is, the version of qhyccd. DLL. Subday is the subversion number, which is

used to distinguish multiple versions in the same day. You can determine whether the SDK in use is the latest version based on the obtained SDK version.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
uint32_t year,month,day,subday;
ret = GetQHYCCDSDKVersion(&year, &nonth, &day, &subday);
if(ret == QHYCCD_SUCCESS)
{
	printf("%d-%d-%-,%d\n",year,month,day,subday);
}
else
{
	printf("Get QHYCCD SDK version failed.\n");
}
```

15.uint32_t GetQHYCCDChipInfo(qhyccd_handle *h, double *chipw, double *chiph, uint32_t *imagew, uint32_t *imageh, double *pixelw, double *pixelh, uint32_t *bpp);

Parameter Introduction:

handle: Device handle of the camera;

chipw: Chip width, in millimeters;

chiph: chip height, in millimeters;

imagew: Image width, unit pixel;

imageh: image height, unit pixel point;

pixelw: Pixel width, in microns;

pixelh: Pixel height, in microns;

bpp: Image data bit depth;

Function Introduction:

Get the chip information of the camera, including chip size, maximum resolution of the image, pixel size and image data Bit depth.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
int w,h,bpp;
double chipw,chiph,pixelw,pixelh;
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo success!\n");
	printf("CCD/CMOS chip information:\n");
```

```
        printf("Chip width : %3f mm\n",chipw);
        printf("Chip height : %3f mm\n",chiph);
        printf("Chip pixel width : %3f um\n",pixelw);
        printf("Chip pixel height : %3f um\n",pixelh);
        printf("image width : %d\n",w);
        printf("image height : %d\n",h);
        printf("Camera depth : %d\n",bpp);
}else{
        printf("GetQHYCCDChipInfo failed!\n");
}
```

16.uint32_t GetQHYCCDEffectiveArea(qhyccd_handle *handle, uint32_t *startX, uint32_t *startY, uint32_t *sizeX, uint32_t *sizeY);

Parameter Introduction:

handle: Device handle of the camera;

startX: X coordinates of the starting position of the effective area;

startY: Y coordinate of the starting position of the effective area;

sizeX: Width of the effective area;

sizeY: Height of the effective area;

Function Introduction:

This function outputs the valid size and starting position of the image, or QHYCCD_SUCCESS on success.The starting position takes the first pixel in the upper left corner of the image as the coordinate (0,0) reference point.In addition, because some cameras do not have an overswept area, the effective area is the same size as the image.

Sample Code:

```
int startx,starty,sizex,sizey;
int ret = QHYCCD_ERROR;
ret = GetQHYCCDEffectiveArea(camhandle,&startx,&starty,&sizex,&sizey);
if(ret == QHYCCD_SUCCESS)
{
        printf("Get camera effective area success.\n");}
else{
        printf("Get camera effective area failed.\n");
}
```

17.uint32_t GetQHYCCDOverScanArea(qhyccd_handle *h, uint32_t *startX, uint32_t *startY, uint32_t *sizeX, uint32_t *sizeY);

Parameter Introduction:

handle: Device handle of the camera;

startX: X coordinates of the initial position of the image oversweep area;

startY: Y coordinate of the initial position of the image oversweep area;

sizeX: Width of image oversweep area;

sizeY: Height of image overscanning area;

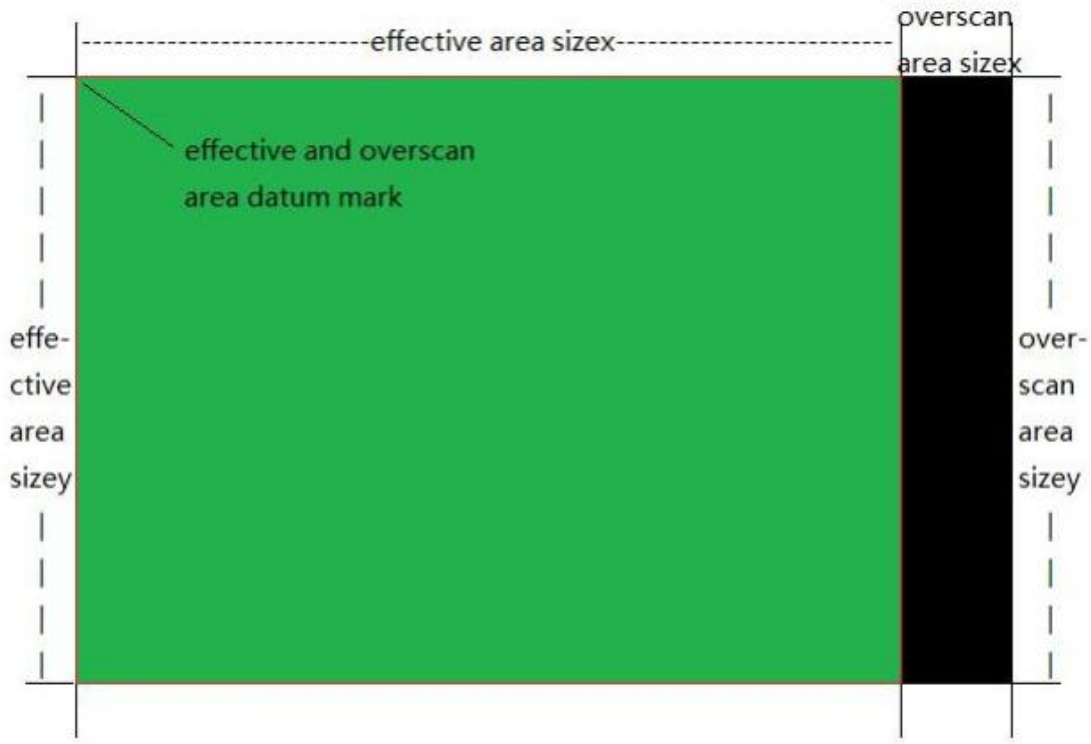
Function Introduction:

This function prints the size and starting position of the sweep area, and returns QHYCCD_SUCCESS on success.The starting position is the coordinate (0,0) of the upper left corner of the image as the reference point. If the size of the overscan area obtained is 0, it means that the camera does not have an overscan area.In addition, the location of the overscan area is not limited to a certain side of the effective area, but may also be located in other locations of the effective area. The overscan area is different for different cameras, but the overscan area is the same for cameras of the same model.

Sample Code:

```
int startx,starty,sizex,sizey;
int ret = QHYCCD_ERROR;
ret = GetQHYCCDOverScanArea(camhandle, &startx, &starty, &sizex, &sizey);
if(ret == QHYCCD_SUCCESS)
{
    printf("Get camera overscan area success.\n");
}else{
    printf("Get camera overscan area failed.\n");
}
```

The following illustration shows the valid and overswept areas:



18.uint32_t SetQHYCCDBinMode(qhyccd_handle *handle, uint32_t wbin, uint32_t hbin);

Parameter Introduction:

handle: Handle to camera device;

wbin: horizontal BIN;

hbin: vertical BIN;

Function Introduction:

To set the camera's BIN mode, such as 1X1, 2X2, etc., IsQHYCCDControlAvailable();Function to get the BIN mode supported by the camera, called with SetQHYCCDResolution();Function used in conjunction with.If the command is executed successfully, QHYCCD_SUCCESS is displayed.

Sample Code:

```
ret = SetQHYCCDBinMode(camhandle,2,2);
if(ret = QHYCCD_SUCCESS)
{
	printf("Set camera bin mode successfully.\n");
}else{
	printf("Set camera bin mode failed.\n");
}
```

19.uint32_t SetQHYCCDResolution(qhyccd_handle *handle, uint32_t x, uint32_t y, uint32_t xsize, uint32_t ysize);

Parameter Introduction:

handle: Device handle of the camera;

x: X coordinates of the starting position;

y: Y coordinates of the starting position;

xsize: Image width set;

ysize: Image height set;

Function Introduction:

Used to set the resolution and ROI of the camera image, the base reference point is the upper left corner of the image.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
ret = SetQHYCCDResolution(camhandle, 0, 0, 500, 500);//The starting position is in the upper left corner and the ROI is 500X500
if(ret == QHYCCD_SUCCESS)
{
	printf("Set camera resolution success.\n");}
else{
	printf("Set camera resolution failed.\n");
}
```

20.uint32_t GetQHYCCDCurrentROI(qhyccd_handle *handle, uint32_t *startX, uint32_t *startY, uint32_t *sizeX, uint32_t *sizeY);

Parameter Introduction:

handle: Device handle of the camera;

startX: X coordinates of the starting position;

startY: Y coordinates of the starting position;

sizeX: Image width set;

sizeY: Image height set;

Function Introduction:

The setting used to get the ROI of the camera image. The base reference point is the upper left corner of the image.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t startX, startY, sizeX, sizeY;
ret = GetQHYCCDCurrentROI(camhandle, &startX, &startY, &sizeX, &sizeY);
if(ret == QHYCCD_SUCCESS)
{
	printf("Get camera resolution success.\n");}
else{
	printf("Get camera resolution fail.\n");
}
```

21.uint32_t SetQHYCCDDebayerOnOff(qhyccd_handle *handle, bool onoff);

Parameter Introduction:

handle: Device handle of the camera;

onoff: Color mode on or off, true: on, false: off;

Function Introduction:

The color mode used to set the color camera on and off, only for the color camera, before calling this function need to call IsQHYCCDControlAvailable function to determine whether the camera is a color camera.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
ret = SetQHYCCDDebayerOnOff(camhandle,true);
if(ret == QHYCCD_SUCCESS)
{
	printf("Set camera debayer on success.\n");}
else{
	printf("Set camera debayer on fail.\n");
```

```
}
```

22.uint32_t IsQHYCCDControlAvailable(qhyccd_handle *handle, CONTROL_ID controlId);

Parameter Introduction:

handle: Device handle of the camera;controlId: enumeration variable representing camera functionality;

Function Introduction:

Check whether the camera has a certain function based on the CONTROL_ID.Return QHYCCD_SUCCESS if the camera has a function, or QHYCCD_ERROR otherwise.For the specific meanings of each CONTROL_ID, see the description in "Functions 3. Check the Functions supported by the Camera".

Sample Code:

```
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
    printf("This camera can setup gain.\n");
}else{
    printf("This camera can setup gain.\n");
}
```

23.uint32_t GetQHYCCDParamMinMaxStep(qhyccd_handle *handle, CONTROL_ID controlId, double *min, double *max, double *step);

Parameter Introduction:

handle: Device handle of the camera;

controlId: enumeration variable representing camera functionality;

min: used to store the minimum value of the parameter setting range;

max: used to store the maximum value of the parameter setting range;

step: Minimum step size for storing parameter Settings;

Function Introduction:

You can use CONTROL_ID to obtain the value range of camera parameters. You can use this function to obtain the maximum and minimum values of the range and the minimum setting step. If the command is executed successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
double min,max,step;
ret = GetQHYCCDParamMinMaxStep(camhandle, CONTROL_GAIN, &min, &max, &step);
if(ret == QHYCCD_SUCCESS)
{
```

```
    printf("min = %lf max = %lf step = %lf\n",min,max,step);
}else{
    printf("Get param min max step fail\n");
}
```

24.double GetQHYCCDParam(qhyccd_handle *handle, CONTROL_ID controlId);

Parameter Introduction:

handle: Device handle of the camera;

controlId: enumeration variable representing camera functionality;

Function Introduction:

According to the CONTROL_ID, the system obtains the Settings of camera parameters, such as exposure time, gain, and bias.Returns camera parameters on success, or QHYCCD_ERROR on failure.Note that some of these functions are not supported by all cameras. Before using them, you need to use the IsQHYCCDControlAvailable function to check whether the camera supports this function.

Sample Code:

```
ret = GetQHYCCDParam(camhandle,CONTROL_EXPOSURE);
if(ret != QHYCCD_ERROR)
{
    printf("The camera's expose time is %d ms.\n",ret/1000);}
else{
    printf("Get the camera's expose time fail.\n");
}
```

The following is all CONTROL_ID supported by GetQHYCCDParam:

Index	CONTROL_ID	Introduction
0	CONTROL_WBR	Get red balance value
1	CONTROL_WBG	Get green balance value
2	CONTROL_WBB	Get Blue balance value
3	CONTROL_EXPOSURE	Get expose time value
4	CONTROL_GAIN	Get gain value
5	CONTROL_OFFSET	Get offset value
6	CONTROL_SPEED	Get speed value
7	CONTROL_USBTRAFFIC	Get traffic value
8	CONTROL_VACUUM_PUMP	Get vacuum pump setting value
9	CONTROL_SensorChamberCycle_PUMP	Get cycle pump setting value
10	CONTROL_TRANSFERBIT	Get image bits
11	CONTROL_CURTEMP	Get current temperature
12	CONTROL_CURPWM	Get current cooler PWM
13	CONTROL_COOLER	Get cooler setting value
14	CONTROL_BRIGHTNESS	Get brightness value
15	CONTROL_CONTRAST	Get contrast value
16	CONTROL_GAMMA	Get gamma value
17	CONTROL_AMPV	Get amp glow setting value
18	CONTROL_VCAM	Get BroadCast WDM setting value
19	CAM_CHIPTEMPERATURESENSOR_INTERFACE	Get chip temperature sensor setting value
20	CAM_VIEW_MODE	Get view mode setting value(Not enabled)
21	CAM_GPS	Get GPS setting value
22	CONTROL_CFWSLOTSNUM	Get CFW slots number

23	CONTROL_CFWPORT	Get CFW current position
24	CONTROL_DDR	Get DDR setting value
25	CAM_LIGHT_PERFORMANCE_MODE	Get high/low gain setting value
26	CAM_QHY5II_GUIDE_MODE	Get QHY5II camera guider mode setting value
27	DDR_BUFFER_CAPACITY	Get DDR current data
28	DDR_BUFFER_READ_THRESHOLD	Get DDR read out threshold
29	OutputDataActualBits	Get raw image data bits
30	OutputDataAlignment	Get output data alignment(Not enabled)
31	CAM_HUMIDITY	Get humidity
32	CAM_PRESSURE	Get pressure
33	CAM_Sensor_ULVO_Status	Get ULVO status

25.uint32_t SetQHYCCDParam(qhyccd_handle *handle, CONTROL_ID controlId, double value);

Parameter Introduction:

handle: Device handle of the camera;

controlId: Enumeration variable representing camera functionality;

value: Parameter setting value;

Function Introduction:

Set the camera function parameters according to the CONTROL_ID. If the function is executed successfully, QHYCCD_SUCCESS is returned.The IsQHYCCDControlAvailable function can be used to check whether the camera supports this function. The GetQHYCCDParamMinMaxStep function can be used to obtain the parameter setting range and step size.

Sample Code:

```
ret = SetQHYCCDParam(camhandle,CONTROL_EXPOSURE,20*1000);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set camera's expose time success.\n");
}else{
    printf("Set camera's expose time failed.\n");
}
```

CONTROL_ID for all supported SetQHYCCDParam:

Index	CONTROL_ID	Introduction
1	CONTROL_WBR	Set red balance
2	CONTROL_WBG	Set green balance
3	CONTROL_WBB	Set blue balance
4	CONTROL_EXPOSURE	Set expose time
5	CONTROL_GAIN	Set gain
6	CONTROL_OFFSET	Set offset
7	CONTROL_SPEED	Set speed
8	CONTROL_USBTRAFFIC	Set traffic
9	CONTROL_VACUUM_PUMP	Set vacuum pump on/off
10	CONTROL_SensorChamberCycle_PUMP	Set cycle pump on/off
11	CONTROL_TRANSFERBIT	Set image bits

12	CONTROL_ROWNOISERE	Set row denoise to decrease horizontal lines
13	CONTROL_MANULPWM	Set cooler PWM
14	CAM_GPS	Set GPS
15	CAM_IGNOREOVERSCAN_INTERFACE	Set overscan calibration
16	QHYCCD_3A_AUTOBALANCE	Set auto balance
17	QHYCCD_3A_AUTOEXPOSURE	Set auto expose
18	QHYCCD_3A_AUTOFOCUS	Set auto focus
19	CONTROL_BRIGHTNESS	Set brightness
20	CONTROL_CONTRAST	Set contrast
21	CONTROL_GAMMA	Set gamma
22	CONTROL_AMPV	Set amp glow on/off
23	CONTROL_COOLER	Set cooler target temperature
24	CONTROL_VCAM	Set broadCast WDM on/off
25	CAM_VIEW_MODE	Set view mode(Not enabled)
26	CONTROL_CFWPORT	Set CFW target position
27	CONTROL_DDR	Set DDR on/off
28	CAM_LIGHT_PERFORMANCE_MODE	Set switch high/low gain
29	CAM_QHY5II_GUIDE_MODE	Set QHY5II guider mode on/off
30	CONTROL_ImgProc	Set image rotation or mirror

26.uint32_t GetQHYCCDMemLength(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Returns the maximum memory size required for the camera image. This function returns a fixed memory size that is slightly larger than the actual size required to avoid unexpected memory overflow. It is calculated as:

MONO camera: (imagew+100)*(imageh+100)*2

Color camera: (imagew+100)*(imageh+100)*3

Sample Code:

```
uint32_t length = 0;
length = GetQHYCCDMemLength(camhandle);
if(length > 0)
{
    printf("Get memory length successfully.\n");
}else{
    printf("Get memory length failed.\n");
}
```

27.uint32_t ExpQHYCCDSingleFrame(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Start single frame mode exposure, call the camera to start exposing a single frame of the image.If the function is successfully executed, QHYCCD_SUCCESS is returned. Some cameras may return other values. Normally, if the returned value is not QHYCCD_ERROR, the execution is considered successful.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = ExpQHYCCDSingleFrame(camhandle);
if(ret = QHYCCD_SUCCESS)
{
    printf("Camera expose success.\n" );
}else{
    printf("Camera expose failed.\n" );
}
```

28.uint32_t GetQHYCCDSingleFrame(qhyccd_handle *handle, uint32_t *w, uint32_t *h, uint32_t *bpp, uint32_t *channels, uint8_t *imgdata);**Parameter**

Introduction:

handle: Device handle of the camera;

w: Used to store the obtained image width information;

h: Used to store the obtained image height information;

bpp: Used to store the obtained image bits information;

channels: Used to store the obtained image channels information;

imgdata: Used to store the obtained image data;

Function Introduction:

Get a frame of image data from the camera and store the obtained data in ImgData.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = GetQHYCCDSingleFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
if(ret == QHYCCD_SUCCESS)
{
    printf("Get camera single frame succeess.\n" );}
else{
    printf("Get camera single frame failed.\n" );
}
```

29.uint32_t CancelQHYCCDExposingAndReadout(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Stop camera exposure and stop data reading.To ensure the synchronization of the software and the camera, the camera does not output data and software do not receive data.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = CancelQHYCCDExposingAndReadout(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Cancel camera expose and readout success.\n" );
}else{
    printf("Cancel camera expose and readout failed.\n" );
}
```

30.uint32_t CancelQHYCCDExposing(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Stop the camera exposure.For CYUSB cameras, the two stop exposure functions are the same, stopping exposure also stops data reading, but for WINUSB cameras this function only stops exposure time, image data still needs to be read.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = CancelQHYCCDExposing(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Cancel camera expose success!\n" );
}else{
    printf("Cancel camera expose failed.\n" );
}
```

31.uint32_t BeginQHYCCDLive(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

The continuous mode exposure starts. After the exposure starts, the video stream data will be continuously output. The upper computer needs to continuously call GetQHYCCDLiveFrame function to read the image data.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = BeginQHYCCDLive(camhandle);
if(ret = QHYCCD_SUCCESS)
{
    printf("Camera begin live success.\n" );
}else{
    printf("Camera begin live failed.\n" );
}
```

32.uint32_t GetQHYCCDLiveFrame(qhyccd_handle *handle, uint32_t *w, uint32_t *h, uint32_t *bpp, uint32_t *channels, uint8_t *imgdata);

Parameter Introduction:

handle: Device handle of the camera;

w: Used to store the obtained image width information;

h: Used to store the obtained image height information;

bpp: Used to store the obtained image bits information;

channels: Used to store the obtained image channels information;

imgdata: Used to store the obtained image data;

Function Introduction:

Image data is obtained from the camera and stored in ImgData.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int ret = QHYCCD_ERROR;
ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
if( == QHYCCD_SUCCESS)
{
    printf(" Get camera live frame succeess.\n" );
}else{
    printf(" Get camera live frame failed.\n" );
}
```

33.uint32_t StopQHYCCDLive(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Stop the camera's continuous mode.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
ret = StopQHYCCDLive(camhandle);
if(ret == QHYCCD_SUCCESS)
{
	printf("Stop camera live success.\n");
}else{
	printf("Stop camera live failed.\n");
}
```

34.uint32_t ControlQHYCCDTemp(qhyccd_handle *handle, double targettemp);

Parameter Introduction:

handle: Device handle of the camera;targettemp: Camera target temperature;

Function Introduction:

Set the target temperature for camera cooling, which is the same as SetQHYCCDParam's CONRTOL_COOLER function. QHYCCD_SUCCESS is returned upon successful execution.IsQHYCCDControlAvailable();Function to check whether the camera has cooling function.

Sample Code:

```
double temp = 0;
ret = ControlQHYCCDTemp(camhandle,temp);
if(ret == QHYCCD_SUCCESS)
{
	printf("Control camera temperature success.\n");
}else{
	printf("Control camera temperature failed.\n");
}
```

35.uint32_T IsQHYCCDCFWPlugged(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Check whether the filter wheel is connected. Only QHY5IIICOOL and MINICAM5F_M implement this function. If QHYCCD_SUCCESS is returned, it is considered connected.

Sample Code:

```
ret = IsQHYCCDCFWPlugged(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf(" CFW has been connected.\n" );
}else{
    printf(" CFW didn' t be connected.\n" );
}
```

36.uint32_t SendOrder2QHYCCDCFW(qhyccd_handle *handle, char *order, uint32_t length);

Parameter Introduction:

handle: Device handle of the camera;

order: Target position of the filter wheel;length: The character length of: order, usually 1;

Function Introduction:

Control the rotation of the filter wheel to the specified position, return QHYCCD_SUCCESS on success;

Sample Code:

```
char order = ' 0';
ret = SendOrder2QHYCCDCFW(camhandle,&order,1);
if(ret = QHYCCD_SUCCESS)
{
    printf(" Set CFW success.\n" );
}else{
    printf(" Set CFW error.\n" );
}
```

37.uint32_t GetQHYCCDCFWStatus(qhyccd_handle *handle, char *status);

Parameter Introduction:

handle: Device handle of the camera;

status: Filter wheel status information;

Function Introduction:

Get the current position information of the filter wheel, that is, the current hole, if the function is successfully executed, return QHYCCD_SUCCESS.This function has the same CONTROL_CFWPORT function as GetQHYCCDParam, but returns a different value. This function returns a char while GetQHYCCDParam returns a double.The GetQHYCCDParam function is recommended for secondary development in C# because this function does not get the data correctly.

Sample Code:

```
char status;
ret = GetQHYCCDCFWStatus(camhandle, &status);
```

```
if(ret = QHYCCD_SUCCESS)
{
    printf(“Now position is %c.\n”, status );
}else{
    printf(“ Get QHYCCD CFW status error.\n” );
}
```

38.uint32_t SetQHYCCDGPSVCOXFreq(qhyccd_handle *handle, uint16_t i);

Parameter Introduction:

handle: Device handle of the camera;i: frequency of VCOX;

Function Introduction:

VCOX is used to control the FREQUENCY of the GPS camera. Currently only qHY174-GPS supports this function.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int i = 100;
ret = SetQHYCCDGPSVCOXFreq(camhandle,i);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set QHYCCD VCOX frequency success.\n");
}else{
    printf("Set QHYCCD VCOX frequency fail.\n");
}
```

39.uint32_t SetQHYCCDGPSLedCalMode(qhyccd_handle *handle, uint8_t i);

Parameter Introduction:

handle: Device handle of the camera;

i: LED light switch and mode selection, 0 for off, 1 for slave mode, 2 for main mode;

Function Introduction:

Set the switch and calibration mode of LED lamp calibration. When the parameter is 0, the LED is turned off; when the parameter is 1, the LED is calibrated in secondary mode; and when the parameter is 2, the main mode is calibrated. Generally, this function needs to be called only when the LED is turned off.Currently only QHY174-GPS supports this function.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int i = 2;
ret = SetQHYCCDGPSLedCalMode(camhandle,i);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set QHYCCD led cal mode success.\n");
```

```
}else{
	printf("Set QHYCCD led cal mode fail.\n");
}
```

40.void SetQHYCCDGPSPOSA(qhyccd_handle *handle, uint8_t is_slave, uint32_t pos, uint8_t width);

Parameter Introduction:

handle: Device handle of the camera;

is_slave: Camera working mode, 0 for main mode, 1 for slave mode;

pos: pulse position;width: pulse width, usually fixed at 54;

Function Introduction:

Set the LED pulse position for shutter exposure.This position needs to be set when changing exposure time.The measurement circuit will use this position as the shutter start time. For now only qHY174-GPS supports this function.

Sample Code:

```
int pos = 1000,width = 54;
SetQHYCCDGPSPOSA(camhandle,pos,width);
```

41.void SetQHYCCDGPSPOSB(qhyccd_handle *handle, uint8_t is_slave, uint32_t pos, uint8_t width);

Parameter Introduction:

handle: Device handle of the camera;

is_slave: depends on what mode the camera is in. 0: master mode, 1: slave mode.

pos: Set the LED pulse position;

width: LED pulse width, usually set to 54;

Function Introduction:

Set the LED pulse position for shutter exposure.This position needs to be set when changing exposure time.The measurement circuit will use this position as the shutter end time. For now only qHY174-GPS supports this function.

Sample Code:

```
int pos = 10000,width = 54;
SetQHYCCDGPSPOSA(camhandle,pos,width);
```

42.uint32_t SetQHYCCDGPSMasterSlave(qhyccd_handle *handle, uint8_t i);

Parameter Introduction:

handle: Device handle of the camera;

i: Set the camera's master/slave mode. 0 is master mode and 1 is slave mode.

Function Introduction:

This function is used to control the master/slave mode of the GPS camera. Currently only qHY174-GPS supports this function.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
int i = 0;
ret = SetQHYCCDGPSMasterSlave(camhandle,i);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set QHYCCD GPS master slave success.\n");
}else{
    printf("Set QHYCCD GPS master slave failed.\n");
}
```

43.void SetQHYCCDGPSSlaveModeParameter(qhyccd_handle *handle, uint32_t target_sec, uint32_t target_us, uint32_t deltaT_sec, uint32_t deltaT_us, uint32_t expTime);

Parameter Introduction:

handle: Device handle of the camera;

target_sec: Start time, in s, "JS" as defined by QHYCCD, which refers to a period of time;

target_us: Start time, in us.

deltaT_sec: interval, unit: s;

deltaT_us: interval, unit: us;

expTime: Exposure time, unit: us;

Function Introduction:

Set the parameters for shooting in slave mode. Currently, only QHY174-GPS supports this function.If the execution succeeds, QHYCCD_SUCCESS is returned.

Sample Code:

Set the time to 0:30 on October 19, 2017 and we can get JS of 695925030. Now we want the camera to start exposure in 10 minutes (600s) with exposure time of 100ms and exposure interval of 200ms.

```
target_sec=695925030+600;
target_us=0;
deltaT_sec=0deltaT_us=200*1000;
expTime=100*1000;
SetQHYCCDGPSSlaveModeParameter(camhandle, target_sec, target_us, deltaT_sec, deltaT_us, expTime) ;
```

44.uint32_t SetQHYCCDEnableLiveModeAntiRBI(qhyccd_handle *h, uint32_t value);

Parameter Introduction:

h: Device handle of the camera;

value: Enable or disable AntiRBI mode, 0x1C00;

Function Introduction:

It is used to turn on the AntiRBI mode of the camera. After turning on, the image will output the image in the form of one light and one dark. This mode can eliminate the influence of image residual shadow.

Sample Code:

```
ret = SetQHYCCDEnableLiveModeAntiABI(camhandle, 0x1C00);
if (ret == QHYCCD_SUCCESS)
{
    printf("Enable Live Mode AntiRBI successfully.\n");
}else{
    printf("Enable Live Mode AntiRBI failed.\n");
}
```

45.uint32_t EnableQHYCCDBurstMode(qhyccd_handle *h,bool i);

Parameter Introduction:

h: Device handle of the camera;

i: Enable or disable Burst mode, true: enable Burst mode, false: disable Burst mode.

Function Introduction:

Set Burst mode on or off, return QHYCCD_SUCCESS on success.Burst mode is a sub-mode of continuous mode, which can only be set in continuous mode. When Burst mode is enabled, the camera pauses the output video stream and waits for the start command. After receiving the start command, the camera outputs pre-set images according to the Settings.

Sample Code:

```
ret = EnableQHYCCDBurstMode(camhandle, true);
if(ret == QHYCCD_SUCCESS)
{
    printf("Enable Burst Mode Successfully.\n");
}else{
    printf("Enable Burst Mode Failed.\n");
}
```

46.uint32_t EnableQHYCCDBurstCountFun(qhyccd_handle *h,bool i);

Parameter Introduction:

h: Device handle of the camera;

i: Enable or disable the Burst mode counting function, true means enable, false means disable;

Function Introduction:

Turn on or off counting in Burst mode, return QHYCCD_SUCCESS on success.

Sample Code:

```
ret = EnableQHYCCDBurstCountFun(camhandle, true);
if(ret == QHYCCD_SUCCESS)
{
    printf("Enable Burst Mode Count Function successfully.\n");
}else{
    printf("Enable Burst Mode Count Function failed.\n");
}
```

47.uint32_t ResetQHYCCDFrameCounter(qhyccd_handle *h);

Parameter Introduction:

h: Device handle of the camera;

Function Introduction:

Reset the counter to 0. The ReleaseQHYCCDBurstIDLE function is zeroed out every time it is called in Burst mode, so this function does not need to be executed in Burst mode and returns QHYCCD_SUCCESS on success.

Sample Code:

```
ret = ResetQHYCCDFrameCounter(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Reset Frame Counter Successfully.\n");
}else{
    printf("Reset Frame Counter Failed.\n");
}
```

48.uint32_t SetQHYCCDBurstModeStartEnd(qhyccd_handle *h, unsigned short start, unsigned short end);

Parameter Introduction:

h: Device handle of the camera;

start: the beginning frame of the image output;

end: End frame of image output;

Function Introduction:

The output of the specified image is based on the Settings, such as set start to 1, end to 4, the output of frames 2 and 3, QHYCCD_SUCCESS on success.

Sample Code:

```
ret = SetQHYCCDBurstModeStartEnd(camhandle, 1, 4);
```

```
if(ret == QHYCCD_SUCCESS)
{
    printf("Set Burst Mode Start and End Successfully.\n");
}else{
    printf("Set Burst Mode Start and End Failed.\n");
}
```

49.uint32_t SetQHYCCDBurstIDLE(qhyccd_handle *h);

Parameter Introduction:

h: Device handle of the camera;

Function Introduction:

Set the camera to IDLE. If the command is executed successfully, QHYCCD_SUCCESS is returned.This function needs to be used with ReleaseQHYCCDBurstIDLE.

Sample Code:

```
ret = SetQHYCCDBurstIDLE(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set Burst Mode IDLE Successfully.\n");
}else{
    printf("Set Burst Mode IDLE Failed.\n");
}
```

50.uint32_t ReleaseQHYCCDBurstIDLE(qhyccd_handle *h);

Parameter Introduction:

h: Device handle of the camera;

Function Introduction:

It needs to be used together with SetQHYCCDBurstIDLE function to release the IDLE state in Burst mode of the camera. After removing the IDLE state, the camera will output the pre-set image.

Sample Code:

```
ret = ReleaseQHYCCDBurstIDLE(camhandle);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set Burst Mode IDLE Successfully.\n");
}else{
    printf("Set Burst Mode IDLE Failed.\n");
}
```

51.uint32_t SetQHYCCDBurstModePatchNumber(qhyccd_handle *h,uint32_t value);

Parameter Introduction:

h: Device handle of the camera;

value: Packet size;

Function Introduction:

Used to supplement packets in Burst mode to avoid insufficient data to output images.

Sample Code:

```
ret = SetQHYCCDBurstModePatchNumber(camhandle, 32001);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set Burst Mode patch number Successfully.\n");
}else{
    printf("Set Burst Mode patch number Failed.\n");
}
```

52.uint32_t GetQHYCCDTrigerInterfaceNumber(qhyccd_handle *handle, uint32_t *modeNumber)

Parameter Introduction:

handle: Device handle of the camera;

modeNumber: Number of interfaces;

Function Introduction:

Get the number of camera trigger interfaces. Some cameras support multiple trigger interfaces.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
uint32_t number;
retVal = GetQHYCCDTrigerInterfaceNumber(camhandle, &number);
if(ret == QHYCCD_SUCCESS)
{
    printf("Get interface number successfully.\n");
}
```

53.uint32_t GetQHYCCDTrigerInterfaceName(qhyccd_handle *handle, uint32_t modeNumber, char *name)

Parameter Introduction:

handle: Device handle of the camera;

modeNumber: indicates the subscript of the interface name

name: Indicates the interface name

Function Introduction:

Gets the name of the trigger interface supported by the camera.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
char name[40] = { 0 };
for(int i = 0; i < number; i++)
{
    retVal = GetQHYCCDTrigerInterfaceName(camhandle, i, name);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get triger interface name successfully.\n");
    }
}
```

54.uint32_t SetQHYCCDTrigerInterface(qhyccd_handle *handle, uint32_t trigerMode)

Parameter Introduction:

handle: Device handle of the camera;

trigerMode: Trigger interface of the camera

Function Introduction:

Set the camera trigger interface. You can use this function when the camera supports multiple trigger interfaces.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = SetQHYCCDTrigerInterface(camhandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Set triger mode successfully.\n");
}
```

55.uint32_t SetQHYCCDTrigerMode(qhyccd_handle *handle, uint32_t trigerMode);

Parameter Introduction:

handle: Device handle of the camera;

trigerMode: Camera trigger mode

Function Introduction:

Sets the trigger function mode of the camera. If the parameter is 0, the external trigger function is disabled,

the same as SetQHYCCDTrigerFunction(CamHandle, false).When the camera supports multiple trigger modes, you can use this function to set the working mode of the trigger function.The current function is not perfect, can be temporarily ignored.

Sample Code:

```
ret = SetQHYCCDTrigerMode(camhandle, 0);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set Trigger Mode Successfully.\n");
}
```

56.uint32_t SetQHYCCDTrigerFunction(qhyccd_handle, bool value);

Parameter Introduction:

handle: Device handle of the camera;

value: Enable or disable the external triggering function, true: enable, false: disable;

Function Introduction:

Set on or off the external Trigger function of the camera. When the external Trigger function is on, the camera will enter the Trigger In mode. In this mode, the camera will not immediately start exposure, but wait for the external Trigger signal.If the function executes successfully, QHYCCD_SUCCESS is returned.

Sample Code:

```
ret = SetQHYCCDTrigerFunction(camhandle,true);
if(ret == QHYCCD_SUCCESS)
{
    printf(" Open QHYCCD triger success.\n" );
}else{
    printf(" Open QHYCCD triger failed.\n" );
}
```

57.uint32_t EnableQHYCCDTrigerOut(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Enable the Trigger Out function of the camera. When this function is enabled, the camera outputs the measurement aveform related to exposure.Note that the QHY4040 and QHY4040PRO cannot use Trigger In and Trigger Out functions simultaneously due to hardware reasons.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = EnableQHYCCDTrigerOut(camhandle);
if(retVal == QHYCCD_SUCCESS)
```

```
{
    printf("Enable trigger out successfully.\n");
}
```

58.uint32_t EnableQHYCCDTrigerOutA(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

Enable the Trigger Out function of the camera and output the Trigger signal in mode A. When enabled, the camera outputs the measurement waveform related to exposure.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = EnableQHYCCDTrigerOutA(camhandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Enable trigger out successfully.\n");
}
```

59.uint32_t SendSoftTriger2QHYCCDCam(qhyccd_handle *handle)

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

When the camera is In Trigger In mode, software Trigger signals can be sent to the camera through this function to Trigger camera shooting.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = SendQHYCCDTriger2QHYCCDCam(camhandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Send software trigger signal successfully.\n");
}
```

60.uint32_t SetQHYCCDTrigerFilterOnOff(qhyccd_handle *handle, bool onoff)

Parameter Introduction:

handle: Device handle of the camera;

onoff: Enables or disables filtering

Function Introduction:

Enable or disable the filtering function, through which the influence of power fluctuation caused by mechanical switch can be shielded. By default, the filtering function is enabled and the filtering time is 100ms.

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = SetQHYCCDTrigerFilterOnOff(camhandle, false);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Turn trigger filter off successfully.\n");
}
```

61.uint32_t SetQHYCCDTrigerFilterTime(qhyccd_handle *handle, uint32_t time);

Parameter Introduction:

handle: Device handle of the camera;

time: Filter time, expressed in ms. The value ranges from 1 to 100000ms

Function Introduction:

Set the filtering time for trigger mode. Default is 100ms

Sample Code:

```
uint32_t retVal = QHYCCD_ERROR;
retVal = SetQHYCCDTrigerFilterTime(camhandle, 150);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Set trigger filter time successfully.\n");
}
```

62. uint32_t EnableQHYCCDImageOSD(qhyccd_handle *h,uint32_t i);

Parameter Introduction:

h: Device handle of the camera;

i: Whether to display text, 0 indicates no display, 1 indicates frame serial number, 2 indicates GPS information display;

Function Introduction:

You can display text in the upper left corner of the image, such as frame number, GPS information, etc., and return QHYCCD_SUCCESS on success.

Sample Code:

```
ret = EnableQHYCCDImageOSD(camhandle, 1);
if(ret == QHYCCD_SUCCESS)
{
    printf("Enable display image sequence number successfully.\n");
}else{
    printf("Enable display image sequence number failed.\n");
```

```
}
```

63.void EnableQHYCCDMessage(bool enable);

Parameter Introduction:

enable: Enables or disables debug output.

Function Introduction:

Enable or disable the debugging information output of the SDK.Setting information output can be captured by the compiler or the DebugView. The DebugView captures the keyword QHYCCD.

Sample Code:

```
EnableQHYCCDMessage(true);
```

64.void RegisterPnpEventIn(void (*in_pnp_event_in_func)(char *id))

Parameter Introduction:

in_pnp_event_in_func: Registers the address pointer of the function

Function Introduction:

Register the camera access event. When the camera is connected, the SDK will call the function registered in the upper computer.

Sample Code:

```
void pnp_Event_In_Func(char *id)
{
    printf("Camera In.\n");
}

RegisterPnpEventIn(pnp_Event_In_Func);
```

65.void RegisterPnpEventOut(void (*in_pnp_event_out_func)(char *id))

Parameter Introduction:

in_pnp_event_out_func: Registers the address pointer of the function

Function Introduction:

Register the camera disconnect time. When the camera is disconnected, the SDK will call the function registered in the upper computer.**S**

ample Code:

```
void pnp_Event_Out_Func(char *id)
{
    printf("Camera Out.\n");
}
```

```
RegisterPnpEventOut(pnp_Event_Out_Func);
```

66.uint32_t SetQHYCCDTwoChannelCombineParameter(qhyccd_handle *handle, double x,double ah,double bh,double al,double bl);

Parameter Introduction:

handle: Device handle of the camera;

x: Switching point of high and low gain

ah: High gain channel slope coefficient

bh: High gain channel intercept coefficient

al: Slope coefficient of low gain channel

bl: low gain channel ending coefficient

Function Introduction:

This function is applicable to cameras with high-low gain channels. When 16-bit data is output by high-low gain channel splicing, the effect may be affected by temperature and the smooth transition between high-low gain channels cannot be achieved. In this case, you can use this function to adjust to ensure the accuracy and linearity of splicing.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t ret = QHYCCD_ERROR;
ret = SetQHYCCDTwoChannelCombineParameter(camhandle, 4096, 0, 30000, 16, 30000);
if(ret == QHYCCD_SUCCESS)
{
    printf("Setup combine parameter successfully.\n");
}else{
    printf("Setup combine parameter failed.\n");
}
```

67.uint32_t GetQHYCCDPreciseExposureInfo(qhyccd_handle *h, uint32_t *PixelPeriod_ps, uint32_t *LinePeriod_ns, uint32_t *FramePeriod_us, uint32_t *ClocksPerLine, uint32_t *LinesPerFrame, uint32_t *ActualExposureTime, uint8_t *isLongExposureMode);

Parameter Introduction:

h: Device handle of the camera;

PixelPeriod_ps: Length of period time in pixels, in picoseconds

LinePeriod_ns: Indicates the line cycle time, in nanoseconds

FramePeriod_us: The length of the frame interval, in microseconds, which varies in single-frame continuous mode and in different readout modes

ClocksPerLine: Number of clocks per line

LinesPerFrame: The number of lines per frame

ActualExposureTime: Actual exposure time, in microseconds

isLongExposureMode: Long exposure flag, 0 for short exposure, 1 for long exposure, judging by whether the frame period is longer than

Function Introduction:

Obtain accurate exposure time information, including line period, frame period, timing exposure time, etc.The function returns QHYCCD_SUCCESS on success.

Sample Code:

```
uint32_t PixelPeriod_ps, LinePeriod_ns, FramePeriod_us, ClocksPerLine, LinesPerFrame, ActualExposureTime;
uint8_t isLongExposureMode;

ret = GetQHYCCDPreciseExposureInfo(camhandle, &PixelPeriod_ps, &LinePeriod_ns, &FramePeriod_us,
&ClocksPerLine, &LinesPerFrame, &ActualExposureTime, &isLongExposureMode);
if(ret == QHYCCD_SUCCESS)
{
    printf("Get precise exposure information successfully.\n");
}else{
    printf("Get precise exposure information failed.\n");
}
```

68.uint32_t GetQHYCCDRollingShutterEndOffset(qhyccd_handle *h,uint32_t row,double *offset);

Parameter Introduction:

h: Device handle of the camera;

row: The number of rows in the image,its range is 0~ImageHeight-1

offset: Indicates the calculation result of the returned offset

Function Introduction:

The offset value of exposure time is calculated according to the number of image lines. The return value of this function can be used together with the GPS time to obtain the exact exposure time of each image line.QHYCCD_SUCCESS is returned on success.

Sample Code:

```
double offset;
ret = GetQHYCCDRollingShutterEndOffset(camhandle, 1000, &offset);
if(ret == QHYCCD_SUCCESS)
{
    printf("Enable display image sequence number successfully.\n");
}else{
```

```
    printf("Enable display image sequence number failed.\n");
}
```

69.void QHYCCDSensorPhaseReTrain(qhyccd_handle *handle);

Parameter Introduction:

handle: Device handle of the camera;

Function Introduction:

To solve the image fringe problem caused by the phase of the camera, call this function will recalculate the phase value inside the camera.

Sample Code:

```
QHYCCDSensorPhaseReTrain(camhandle);
```

Ⅲ.Function Setting Introduction

The standard process of camera control is as follows: 1. Connect the camera →2. Control the camera →3.The connection operation is used to open the camera and perform related initial Settings. The control operation is used to control various functions of the camera, including parameter setting, refrigeration, CFW, GPS, Trigger, Burst and other functions. The disconnect operation is used to shut down the camera and release related resources.

1.Gets the camera handle

Scan the number and ID of connected cameras on the device and open the camera according to the ID to get the device handle.

The functions used are InitQHYCCDResource, ScanQHYCCD, GetQHYCCDId, OpenQHYCCD. The following is the example code for obtaining the camera handle:

```
uint32_t retVal = QHYCCD_ERROR;
char camId[40] = {0};
qhyccd_handle *camHandle = NULL;

//Initialize resources within the SDK
retVal = InitQHYCCDResource();
if(retVal == QHYCCD_SUCCESS)
{
    printf("Initialize SDK resource successfully.\n");
}
else
{
    printf("Initialize SDK resource failed.\n");
    return -1;
}

//Scan the camera and return the number of devices
camNum = ScanQHYCCD();
if(camNum == 0)
{
    printf("Didn't find QHYCCD cameras.\n");
```

```
        return -1;
}

//Connect the first device in the QHY camera list
for(int i = 0; i < camNum; i ++)
{
        retVal = GetQHYCCDId(i, camId); //获取设备 ID
        if(retVal == QHYCCD_SUCCESS)
        {
                printf("Get camera's ID successfully.\n");
                break;
        }
        else
        {
                printf("Get camera's ID failed.\n");
                return -1;
        }
}

camHandle = OpenQHYCCD(camId); //Open the device based on the device ID and return the device handle
if(camHandle != NULL)
{
        printf("Openc camera successfully.\n");
}else{
        printf("Openc camera failed.\n");
        return -1;
}
```

2.Set read mode

Set the readout mode of the camera. Different types of cameras have different readout modes, and the camera has different characteristics in different readout modes.

Use the function of GetQHYCCDNumberOfReadMode, GetQHYCCDReadModeName, GetQHYCCDReadModeResolution, SetQHYCCDReadMode, to set the read mode under the sample code:

```
//Get the number of camera readout modes, return 1 to indicate only one STANDARD MODE
retVal = GetQHYCCDNumberOfReadMode(camHandle,readModeNum);
if(retVal == QHYCCD_SUCCESS)
{
        printf("Get Read Mode number successfully.\n");
}else{
        printf("Get Read Mode number failed.\n");
        return -1;
}

for(int i = 0; i < readModeNum; i++)
{
        //Gets the name of the read mode
        retVal = GetQHYCCDReadModeName(camHandle, i, readModeName);
        if(retVal == QHYCCD_SUCCESS)
        {
                printf("Get Read Mode name successfully.\n");
        }
        else
```

```
    {
        printf("Get Read Mode name failed.\n");
        return -1;
    }

    //Gets the resolution of the current readout mode
    retVal = GetQHYCCDReadModeResolution(camHandle, i, readModeWidth,readModeHeight);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get Read Mode resolution successfully.\n");
    }
    else
    {
        printf("Get Read Mode resolution failed.\n");
        return -1;
    }
}

//This parameter is used to set the read mode. The value ranges from 0 to readModeNum-1. Mode 0 is used as an example
retVal = SetQHYCCDReadMode(camHandle,0);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Get Read Mode name successfully.\n");
}else{
    printf("Get Read Mode name failed.\n");
    return -1;
}
```

3.Set single frame or continuous mode

Sets the camera to work in single frame or continuous mode.

The functions used are IsQHYCCDControlAvailable, SetQHYCCDStreamMode, below is the example code for setting single frame or continuous mode:

```
//Set the single frame mode
retVal = IsQHYCCDControlAvailable(camHandle, CAM_SINGLEFRAMEMODE); //Check whether single frame mode is supported
if(retVal == QHYCCD_SUCCESS)
{
    printf("This camera has single mode.\n");
}else{
    printf("This camera have no single mode.\n");
    return -1;
}

retVal = SetQHYCCDStreamMode(camHandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Set Stream Mode successfully.\n");
}else{
    printf("Set Stream Mode failed.\n");
    return -1;
}
```

```
//Set live mode
retVal = IsQHYCCDControlAvailable(camHandle, CAM_LIVEFRAMEMODE); //Check whether continuous mode is supported
if(retVal == QHYCCD_SUCCESS)
{
	printf("This camera has single mode.\n");
}else{
	printf("This camera have no single mode.\n");
	return -1;
}

retVal = SetQHYCCDStreamMode(camHandle, 1);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set Stream Mode successfully.\n");
}else{
	printf("Set Stream Mode failed.\n");
	return -1;
}
```

4.Initializing the camera

Initialize the camera parameters and adjust the working status of the camera.

The function used is InitQHYCCD and the following is the example code for initializing the camera:

```
retVal = InitQHYCCD(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Initialize camera successfully.\n");
}else{
	printf("Initialize camera successfully.\n");
	return -1;
}
```

5.Connect the camera

This operation is used to start the camera and initialize the camera. After the camera is successfully opened, the device handle of the camera is returned to set the read mode and working mode of the camera through the handle, and initialize the camera parameters.

Use the function of InitQHYCCDResource, ScanQHYCCD, GetQHYCCDId, OpenQHYCCD, GetQHYCCDNumberOfReadMode, GetQHYCCDReadModeName, GetQHYCCDReadModeResolution, SetQHYCCDReadMode, SetQHYCCDStreamMode and InitQHYCCD, the following is the example code to connect the camera:

```
int camNum = 0,readModeNum = 0;
uint32_t retVal = QHYCCD_ERROR;
char camId[40] = {0};
char readModeName[40] = {0};
int readModeWidth = 0,readModeHeight = 0;
qhyccd_handle *camHandle = NULL;

/*****************************************************************************/
/************************** Open the camera to get the handle ***************************/
```

```
/*****************************************************************************/
//Initialize resources within the SDK
retVal = InitQHYCCDResource();
if(retVal == QHYCCD_SUCCESS)
{
	printf("Initialize SDK resource successfully.\n");
}else{
	printf("Initialize SDK resource failed.\n");
	return -1;
}

//Scan the camera and return the number of devices
camNum = ScanQHYCCD();
if(camNum == 0)
{
	printf("Didn't find QHYCCD cameras.\n");
	return -1;
}

//Connect the first device in the QHY camera list
for(int i = 0; i < camNum; i ++)
{
	retVal = GetQHYCCDId(i, camId); //Obtaining the Device ID
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Get camera's ID successfully.\n");
		break;
	}
	else
	{
		printf("Get camera's ID failed.\n");
		return -1;
	}
}

camHandle = OpenQHYCCD(camId); //Open the device based on the device ID and return the device handle
if(camHandle != NULL)
{
	printf("Openc camera successfully.\n");
}else{
	printf("Openc camera failed.\n");
	return -1;
}

/*****************************************************************************/
/******************************** Mode setting *******************************/
/*****************************************************************************/
//Get the number of camera readout modes, return 1 to indicate only one STANDARD MODE
retVal = GetQHYCCDNumberOfReadMode(camHandle,readModeNum);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Get Read Mode number successfully.\n");
}else{
	printf("Get Read Mode number failed.\n");
```

```
        return -1;
}

for(int i = 0; i < readModeNum; i++)
{
        //Gets the name of the read mode
        retVal = GetQHYCCDReadModeName(camHandle, i, readModeName);
        if(retVal == QHYCCD_SUCCESS)
        {
                printf("Get Read Mode name successfully.\n");
        }
        else
        {
                printf("Get Read Mode name failed.\n");
                return -1;
        }

        //Gets the resolution of the current readout mode
        retVal = GetQHYCCDReadModeResolution(camHandle, i, readModeWidth,readModeHeight);
        if(retVal == QHYCCD_SUCCESS)
        {
                printf("Get Read Mode resolution successfully.\n");
        }
        else
        {
                printf("Get Read Mode resolution failed.\n");
                return -1;
        }
}

//This parameter is used to set the read mode. The value ranges from 0 to readModeNum-1. Mode 0 is used as an example
retVal = SetQHYCCDReadMode(camHandle,0);
if(retVal == QHYCCD_SUCCESS)
{
        printf("Get Read Mode name successfully.\n");
}else{
        printf("Get Read Mode name failed.\n");
        return -1;
}

//Set the single frame or continuous mode. If the single frame parameter is 0, the continuous mode is 1. Set the single frame mode here
retVal = IsQHYCCDControlAvailable(camHandle, CAM_SINGLEFRAMEMODE); //Check whether single frame mode is supported
if(retVal == QHYCCD_SUCCESS)
{
        printf("This camera has single mode.\n");
}else{
        printf("This camera have no single mode.\n");
        return -1;
}

retVal = SetQHYCCDStreamMode(camHandle, 0);
```

```
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set Stream Mode successfully.\n");
}else{
	printf("Set Stream Mode failed.\n");
	return -1;
}

/*********************************************************************************/
/******************************** Initializing the camera*************************/
/*********************************************************************************/
retVal = InitQHYCCD(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Initialize camera successfully.\n");
}else{
	printf("Initialize camera successfully.\n");
	return -1;
}
```

6.Disconnect the camera

Close the camera and release related resources. Two points need to be paid attention to when disconnecting the camera. First, before disconnecting the camera, if the camera is shooting, you need to end the shooting task first.Second, after executing CloseQHYCCD, the device handle of the camera will be destroyed. Do not interact with the camera at this time.

CloseQHYCCD and ReleaseQHYCCDResource are used in the following code:

```
//Close the camera
retVal = CloseQHYCCD(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Close camera successfully.\n");
}else{
	printf("Close camera failed.\n");
}

//Releasing SDK Resources
retVal = ReleaseQHYCCDResource();
if(retVal == QHYCCD_SUCCESS)
{
	printf("Release resource successfully.\n");
}else{
	printf("Release resource failed.\n");
}
```

7.Check the features supported by the camera

Check whether the camera supports a certain function based on the CONTROL_ID. You can use the function after obtaining the device handle of the camera.

This function can be used after obtaining the camera handle through the OpenQHYCCD function. The function used is IsQHYCCDControlAvailable. Taking the brightness function as an example, the code for checking whether the camera supports this function is as follows:

```
int retVal = QHYCCD_ERROR;

retVal = IsQHYCCDControlAvailable(camhandle, CONTROL_BRIGHTNESS);
```

```
if(retVal == QHYCCD_SUCCESS)
{
    printf("Camera have this function.\n");
}else{
    printf("Camera don't have this function.\n");
}
```

CONTROL_ID 的定义位于 qhyccdstruct.h 中，下为所有 CONTROL_ID 的定义：

Index	CONTROL_ID	Introduction
0	CONTROL_BRIGHTNESS	Check if support brightness
1	CONTROL_CONTRAST	Check if support contrast
2	CONTROL_WBR	Check if support red balance
3	CONTROL_WBB	Check if support blue balance
4	CONTROL_WBG	Check if support green balance
5	CONTROL_GAMMA	Check if support gamma
6	CONTROL_GAIN	Check if support gain
7	CONTROL_OFFSET	Check if support offset
8	CONTROL_EXPOSURE	Check if support expose time setting
9	CONTROL_SPEED	Check if support speed
10	CONTROL_TRANSFERBIT	Check if support bits setting
11	CONTROL_CHANNELS	Check if support get channels number(Discontinued)
12	CONTROL_USBTRAFFIC	Check if support traffic
13	CONTROL_ROWNOISERE	Check if support row denoise
14	CONTROL_CURTEMP	Check if support get current temperature
15	CONTROL_CURPWM	Check if support get current PWM
16	CONTROL_MANULPWM	Check if support manual cool mode
17	CONTROL_CFWPORT	Check if support CFW
18	CONTROL_COOLER	Check if support auto cool mode
19	CONTROL_ST4PORT	Check if support ST4 port
20	CAM_COLOR	Check if support get bayer matrix
21	CAM_BIN1X1MODE	Check if support 1X1 bin mode
22	CAM_BIN2X2MODE	Check if support 2X2 bin mode
23	CAM_BIN3X3MODE	Check if support 3X3 bin mode
24	CAM_BIN4X4MODE	Check if support 4X4 bin mode
25	CAM_MECHANICALSHUTTER	Check if support machine shutter
26	CAM_TRIGER_INTERFACE	Check if support trigger mode
27	CAM_TECOVERPROTECT_INTERFACE	Check if support temperature over protect,this function will limit cooler max PWM be 70%(Disabled)
28	CAM_SINGNALCLAMP_INTERFACE	Check whether the camera supports the SINGNALCLAMP function, which is a unique feature of CCD cameras for dark bands behind bright stars
29	CAM_FINETONE_INTERFACE	Check whether the camera supports fine tuning, which is used for CCD cameras to optimize the noise characteristics of the camera by fine-tuning the CCD drive and sampling timing
30	CAM_SHUTTERMOTORHEATING_INTERFACE	Check whether the camera supports shutter motor heating
31	CAM_CALIBRATEFPN_INTERFACE	Check whether the camera supports FPN calibration, which reduces FPN noise such as vertical stripes
32	CAM_CHIPTEMPERATURESENSOR_INTERFACE	Check whether the camera supports an on-chip temperature sensor
33	CAM_USBREADOUTSLOWEST_INTERFACE	Check whether the camera supports the USB

		minimum speed readout function (this function duplicates the CONTROL_SPEED function and is no longer in use)
34	CAM_8BITS	Check whether the camera supports 8-bit image data output
35	CAM_16BITS	Check whether the camera supports 16-bit image data output
36	CAM_GPS	Check whether the camera supports GPS
37	CAM_IGNOREOVERSCAN_INTERFACE	Check whether the camera supports the function of overscanning area calibration
38	QHYCCD_3A_AUTOBALANCE	Check whether the camera supports automatic white balance
39	QHYCCD_3A_AUTOEXPOSURE	Check whether the camera supports auto exposure
40	QHYCCD_3A_AUTOFOCUS	Check whether the camera supports autofocus
41	CONTROL_AMPV	Check whether the camera supports glow suppression
42	CONTROL_VCAM	Check whether the camera supports WDM broadcast
43	CAM_VIEW_MODE	Check whether preview mode is supported (not enabled)
44	CONTROL_CFWSLOTSNUM	Check whether the camera can obtain the number of filter wheel holes
45	IS_EXPOSING_DONE	Check whether the camera is exposed (not enabled)
46	ScreenStretchB	Check whether the camera can be stretched Black gray scale
47	ScreenStretchW	Check whether the camera can White grayscale stretching
48	CONTROL_DDR	Check whether the camera supports DDR
49	CAM_LIGHT_PERFORMANCE_MODE	Check whether the camera supports the high-low gain switching function
50	CAM_QHY5II_GUIDE_MODE	Check if the camera is a 5II series camera that supports guide mode
51	DDR_BUFFER_CAPACITY	Check whether the camera can get the current amount of DDR buffer data
52	DDR_BUFFER_READ_THRESHOLD	Check whether the camera can get the buffer read threshold
53	DefaultGain	Check whether the camera can obtain the default gain recommendation
54	DefaultOffset	Check whether the camera can obtain the default bias recommendation
55	OutputDataActualBits	Check whether the camera can get the actual bits of output data
56	OutputDataAlignment	Check whether the camera supports getting output data alignment formats
57	CAM_SINGLEFRAMEMODE	Check whether the camera supports single frame mode
58	CAM_LIVEVIDEOMODE	Check whether the camera supports live frame mode
59	CAM_IS_COLOR	Check if the camera is color
60	hasHardwareFrameCounter	Check whether the camera supports hardware frame counting
61	CONTROL_MAX_ID_Error	Get the maximum value of CONTROL_ID (deprecated)
62	CAM_HUMIDITY	Check whether the camera supports a humidity sensor
63	CAM_PRESSURE	Check whether the camera supports pressure sensors

64	CONTROL_VACUUM_PUMP	Check whether the camera supports vacuum pump
65	CONTROL_SensorChamberCycle_PUMP	Check that the camera supports internal circulation pumps
66	CAM_32BITS	Check whether the camera supports 32-bit image data output
67	CAM_Sensor_ULVO_Status	Check whether the camera supports ULVO status detection
68	CAM_SensorPhaseReTrain	Check whether the camera supports phase adjustment, which handles image streaks due to phase
69	CAM_InitConfigFromFlash	Check whether the camera supports Flash read and write Config
70	CAM_TRIGER_MODE	Check whether the camera supports multiple trigger mode Settings
71	CAM_TRIGER_OUT	Check whether the camera supports trigger output
72	CAM_BURST_MODE	Check whether the camera supports Burst mode
73	CAM_SPEAKER_LED_ALARM	Check whether the camera supports the signal lamp function (currently only for customized models)
74	CAM_WATCH_DOG_FPGA	Check whether camera FPGA supports watchdog processing function (currently only for customized models)
75	CAM_BIN6X6MODE	Check whether the camera supports 6X6 BIN
76	CAM_BIN8X8MODE	Check whether the camera supports 8X8 BIN
77	CAM_GlobalSensorGPSLED	Check whether the camera sensor supports global LED calibration lights
78	CONTROL_ImgProc	Check whether the camera supports image processing
79	CONTROL_MAX_ID	Obtain the maximum value of CONTROL_ID

8.Obtain the SDK version number

Gets the version number currently used by the SDK. Usually, the version number is the date the SDK was released.

This function can be used at any time, using the function GetQHYCCDSDKVersion as follows:

```
int retVal = QHYCCD_ERROR;
uint32_t year,month,day,subday;

retVal = GetQHYCCDSDKVersion(&year,&month,&day,&subday);
if(retVal == QHYCCD_SUCCESS)
{
    printf("The SDK version is %d-%d-%d-%d.\n", year, month, day, subday);
}
```

9.Gets the driver version number

Gets the version of the camera firmware, usually the release date or modification date.

This function can be used after the OpenQHYCCD function is used to obtain the device handle. The function used is GetQHYCCDFWVersion. The example code is as follows:

```
int retVal = QHYCCD_ERROR;
unsigned char fwVer[32];

retVal = GetQHYCCDFWVersion(camHandle,fwVer);
if(retVal == QHYCCD_SUCCESS)
{
    if((fwv[0] >> 4) <= 9)
```

```
    {
        printf("Firmware Version: %d-%d-%d", (fwVer[0]>>4)+0x10, fwVer[0]&~0xf0, fwVer[1]);
    }
    else
    {
        printf("Firmware Version: %d-%d-%d", fwVer[0]>>4, fwVer[0]&~0xf0, fwVer[1]);
    }
}
```

10.Obtain the FPGA version number of the current device

Obtain the FPGA version number of the device. There are two FPGA versions of the device, one is a common version number and the other is a standby version number. If the first version number fails to be read, you can try to read the second version number.

This function can be used after the OpenQHYCCD function is used to obtain the device handle. The function used is GetQHYCCDFPGAVersion. The example code is as follows:

```
int retVal = QHYCCD_ERROR;
uint32_t fpgaVer[32] = {0};

//Gets the first FPGA version number
retVal = GetQHYCCDFPGAVersion(camHandle,0,fpgaVer);
if(retVal == QHYCCD_SUCCESS)
{
    printf("First FPGA Version: %d-%d-%d-%d", fpgaVer[0],fpgaVer[1],fpgaVer[2],fpgaVer[3]);
}

//Get two FPGA version numbers
retVal = GetQHYCCDFPGAVersion(camHandle,1,fpgaVer);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Second FPGA Version: %d-%d-%d-%d", fpgaVer[0],fpgaVer[1],fpgaVer[2],fpgaVer[3]);
}
```

11.Get camera chip information

Get the hardware parameters of the camera, including chip size, pixel size, image size and image bit depth. It should be noted that the bit depth of the image is not fixed. When the bit depth is set, the image bit depth obtained by this function will also change.

This function can be used after initialization with the InitQHYCCD function GetQHYCCDChipInfo, with the following code for example:

```
uint32_t retVal = QHYCCD_SUCCESS;
uint32_t imagew, imageh, bpp;double chipw, chiph, pixelw, pixelh;

retVal = GetQHYCCDChipInfo(camHandle, &chipw, &chiph, &imagew, &imageh, &pixelw, &pixelh, &bpp);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Chip Width: %f mm Chip Height: %f mm", chipw, chiph);
    printf("Pixel Width: %f nm Pixel Height: %f nm", pixelw, pixelh);
    printf("Image Width: %d Image Height: %d Image Bits: %d", imagew,imageh,bpp);
}
```

12.Setting the BIN mode

Generally, cameras have four BIN modes: 1X1 BIN, 2X2 BIN, 3X3 BIN, and 4X4 BIN. Different cameras support different BIN modes. Therefore, you need to check whether the BIN mode is supported by the camera before setting it.

This function can be used after initialization with the InitQHYCCD function. The IsQHYCCDControlAvailable, GetQHYCCDChipInfo, SetQHYCCDBinMode, and SetQHYCCDResolution functions are used. The code for example is as follows:

```
//Set 1 x1 BIN
int retVal = QHYCCD_ERROR;
uint32_t binX = 1, binY = 1;
uint32_t imgW, imgH, imgBpp;double chipW, chipH, pixelW, pixelH;

//Check whether the 1x1 BIN function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CAM_BIN1X1MODE);
if(retVal == QHYCCD_SUCCESS)
{
    //Set 1 x1bin
    retVal = SetQHYCCDBinMode(camHandle, binX, binY);
    if(retVal == QHYCCD_SUCCESS)
    {
        retVal = GetQHYCCDChipInfo(camHandle,&chipW,&chipH,&imgW,&imgH,&pixelW,&pixelH,&imgBpp);
        if(retVal == QHYCCD_SUCCESS)
        {
            //When you set BIN, set the resolution
            retVal = SetQHYCCDResolution(camHandle, 0, 0, imgW / binX, imgH / binY);
            if(retVal == QHYCCD_SUCCESS)
            {
                printf("Set Resolution Successfully.\n");
            }
        }
    }
}
```

13.Sets color mode and gets the Bayer sequence

Set color mode on or off and get the Bayer sequence of the camera. This function is required.Only color cameras support this function. Before setting this function, call IsQHYCCDControlAvailable to check whether the camera is a color camera.

This function can be used after initialization with the InitQHYCCD functions, such as IsQHYCCDControlAvailable and SetQHYCCDDebayerOnOff. The following code is used for example:

```
uint32_t retVal = QHYCCD_SUCCESS;
int bayer;

//Check whether it is a color camera
retVal = IsQHYCCDConrolAvail(camHandle, CAM_IS_COLOR);
if(retVal == QHYCCD_SUCCESS)
{
    //Turn on color mode
    retVal = SetQHYCCDDebayerOnOff(camhandle, true);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Setup Color Mode ON Successfully.\n");
    }

    //Get the camera Bayer sequence
    bayer = IsQHYCCDControlAvailable(camhandle, CAM_COLOR);
    if(retVal != QHYCCD_ERROR)
    {
        printf("Get camera's bayer format successfully.\n");
```

```
    }
}
```

For camera Bayer sequences, these are the four enumeration variables defined in qhyccdstruct.h. Enumeration is defined as follows:enum BAYER_ID{

```
  BAYER_GB = 1,
  BAYER_GR,
  BAYER_BG,
  BAYER_RG};
```

These four enumerated variables represent GBRG, GRBG, BGGR, and RGGB.

14.Set digit and image data format

Set the output image data bits. This function is a necessary function.Note that the output image data of digits and the original data bits may not be consistent, such as camera raw data for the 12, then if set camera output 16-bit data, inside the camera can be in the original data of low zero padding, converted to 16 bits of data, if set the camera output 8 bits of data, high inside the camera will pick up the original data of eight, is transformed into eight bits of data.

This function can be used after initialization with InitQHYCCD function SetQHYCCDParam or SetQHYCCDBitsMode, both functions have the same function, the following is the example code to set the image bit:

```
//Set the 8-bit image
uint32_t retVal = QHYCCD_ERROR;
retVal = IsQHYCCDControlAvailable(camHandle, CAM_8BITS);
if(retVal == QHYCCD_SUCCESS)
{
    retVal = SetQHYCCDParam(camHandle, CONTROL_TRANSFERBITS, 8);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set Image Bits Successfully.\n");
    }
}
//Set the 16-bit image
uint32_t retVal = QHYCCD_ERROR;
retVal = IsQHYCCDControlAvailable(camHandle, CAM_8BITS);
if(retVal == QHYCCD_SUCCESS)
{
    retVal = SetQHYCCDParam(camHandle, CONTROL_TRANSFERBITS, 16);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set Image bits successfully.\n");
    }
}
```

This function is commonly used with SetQHYCCDDebayerOnOff to set the data format of the image. The data structure of the image can be set differently depending on the single frame and continuous mode.

In the single-frame mode, some cameras cannot output 8-bit images, and the astronomical software generally chooses the 16-bit black and white mode when shooting in the single-frame mode. Therefore, in the single-frame mode, the cameras are generally uniformly set to the 16-bit black and white mode, so there is no need to consider the problem of data format switching.If you work in continuous mode, there are usually three image data modes: 8-bit black and white (RAW8), 8-bit color (RGB24) and 16-bit black and white (RAW16). At this time, data format switching needs to be considered. For details, see Readout mode, BIN, and Data Format Switching.

Here is sample code for setting the three data formats:

```
//Set 8-bit black and white mode
retVal = SetQHYCCDParam(camHandle, CONTROL_TRANSFERBIT, 8);
if(retVal == QHYCCD_SUCCESS)
```

```
{
	printf("Set bits mode successfully.\n");
}
retVal = SetQHYCCDDebayerOnOff(camHandle, false);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set color mode successfully.\n");
}

//Set 16-bit black and white mode
retVal = SetQHYCCDParam(camHandle, CONTROL_TRANSFERBIT, 16);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set bits mode successfully.\n");
}
retVal = SetQHYCCDDebayerOnOff(camHandle, false);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set color mode successfully.\n");
}

//Set 8-bit color mode
retVal = SetQHYCCDParam(camHandle, CONTROL_TRANSFERBIT, 8);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set bits mode successfully.\n");
}
retVal = SetQHYCCDDebayerOnOff(camHandle, true);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set color mode successfully.\n");
}
```

15.Switch readout mode, BIN mode, image data format

When you need to change the readout mode, BIN mode or data format of the camera during shooting, you need to end the shooting task of the camera first, and then set the readout mode, BIN mode and data format. During setting, you need to call the InitHYCCD function to initialize the camera, and continue shooting task after setting.In addition, it should be noted that the initialization operation is performed during the switchover, so the parameter values such as Expose Time, Gain, Offset, and Traffic need to be reset after the switchover.

After connecting the camera, you can use the switching function, which can be repeated many times.The functions used include SetQHYCCDReadMode, SetQHYCCDStreamMode, InitQHYCCD, SetQHYCCDBinMode, SetQHYCCDResolution, SetQHYCCDParam, etc. Example codes are as follows:

```
//The camera shooting task is complete. For details, see the single frame or continuous mode shooting function

//Reset the read mode
retVal = SetQHYCCDReadMode(camHandle, readMode);
if(retVal == QHYCCD_ERROR)
{
	printf("Get Read Mode name failed.\n");
	return -1;
}

//Reset the Stream Mode
retVal = SetQHYCCDStreamMode(camHandle, streamMode);
```

```
if(retVal == QHYCCD_ERROR)
{
	printf("Set Stream Mode failed.\n");
	return -1;
}

//Re-initialize the camera
retVal = InitQHYCCD(camHandle);
if(retVal == QHYCCD_ERROR)
{
	printf("Initialize camera successfully.\n");
	return -1;
}

//Reset the BIN mode
retVal = SetQHYCCDBinMode(camhandle, binx, biny);
if(retVal == QHYCCD_ERROR)
{
	printf("Set Bin Mode failed.\n");
	return -1;
}
retVal = SetQHYCCDResolution(camhandle, startx, starty, sizex, sizey);
if(retVal == QHYCCD_ERROR)
{
	printf("Set Resolution failed.\n");
	return -1;
}

//Reformat the data
retVal = SetQHYCCDDebayerOnOff(camhandle, color);
if(retVal == QHYCCD_ERROR)
{
	printf("Set Color Mode failed.\n");
	return -1;
}
retVal = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, bits);
if(retVal == QHYCCD_ERROR)
{
	printf("Set Bits Mode failed.\n");
	return -1;
}

//Reset the camera parameters. For details, see the parameter Settings function
retVal = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, time);
if(retVal == SetQHYCCD_SUCCESS)
{
	printf("Setup expose time successfully.\n");
}
retVal = SetQHYCCDParam(camhandle, CONTROL_GAIN, gain);
if(retVal == SetQHYCCD_SUCCESS)
{
	printf("Setup gain successfully.\n");
}
retVal = SetQHYCCDParam(camhandle, CONTROL_OFFSET, offset);
```

```
if(retVal == SetQHYCCD_SUCCESS)
{
	printf("Setup offset successfully.\n");
}
retVal = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, traffic);
if(retVal == SetQHYCCD_SUCCESS)
{
	printf("Setup traffic successfully.\n");
}
...

//To restart the shooting task, see the single frame or continuous mode shooting function for details
```

16.Setting and obtaining region of Interest (ROI)

Displays the image of the specified region, using the first pixel in the upper left corner as the reference point.

There are two kinds of ROI, ROI is software and hardware ROI respectively, ROI is still camera output whole image software, within the SDK for cutting, so the image read will not shorten the time, cannot improve the continuous mode frame rate, hardware ROI is to point to inside the camera image cropping, only the output image data in specified area,Therefore, it can shorten the image readout time and improve the frame rate in continuous mode.

This function is available after setting the BIN mode. It is optional. The functions used are SetQHYCCDResolution and GetQHYCCDCurrentROI. Below is the example code for displaying a 500x400 image starting from the first pixel in the upper left corner:

```
//Set the ROI
int retVal = QHYCCD_ERROR;

retVal = SetQHYCCDResolution(camHandle, 0, 0, 500, 400);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set ROI Successfully.\n");
}

//Get ROI Settings
uint32_t x, y, sizex, sizey;
retVal = GetQHYCCDCurrentROI(camHandle, &x, &y, &sizex, &sizey);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Get ROI setting successfully.\n");
}
```

17.Gets the range of camera oversweep area

To obtain the overswept area of the image, the image data of the overswept area is needed for dark field correction. If the returned parameter of the overswept area is 0, it means that the camera has no overswept area. The starting position and size of the overswept area are different in different BIN modes.

This function can be used after setting the BIN mode, using the function GetQHYCCDOverScanArea. The following is an example code to obtain the range of the overscan area:

```
uint32_t retVal = QHYCCD_ERROR;
uint32_t startx, starty, sizex, sizey;

retVal = GetQHYCCDOverScanArea(camHandle, &startx, &starty, &sizex, &sizey);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Get overscan area successfully.\n");
```

```
}
```

18.Get the effective area range of the camera

Get the effective area range of the camera, according to which you can set ROI and output the effective area directly.

This function can be used after setting the BIN mode, using the function GetQHYCCDEffectiveArea.

```
uint32_t retVal = QHYCCD_ERROR;
uint32_t startx, starty, sizex, sizey;

retVal = GetQHYCCDEffectiveArea(camHandle, &startx, &starty, &sizex, &sizey);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Get effective area successfully.\n");
}
```

19.Set camera oversweep area correction

Set the camera's oversweep area correction. After setting, the oversweep area will be removed and only the image data of the valid area will be output.

There are two ways to set the over-sweep area correction. The first way is to set ROI. It is necessary to obtain the effective area range first, and then set ROI by SetQHYCCDResolution function to directly output the image data of the effective area.

This function can be used after setting the BIN mode, or can be set directly when setting the BIN mode. The functions used are GetQHYCCDEffectiveArea and SetQHYCCDResolution. Example code is as follows:

```
uint32_t startx, starty, sizex, sizey;

retVal = GetQHYCCDEffectiveArea(camHandle, &startx, &starty, &sizex, &sizey);
if(retVal == QHYCCD_SUCCESS)
{
    retVal = SetQHYCCDResolution(camhandle, startx, starty, sizex, sizey);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set ROI successfully.\n");
    }
}
```

The second method is to set it through the SetQHYCCDParam function. After the setting, only the image data of the effective area will be output. The difference between the two Settings is that the reference point of ROI will change when overscanning area correction and ROI are set at the same time, which will be described in the mixed use function of BIN, ROI and overscanning area below.

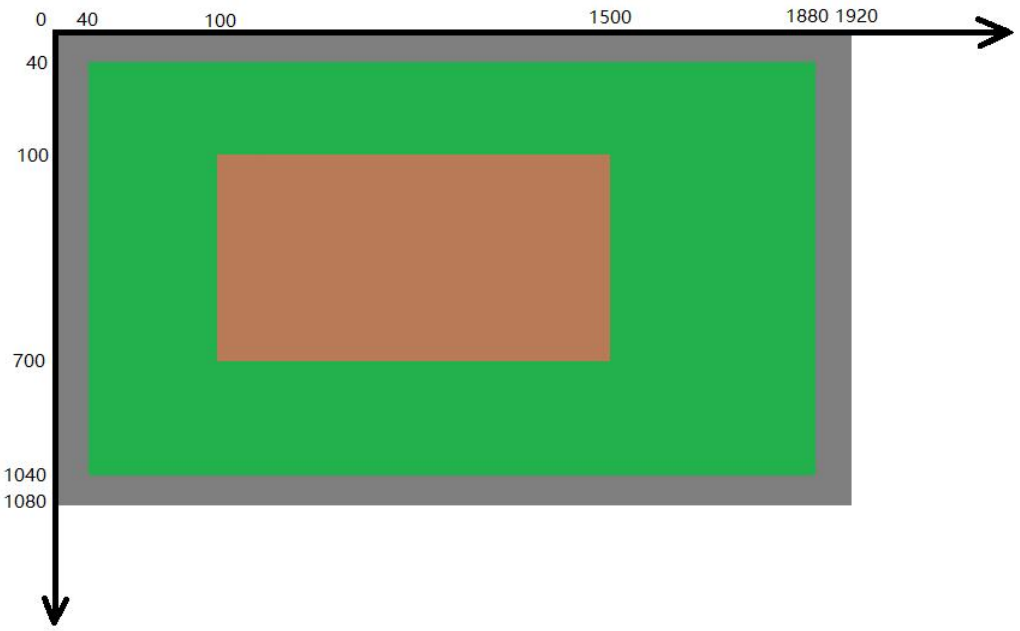
This function can be used after setting the BIN mode. The following is a sample code for setting the oversweep correction:

```
retVal = SetQHYCCDParam(camHandle, CAM_IGNOREOVERSCAN_INTERFACE, 1.0);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Set ignore overscan successfully.\n");
}
```

20.Hybrid use of BIN, ROI and overswept area correction

When using mode 1 to set the oversweep area correction, all the camera's image clipping operations are set with the first pixel in the upper left corner of the original image as the reference point. If the image is compared to the coordinate system, the first pixel in the upper left corner is the coordinate origin.

Assume that an image with full resolution size of 1920x1080 is distributed around the image with a width of 40 and a height of 40. The initial position of the effective region is (40,40) and the size is 1840x1000. The initial position of ROI is (100,100) and the size is 1400x600, as shown in the figure



below:

The code for setting ROI is as follows:

```
retVal = SetQHYCCDResolution(camHandle, 100, 100, 1400, 600);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

The code for setting the oversweep correction is as follows:

```
retVal = SetQHYCCDResolution(camHandle, 40, 40, 1840, 1000);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

After the oversweep area correction is set, the image will only show the effective area, and the starting position of ROI will change from (100, 100) to (60, 60), but in fact, the setting of ROI is still based on the original full-resolution image, and the starting position needs to add the size of oversweep area. If the ROI of the same position and size is set at this time,The code is:

```
retVal = SetQHYCCDResolution(camHandle, 40 + 60, 40 + 60, 1400, 600);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

After setting 2X2 BIN, the oversweep area correction and ROI are similar to those in 1X1 BIN mode, but the size is half of the original. For example, after 2X2 BIN, the image in the example above will become a full-resolution 960x540 image with a width of 20 and a height of 20. The starting position of the effective area is (20,20).Size 920x500, ROI starting position (50,50), size 700x300.

At this point, the code for setting ROI is:

```
retVal = SetQHYCCDResolution(camHandle, 50, 50, 700, 300);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

The code for setting the oversweep correction is:

```
retVal = SetQHYCCDResolution(camHandle, 20, 20, 920, 500);
if(retVal == SetQHYCCD_SUCCESS)
```

```
{
    printf("Setup ROI successfully.\n");
}
```

Similar to the 1X1 BIN mode, after setting the over-sweep area correction, the image will only show the part of the effective area. At this time, from the perspective of the image, the initial ROI position changes from the original (50,50) to (30,30).But in fact, the SETTING of ROI is still based on the image with full resolution in 2X2 BIN mode. The starting position of the setting needs to add the size of the swept area. If the ROI of the same position and size is set at this time, the code is:

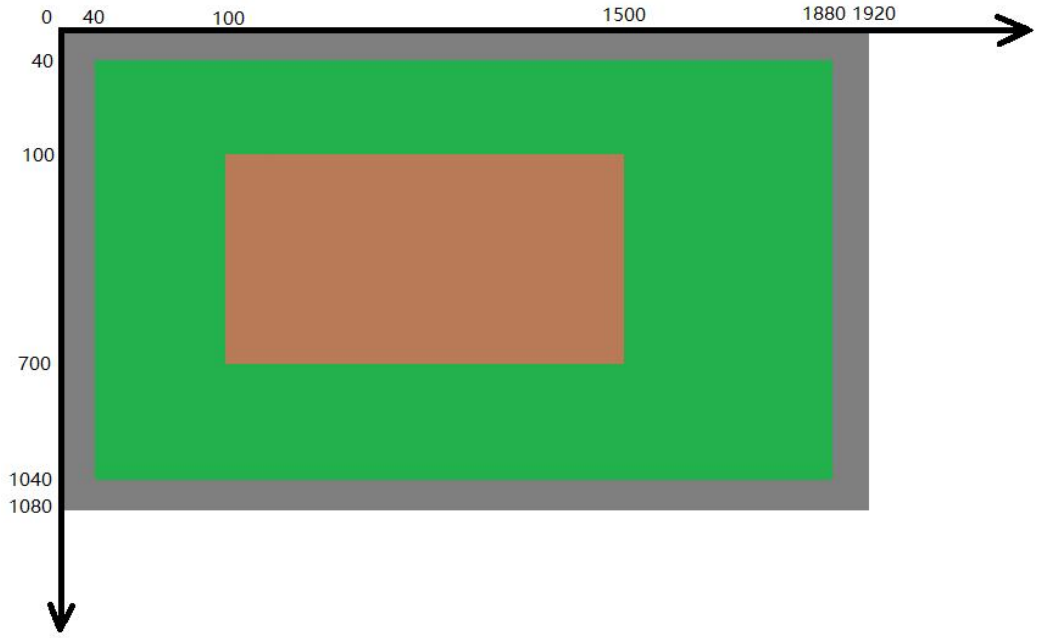
```
retVal = SetQHYCCDResolution(camHandle, 20 + 30, 20 + 30, 920, 500);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

The other BIN modes work the same way and can be set in the same way.

Method 2:

When using method 2 to set the oversweep correction, the ROI reference position will change to the first pixel in the upper left corner of the effective region.

Assume that an image with full resolution size of 1920x1080 is distributed around the image with a width of 40 and a height of 40. The initial position of the effective region is (40,40) and the size is 1840x1000. The initial position of ROI is (100,100) and the size is 1400x600, as shown in the figure



below:

The code for setting ROI is as follows:

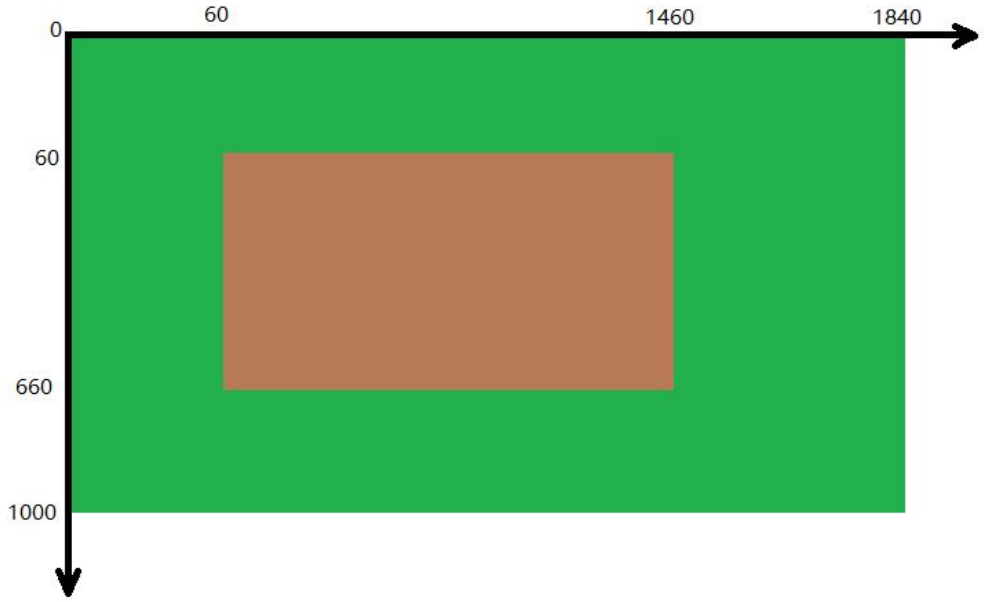
```
retVal = SetQHYCCDResolution(camHandle, 100, 100, 1400, 600);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

The code for setting the oversweep correction is as follows:

```
retVal = SetQHYCCDParam(camHandle, CAM_IGNOREOVERSCAN_INTERFACE, 1.0);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

After the over-sweep area correction is set, the image will only show the effective area, and the starting

position of ROI will change from (100, 100) to (60, 60), as shown



below:

At this point, if the ROI of the same position and size is set, the code is:

```
retVal = SetQHYCCDResolution(camHandle, 60, 60, 1400, 600);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

After setting 2X2 BIN, the oversweep area correction and ROI are similar to those in 1X1 BIN mode, but the size is half of the original. For example, after 2X2 BIN, the image in the example above will become a full-resolution 960x540 image with a width of 20 and a height of 20. The starting position of the effective area is (20,20).Size 920x500, ROI starting position (50,50), size 700x300.

At this point, the code for setting ROI is:

```
retVal = SetQHYCCDResolution(camHandle, 50, 50, 700, 300);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

The code for setting the oversweep correction is:

```
retVal = SetQHYCCDResolution(camHandle, 20, 20, 920, 500);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

Similar to the 1X1 BIN mode, after setting the over-sweep correction, the image will only show the valid area, and the starting ROI position will be changed from (50,50) to (30,30).At this point, if the ROI of the same position and size is set, the code is:

```
retVal = SetQHYCCDResolution(camHandle, 30, 30, 920, 500);
if(retVal == SetQHYCCD_SUCCESS)
{
    printf("Setup ROI successfully.\n");
}
```

Other BIN modes work similarly and can be set up in the same way.

21.Gets the actual number of bits of data output

Gets the actual number of bits of data output by the camera. This number is the actual number of bits of raw data output by the chip.The camera will process the original data internally, and the 8-bit image data can be obtained by taking the high position, or the 16-bit image data can be obtained by adding the low zero position.

You can use this function after formatting the data, using the function GetQHYCCDParam. Here is the sample code for getting the actual bits:

```
uint32_t retVal;
retVal = (uint32_t)GetQHYCCDParam(camHandle, OutputDataActualBits);
if(retVal != QHYCCD_ERRROR)
{
    printf("Camera actual output bits is %d\n", retVal);
}
```

22.Gets the alignment format of the camera output data

Get the alignment format of the camera output data. If the return value is 1, it indicates high alignment; if the return value is 0, it indicates low alignment.

```
uint32_t retVal;
retVal = (uint32_t)GetQHYCCDParam(camHandle, OutputDataAlignment);
if(retVal != QHYCCD_ERRROR)
{
    printf("Get camera output data aligenment successfully.\n");
}
```

23.The length of memory required to get the camera image

The maximum memory length required to obtain the camera image, this function returns a fixed memory length, which will be slightly larger than the actual required value to avoid some unexpected memory overflow problems, and its calculation is as follows:Black and white camera :(imagew+100)*(imageh+100)*2Color camera :(imagew+100)*(imageh+100)*3

This function can be used after the camera is initialized with the InitQHYCCD function GetQHYCCDMemLength. The following is an example code to obtain the memory length:

```
int length = GetQHYCCDMemLength(camhandle);
if(length > 0)
{
    printf("Get image memory length successfully.\n");
}
```

In addition, the required memory length can also be calculated according to the image size and data format as follows:length = imagew * imageh * channels * bits / 8Channels depends on whether the color mode is turned on or off. When the color mode is turned on, Channels value is 3; when the color mode is turned off, Channels value is 1; bits refers to the set image data bits.

24.Sets and gets the exposure time

Before setting the exposure time of the camera, use the IsQHYCCDControlAvailable function to check whether the exposure time can be set. Use GetQHYCCDParamMinMaxStep to obtain the parameter setting range. Check and confirm the parameter range only once.

This function can be used after the camera is initialized with the InitQHYCCD function. IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam functions are used. The following is an example code for setting the exposure time:

```
int retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether exposure time is supported
```

```
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_EXPOSURE);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the exposure time setting range
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_EXPOSURE, &min, &max, &step);
    if(retVal = QHYCCD_SUCCESS)
    {
        printf("Get expose time range successfully.\n");
    }

    //Set exposure time
    retVal = SetQHYCCDParam(camHandle, CONTROL_EXPOSURE, 1000.0);    //Set the exposure time to 1ms
    if(retVal == QHYCCD_SUCCESS)
    {
       printf("Set Exposure Successfully.\n");
    }

    //Get exposure time
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_EXPOSURE);
}
```

Note that the exposure time Settings can be adjusted in either single frame or continuous mode, but if the camera is being exposed, you need to stop the exposure and then start shooting again.

CancelQHYCCDExposingAndReadout function called single frame mode need to terminate exposure, and then adjust the exposure time, and then call ExpQHYCCDSingleFrame function to start shooting.In single-frame mode, however, only one image is taken at a time, so there is little need to worry about changing the exposure time midway through the exposure.Example code for setting exposure time in single frame mode:

```
//This function is executed to terminate exposure if it is being exposed, otherwise not
retVal = CancelQHYCCDExposingAndReadout(camHandle(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Cancel expose successfully.\n");
}

//Set exposure time
retVal = SetQHYCCDParam(camHandle, CONTROL_EXPOSURE, 20000.0);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Set expose time successfully.\n");
}

//This function needs to be executed to restart the exposure if it is being exposed, otherwise not
retVal = ExpQHYCCDSingleFrame(camHandle);
if(retVal != QHYCCD_ERROR)
{
    printf("Begin single capture successfully.\n");
}
```

In continuous mode, both long and short exposures need to be considered.In short exposure mode, frame rate refresh is faster and it does not take too much time, so it can be directly set. However, in long exposure mode, the exposure time setting will take too much time because the exposure time setting will take effect in the next frame. Therefore, after setting the exposure time, it is necessary to call StopQHYCCDLive function to terminate exposure.Call the BeginQHYCCDLive function to start shooting again.The cut-off point between long exposure and short exposure is optional, 1s or 2s or whatever.The following is an example code for setting the exposure time for long exposure shooting in continuous mode:

```
//Set exposure time
retVal = SetQHYCCDParam(camHandle, CONTROL_EXPOSURE, 20000.0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set expose time successfully.\n");
}

//Stop live mode
retVal = StopQHYCCDLive(camHnadle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Stop live capture successfully.\n");
}
//Restart live mode
retVal = BeginQHYCCDLive(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Begin live capture successfully.\n");
}
```

25.Setting Gain

Set the camera gain, the larger the parameter value, the brighter the image.The gain is the multiplication of the camera's exposure by multiplying the image data by a coefficient.The coefficient used in the multiplication operation is determined by the chip, and the gain curve and unit gain of each camera can be viewed on the product interface. This operation will not cause the calculated pixel value to exceed the upper limit, for example, the maximum pixel value of 8-bit images is usually 255, and the maximum pixel value of 16-bit images is usually 65535.Before setting the gain, use the IsQHYCCDControlAvailable function to check whether the gain is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

Since gain is data processing after exposure, it is not necessary to end the shooting task and start again.In addition, due to the difference between single frame mode and continuous mode, the parameter setting ranges of individual cameras in single frame mode and continuous mode are inconsistent.

This function can be used after the camera is initialized with the InitQHYCCD function. IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam functions are used. Example codes for gain Settings are as follows:

```
int retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether gain Settings are supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_GAIN);
if(retVal == QHYCCD_SUCCESS)
{
	//Gets the range and minimum step size for parameter Settings
	retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_GAIN, &min, &max, &step);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Get gain range successfully.\n");
	}

	//Set the gain
	retVal = SetQHYCCDParam(camHandle, CONTROL_GAIN, 1.0);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set gain successfully.\n");
```

```
    }

    //Access to gain
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_OFFSET);
}
```

26.Set Offset

Set the offset value of the camera, the larger the parameter value, the brighter the image.Similar to setting gain, offset is actually processing image data, but the difference is that gain is multiplication and offset is addition. After the exposure, the camera adds a certain value to the original image data. This operation will not cause the calculated pixel value to exceed the upper limit.For example, the maximum pixel value of an 8-bit image is usually 255, and the maximum pixel value of a 16-bit image is usually 65535.

Also, the offset can be set directly without having to stop and start again, and the range of Settings varies from camera to camera depending on single frame and continuous mode.

Before setting the offset, use the IsQHYCCDControlAvailable function to check whether the offset is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

You can use this function after initializing the camera with the InitQHYCCD function, using IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam functions. The following is the example code for setting the offset value:

```
int retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether offset is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_OFFSET);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the value range and minimum step size for parameter Settings
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_OFFSET, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get offset range successfully.\n");
    }

    //Set the Offset
    retVal = SetQHYCCDParam(camHandle, CONTROL_OFFSET, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set offset successfully.\n");
    }

    //Get offset
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_OFFSET);
}
```

27.Set and get Traffic

Set this parameter to adjust the frame rate in continuous mode to obtain the maximum frame rate suitable for the computer. The larger the parameter value is, the slower the frame rate is.Note that this parameter does not directly set the frame rate of continuous mode to a fixed value, but only adjusts it.

Its working principle is, in the back of the image data in a row to add some extra empty data, this can increase the whole image data read time in order to reduce the frame rate, single frame mode can also be set,

but the effect is not obvious, because single frame mode itself read a much slower speed than continuous mode, add a bit of time, even if the effect is not very obvious.

Before setting Traffic, you need to use the IsQHYCCDControlAvailable function to check whether the Traffic setting is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

This function can be used after the camera is initialized with the InitQHYCCD function, using IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam,Example code for setting Traffic:

```
int retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether Traffic is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_USBTRAFFIC);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the value range and minimum step size for parameter Settings
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_USBTRAFFIC, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get traffic range successfully.\n");
    }

    //Set up the Traffic
    retVal = SetQHYCCDParam(camHandle, CONTROL_USBTRAFFIC, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set traffic successfully.\n");
    }

    //To get Traffic
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_USBTRAFFIC);
}
```

28.Sets and gets the RGB component of white balance

Set the CAMERA's RGB gain to make the image closer to its true color.This setting only applies to the continuous mode of the color camera, the larger the value, the higher the color ratio, and similar to the gain setting, can be set at any time.

Before setting the RGB gain, use the IsQHYCCDControlAvailable function to check whether the RGB gain is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. The check and parameter setting range need to be determined only once.

This function can be used after the camera is initialized with the InitQHYCCD function. The functions used are IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep and SetQHYCCDParam. The following is an example code for setting RGB gain:

```
uint32_t retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether red and white balance is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_WBR);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_WBR, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
```

```
    {
        printf("Get red gain range successfully.\n");
    }

    //Set the red and white balance parameters
    retVal = SetQHYCCDParam(camHandle, CONTROL_WBR, 64.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set red gain successfully.\n");
    }

    //Gets red and white balance parameters
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_WBR);
}

//Check whether green and white balance is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_WBG);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_WBG, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get green gain range successfully.\n");
    }

    //Set the green and white balance parameters
    retVal = SetQHYCCDParam(camHandle, CONTROL_WBG, 64.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set green gain successfully.\n").
    }

    //Get the green and white balance parameters
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_WBG);
}

//Check whether blue and white balance is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_WBB);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_WBB, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get blue gain range successfully.\n");
    }

    //Set the blue and white balance parameters
    retVal = SetQHYCCDParam(camHandle, CONTROL_WBB, 64.0);
    if(retVal == QHYCCD_SUCCESS)
    {
```

```
        printf("Set blue gain successfully.\n");
    }

    //Gets the blue and white balance parameters
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_WBB);
}
```

29.Sets and gets brightness

This setting is used to process the image data within the SDK. The default value is 0.0. The higher the value is, the brighter the image will be.

This parameter is unconditional and can be set at any time.Before setting this parameter, use the IsQHYCCDControlAvailable function to check whether the parameter setting is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

This function can be used after the camera is initialized with the InitQHYCCD function, using IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam functions. Below is the example code for setting brightness:

```
uint32_t retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether brightness is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_BRIGHTNESS);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range and step size
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_BRIGHTNESS, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get brightness range successfully.\n");
    }

    //Set the brightness
    retVal = SetQHYCCDParam(camHandle, CONTROL_BRIGHTNESS, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set brightness successfully.\n);
    }

    //Get brightness value
    double value;
    value = GetQHYCCDParam(canHandle, CONTROL_BRIGHTNESS);
}
```

30.Set contrast

This setting is used to process image data within the SDK. The default value is 0.0. The higher the value is, the brighter the image is.

This parameter is unconditional and can be set at any time.Before setting this parameter, use the IsQHYCCDControlAvailable function to check whether the parameter setting is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

This function can be used after the camera is initialized with the InitQHYCCD function, using IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam. The following is the example

code for setting contrast:

```
uint32_t retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether contrast is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_CONTRAST);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range and step size
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_CONTRAST, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get contrast range successfully.\n");
    }

    //Set contrast
    retVal = SetQHYCCDParam(camHandle, CONTROL_CONTRAST, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set contrast Successfully.\n");
    }

    //Gets the contrast setting
    double value;
    value = GetQHYCCDParam(canHandle, CONTROL_CONTRAST);
}
```

31.Set the GAMMA value

Set the Gamma value of the image, which is used to process the image data within the SDK. The default value is 1. The higher the value is, the brighter the image will be.

This parameter is unconditional and can be set at any time.Before setting this parameter, you need to use the IsQHYCCDControlAvailable function to check whether the parameter setting is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

This function can be used after the camera is initialized with the InitQHYCCD function, using IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam. The following is the example code for setting contrast:

```
uint32_t retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether the Gamma setting is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_GAMMA);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range and step size
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_GAMMA, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get gamma range successfully.\n");
    }

    //Set the Gamma
    retVal = SetQHYCCDParam(camHandle, CONTROL_GAMMA, 1.0);
    if(retVal == QHYCCD_SUCCESS)
```

```
    {
        printf("Set gamma successfully.\n");
    }

    //For Gamma value
    double value;
    value = GetQHYCCDParam(camhandle, CONTROL_GAMMA);
}
```

32.Set the image transfer speed

Set the transmission speed of image data. This setting can shorten the read time in single frame mode or improve the frame rate in continuous mode.

Before setting this parameter, you need to use the IsQHYCCDControlAvailable function to check whether the parameter setting is supported, and use the GetQHYCCDParamMinMaxStep function to determine the parameter setting range. You need to determine the parameter setting range only once.

This function can be used after the camera is initialized with the InitQHYCCD function. IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, and SetQHYCCDParam functions are used. The following is an example code for setting the transmission speed:

```
uint32_t retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether speed Settings are supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_SPEED);
if(retVal == QHYCCD_SUCCESS)
{
    //Gets the parameter setting range and step size
    retVal = GetQHYCCDParamMinMaxStep(camHandle, CONTROL_SPEED, &min, &max, &step);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get speed range successfully.\n");
    }

    //Set the speed
    retVal = SetQHYCCDParam(camHandle, CONTROL_SPEED, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set speed successfully.\n");
    }

    //Gets the speed setting value
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_SPEED);
}
```

33.Set the glow suppression function

Set whether to enable or disable the glow suppression function of the camera. When the parameter is set to 1, it is enabled, and when the parameter is set to 0, it is disabled. Enabling this function can reduce the glow effect in the image.

Before setting this parameter, you need to use the IsQHYCCDControlAvailable function to check whether the parameter is supported.

You can use this function after initializing the camera with the InitQHYCCD functions, such as IsQHYCCDControlAvailable and SetQHYCCDParam. Here is the code for setting glow suppression:

```
uint32_t retVal = QHYCCD_ERROR;
```

```
//Check whether glow control is supported
retVal = IsQHYCCDControlAvailable(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
	//Enable glow control
	retVal = SetQHYCCDParam(camHandle, CONTROL_AMPV, 1.0);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set AMPV successfully.\n");
	}

	//Set glow control off
	retVal = SetQHYCCDParam(camHandle, CONTROL_AMPV, 0.0);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set AMPV successfully.\n");
	}

	//Gets the glow control function setting value
	double value;
	value = GetQHYCCDParam(camHandle, CONTROL_AMPV);
}
```

34.Set the DDR

To enable or disable the CAMERA DDR function, set the parameter to 1 and 0.When DDR is enabled, image data can be buffered to avoid image loss caused by data packet loss.In single-frame mode, DDR is enabled by default, but cannot be disabled. In continuous mode, DDR can be enabled or disabled as required.

Before setting this parameter, you need to use the IsQHYCCDControlAvailable function to check whether the parameter is supported.

You can use this function after initializing the camera with the InitQHYCCD functions, such as IsQHYCCDControlAvailable and SetQHYCCDParam. Here is the code for setting glow suppression:

```
uint32_t retVal = QHYCCD_ERROR;

//Check whether the DDR function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_DDR);
if(reVal == QHYCCD_SUCCESS)
{
	//Enable DDR
	retVal = SetQHYCCDParam(camHandle, CONTROL_DDR, 1.0);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set DDR successfully.\n");
	}

	//Disable DDR
	retVal = SetQHYCCDParam(camHandle, CONTROL_DDR, 0.0);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set DDR successfully.\n");
	}

	//Obtain DDR parameter Settings
	double value;
```

```
    value = GetQHYCCDParam(camHandle, CONTROL_DDR);
}
```

35.Set row noise reduction

Set the row noise reduction on or off. When the function is on, the SDK will calculate according to the average value of the oversweep area, so as to reduce the horizontal random fringe.Only qHY5II-M cameras currently support this feature.

This function can be used after initializing the camera with the InitQHYCCD function. The following is an example code for setting the line noise reduction function:

```
uint32_t retVal = QHYCCD_ERROR;

//Check whether the row noise reduction function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_ROWNOISERE);
if(reVal == QHYCCD_SUCCESS)
{
    retVal = SetQHYCCDParam(camHandle, CONTROL_ROWNOISERE, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set row noisere successfully.\n");
    }
}
```

36.Example Set the WDM broadcast function

Enable or disable the WDM broadcast function, through which video images can be sent to multiple target software.For example, if SharpCap is used to connect a camera with WDM function, you can send SharpCap's display video image to other software that supports WDM for display. It is suitable for online video broadcasting applications.

This function can only be used in continuous mode, and currently does not support sending 16-bit image data, currently only supports sending 8-bit black and white and color image data.In addition, you need to install the BroadCast WDM driver. You can download the AllInOne installation package from the official website to install the driver.

This function can be used after the camera is initialized with the InitQHYCCD function. The functions used are IsQHYCCDControlAvailable and SetQHYCCDParam. The following is the example code for enabling the setting function:

```
uint32_t retVal = QHYCCD_ERROR;

//Check whether the broadcast function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_VCAM);
if(reVal == QHYCCD_SUCCESS)
{
    //Enable the broadcast function
    retVal = SetQHYCCDParam(camHandle, CONTROL_VCAM, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set row noisere successfully.\n");
    }

    //Disable the broadcast function
    retVal = SetQHYCCDParam(camHandle, CONTROL_VCAM, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set row noisere successfully.\n");
    }
```

```
    //Gets broadcast function setting parameters
    double value;
    value = GetQHYCCDParam(camHandle, CONTROL_VCAM);
}
```

37.Sets and reads the high-low gain mode

Set the high and low gain. 0 indicates the low gain mode, and 1 indicates the high gain mode. Currently, only QHY5III178 and QHY178 use this function, other cameras have combined the high and low gain Settings into the gain setting function.

This function can be used after the camera is initialized with the InitQHYCCD function. The IsQHYCCDControlAvailable and SetQHYCCDParam functions are used. The following is an example code for setting high and low gain:

```
uint32_t retVal = QHYCCD_ERROR;

//Check whether the high-low gain switchover is supported
retVal = IsQHYCCDControlAvailable(camHandle, CAM_LIGHT_PERFORMANCE_MODE);
if(reVal == QHYCCD_SUCCESS)
{
    //Set the low gain mode
    retVal = SetQHYCCDParam(camHandle, CAM_LIGHT_PERFORMANCE_MODE, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set low gain mode successfully.\n");
    }

    //Set the high gain mode
    retVal = SetQHYCCDParam(camHandle, CAM_LIGHT_PERFORMANCE_MODE, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set high gain mode successfully.\n");
    }

    //Gets the high and low gain mode Settings
    double value;
    value = GetQHYCCDParam(camHandle, CAM_LIGHT_PERFORMANCE_MODE);
}
```

38.Set the guide mode switch for series 5II cameras

Set the star guide mode of series 5II cameras on and off, 1 is on, 0 is off, the default is on.

When you start the guide star pattern will not be able to output the real 16 bits of data, the output of the original data bits or eight, actually the camera raw data of eight low zero padding into 16 bits of data, is turned off, the camera can produce 12 raw data, camera for twelve raw data low zero padding is transformed into 16 bits of data.

This function can be used after the camera is initialized with the InitQHYCCD function. IsQHYCCDControlAvailable and SetQHYCCDParam functions are used. The following is an example code for setting the guide mode:

```
retVal = IsQHYCCDControlAvailable(camHandle, CAM_QHY5II_GUIDE_MODE);
if(reVal == QHYCCD_SUCCESS)
{
    retVal = SetQHYCCDParam(camHandle, CAM_QHY5II_GUIDE_MODE, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set QHY5II guide mode successfully.\n");
```

```
    }
}
```

39.Single frame shooting function

Control the single frame shooting function of the camera, and each exposure can only get one frame of image. If the exposure is not complete, the FUNCTION GetQHYCCDSingleFrame will be called to get the image, and the SDK will block and wait until the exposure is complete and the data is read.

Connect the camera and set after BIN, resolution, number and color patterns can use this function, use the function of ExpQHYCCDSingleFrame, GetQHYCCDSingleFrame and CancelQHYCCDExposingAndReadout,The following is a sample code for single frame mode:

```
int retVal = QHYCCD_ERROR;
int length = 0;
uint32_t w, h, bits, channels;
uint8_t *ImgData;

//Get image data memory length and open up storage space
length = GetQHYCCDLength(camHandle);ImgData = (uint8_t *)malloc(length);

//Start single frame exposure
retVal = ExpQHYCCDSingleFrame(camHandle);
if(retVal != QHYCCD_ERROR)
{
    printf("Start expose successfully.\n");

    //Get single frame mode image data
    retVal = GetQHYCCDSingleFrame(camHandle, &w, &h, &bits, &channels, ImgData);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Get single frame successfully.\n");
    }
}

//End exposure and shooting
retVal = CancelQHYCCDExposingAndReadout(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Stop single frame successfully.\n")
}
```

When you need to stop shooting after the shooting is completed or in some cases, you need to use the end shooting function. The example code is as follows:

```
retVal = CancelQHYCCDExposingAndReadout(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Stop single frame successfully.\n")
}
```

After stopping shooting, ExpQHYCCDSingleFrame function should be called before GetQHYCCDSingleFrame function can continue to obtain image data.

40.Continuous shooting function

Control the continuous shooting function of the camera to obtain video data stream from the camera. Real-time continuous images can be obtained according to the video stream data.In continuous mode, the GetQHYCCDLiveFrame function is continuously called to continuously obtain image data.

This function can be used after connecting the camera and setting BIN, resolution, bit and color mode. The functions used are BeginQHYCCDLive, GetQHYCCDLiveFrame and StopQHYCCDLive. The following is an example

code for shooting thirty frames of images in continuous mode:

```
int retVal = QHYCCD_ERROR;
int length = 0, num = 0;
uint32_t w, h, bits, channels;
uint8_t *ImgData;

//Get image data memory length and open up storage space
length = GetQHYCCDLength(camHandle);
ImgData = (uint8_t *)malloc(length);

//Start continuous mode exposure
retVal = BeginQHYCCDLive(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    while(num < 100)
    {
        //Get continuous image data
        retVal = GetQHYCCDLiveFrame(camHandle, &w, &h, &bits, &channels, ImgData);
        if(retVal == QHYCCD_SUCCESS)
        {
            printf("Get live frame successfully.\n");
            num ++;
        }
    }
}

//End continuous mode exposure and shooting
retVal = StopQHYCCDLive(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Stop live frame successfully.\n");
}
```

To use the end shooting function after the shooting is completed, the sample code is as follows:

```
retVal = StopQHYCCDLive(camHandle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Stop live frame successfully.\n");
}
```

BeginQHYCCDLiveFrame function needs to be called after stopping shooting before GetQHYCCDLiveFrame function can be used to obtain image data. Since image data is obtained by circular call in continuous mode,Therefore, it is safer to exit the loop that gets the data first and then call the StopQHYCCDLive function.

41.Obtain the information about the camera humidity sensor

Obtain the information about the humidity sensor of the camera. The information data is constantly changing, so continuous acquisition is required to obtain real-time information.

Before obtaining this parameter, use the IsQHYCCDControlAvailable function to check whether the function is supported. You need to check this function only once.

This function can be used after the camera is initialized with the InitQHYCCD function. The IsQHYCCDControlAvailable and GetQHYCCDParam functions are used. The following is an example code for obtaining humidity sensor information:

```
int retVal = QHYCCD_ERROR;
double hum;
retVal = IsQHYCCDControlAvailable(camHandle,CAM_HUMIDITY );
if(retVal == QHYCCD_SUCCESS)
```

```
{
    hum = GetQHYCCDParam(camHandle, CAM_HUMIDITY);
    printf("Camera humidity: %f\n", hum)
}
```

42.Obtain camera pressure sensor information

To obtain the information of the camera pressure sensor, the information data will continue to change, so continuous acquisition is needed to obtain real-time information.

Before obtaining this parameter, use the IsQHYCCDControlAvailable function to check whether the function is supported. You need to check this function only once.

This function can be used after the camera is initialized with the InitQHYCCD function. The IsQHYCCDControlAvailable and GetQHYCCDParam functions are used. The following is an example code for obtaining pressure sensor information:

```
int retVal = QHYCCD_ERROR;
double pre;
retVal = IsQHYCCDControlAvailable(camHandle,CAM_PRESSURE );
if(retVal == QHYCCD_SUCCESS)
{
    pre = GetQHYCCDParam(camHandle, CAM_PRESSURE);
    printf("Camera pressure: %f\n", pre);
}
```

43.Set the camera circulation pump switch

Set the camera circulation pump on and off, 1 is on, 0 is off.

This function can be used after the camera is initialized with the InitQHYCCD function. The functions used are IsQHYCCDControlAvailable and SetQHYCCDParam:

```
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_SensorChamberCycle_PUMP);
if(retVal == QHYCCD_SUCCESS)
{
    //Open circulating pump
    retVal = SetQHYCCDParam(camHandle, CONTROL_SensorChamberCycle_PUMP, 1.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set pump on successfully.\n");
    }

    //Shut off circulating pump
    retVal = SetQHYCCDParam(camHandle, CONTROL_SensorChamberCycle_PUMP, 0.0);
    if(retVal == QHYCCD_SUCCESS)
    {
        printf("Set pump off successfully.\n");
    }
}
```

44.Camera temperature control function

The camera's cooling functions include cooling control, reading the current cooling power and the current temperature.

Refrigeration control is divided into manual control and automatic control. Manual control is to directly set the refrigeration power of the refrigerator, so that the camera can be cooled at a certain fixed power.Automatic refrigeration only needs to set the target temperature, and the camera adjusts the cooling power according to the internal algorithm to make the camera temperature at the target temperature.

In automatic mode, the cooling control of CMOS and CCD camera is slightly different. The CMOS camera can automatically adjust the cooling power in the camera driver. The target temperature can be set only once, or it can be set several times in a row.

The camera's cooling power and temperature are constantly changing, so continuous access is needed to get real-time information.

In addition, when using the temperature control function in single-frame mode, it is necessary to temporarily stop setting and obtaining the camera temperature when reading the image data, and then continue using the temperature control function after the camera reads the data.

The refrigeration range of CMOS cameras is generally about 35℃ lower than the ambient temperature. When the camera conducts short exposure, additional heat will be generated, which will reduce the refrigeration range of the camera.Relatively speaking, the refrigeration performance of CCD cameras is better.In addition, the camera does not immediately stabilize at the target temperature when it reaches it, but will fluctuate for a while and then stay at the target temperature.

Before setting the cooling function, call IsQHYCCDControlAvailable to check whether the camera has the cooling function.

This function can be used after the camera is initialized with the InitQHYCCD function. IsQHYCCDControlAvailable, GetQHYCCDParam, and SetQHYCCDParam functions are used.

```
uint32_t retVal = QHYCCD_ERROR;
double pwm, temp, nowpwm, nowtemp;

//Check whether the camera supports cooling
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_COOLER);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Camera has cooler function.\n");
}

//Set the cooling power in manual mode
pwm = 50.0;
retVal = SetQHYCCDParam(camHandle, CONTROL_MANULPWM, pwm / 100.0 * 255.0); //Set the cooling power to 50%
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set manual PWM successfully.\n");
}

//Set the target cooling temperature, automatic mode
temp = -10.0;while(true)
{
	retVal = SetQHYCCDParam(camHandle, CONTROL_COOLER, temp);    //Set the brightness to -10.0
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Set target temperature successfully.\n");
	}

	sleep(1000); //You don't have to set it too often, just once every second
}

//Gets the current cooler power
while(true)
{
	nowpwm = GetQHYCCDParam(camHandle, CONTROL_CURPWM);
	printf("Now PWM : %f%%.\n", nowpwm / 255.0 * 100.0);

	sleep(1000); //每秒获取一次以得到实时数据
}
```

```
//Get current temperature
while(true)
{
	nowtemp = GetQHYCCDParam(camHandle, CONTROL_CURTEMP);
	printf("Now TEMP : %f.\n", nowtemp);

	sleep(1000);//Once a second to get real-time data
}

//Closed refrigeration
retVal = SetQHYCCDParam(camHandle, CONTROL_MANULPWM, 0.0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set cooler off successfully.\n");
}
```

45.Filter wheel control function

Some cameras can connect the filter wheel through the 4PIN interface, and directly control the filter wheel through the SDK. The functions involved include checking whether the camera supports the filter wheel control function, checking whether the filter wheel is connected, obtaining the number of filter wheel holes, setting the target hole position to control the rotation of the filter wheel, and obtaining the current position information of the filter wheel.

The Settings of filter wheel holes range from 0 to cfwNum-1, where cfwNum is the number of filter wheel holes.When the filter wheel rotates to the target position, it will return the position information, so it can continuously obtain the position information of the filter wheel, and confirm whether the filter wheel is still rotating by judging whether the current position of the filter wheel is the same as the target position.

SetQHYCCDParam and SendOrder2QHYCCDCFW have the same function when setting the position of the filter wheel. The difference is the parameter setting method. SetQHYCCDParam sets the parameters of the ASCII code '0', '1', '2'...The corresponding values are 48, 4, 50...And SendOrder2QHYCCDCFW uses the ASCII code of the char type as the parameter to set. When obtaining the position of the filter wheel, GetQHYCCDParam and GetQHYCCDCFWStatus have the same functions.The difference is that GetQHYCCDParam obtains the filter wheel position corresponding to ASCII code, such as 48, 49, 50, etc., while GetQHYCCDCFWStatus obtains the value of CHAR ASCII code.

In addition, if you use C/C++ for development, you can use any function to set or obtain the position of the filter wheel. If you use other languages, it is recommended to use SetQHYCCDParam and GetQHYCCDParam functions, otherwise you may not be able to correctly set the target position or obtain the position information.This function can be used after the camera is initialized with the InitQHYCCD function,The functions used are IsQHYCCDControlAvailable, IsQHYCCDCFWPlugged, SetQHYCCDParam (SendOrder2QHYCCDCFW), and GetQHYCCDParam (GetQHYCCDCFWStatus).SetQHYCCDParam has the same function as SendOrder2QHYCCDCFW, and GetQHYCCDParam has the same function as GetQHYCCDCFWStatus.The following is an example of filter wheel control code:

```
uint32_t retVal = QHYCCD_ERROR;

//Check whether filter wheel control is supported
retVal = IsQHYCCDControlAvailable(camHandle, CONTROL_CFWPORT);
if(retVal == QHYCCD_SUCCESS)
{
	printf("This camera can control CFW.\n");
}

//Get the number of filter wheel holes
int posNum;
posNum = GetQHYCCDParam(camHandle, CONTROL_CFWSLOTSNUM);
printf("This CFW has %s slots.\n");
```

```
//Set the target position of the filter wheel to the third hole, method 1
retVal = SetQHYCCDParam(camHandle, CONTROL_CFWPORT, 50.0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Setup target position successfully.\n");
}
//Set the target position of the filter wheel to the third hole, method two
retVal = SendOrder2QHYCCDCFW(camHandle, '2', 1);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Setup target position successfully.\n");
}

//Obtain the position information of the filter wheel to judge the running state of the filter wheel, method 1
double nowPos;
while(nowPos != targetPos)
{
	nowPos = GetQHYCCDParam(camHandle, CONTROL_CFWPORT);
	printf("Now CFW position is %c.\n", (int)nowPos);
}
//Obtain the position information of the filter wheel to judge the running state of the filter wheel
char nowPos;
while(nowPos != targetPos)
{
	retVal = GetQHYCCDCFWStatus(camHandle, &nowPos);
	if(retVal == QHYCCD_SUCCESS)
	{
		printf("Now CFW position is %c.\n", nowPos);
	}
}
```

46.Set up the GPS

Some cameras support THE GPS function, through which the exposure time of the camera can be measured to obtain the exact time when the frame exposure starts and ends.In addition, the camera can be synchronized through the GPS function.

The GPS function has two working modes, namely master mode and slave mode.Master mode refers to the general working mode that supports ROI.In slave mode, the camera is triggered by time and performs shooting tasks according to the set start time, exposure time and exposure interval. Slave mode does not support ROI.

At present, there are two ways for the camera to receive GPS signals. One is that the camera itself has A GPS interface, which can directly connect the signal receiver to the camera, such as QHY174-GPS; the other is that the camera itself does not have a GPS interface, but connects the GPS signal converter through a 5-core or 6-core GPIO interface, which is connected to the GPS signal receiver.To receive GPS data.

After the GPS signal receiver is connected, it needs to wait for THE GPS signal to lock. The time required for THE GPS signal lock is not fixed due to the influence of signal intensity.After waiting for GPS signal to lock, VCOX and LED need to be calibrated.VCOX calibration is used to eliminate the impact of temperature drift on PPS counters, and LED calibration is used to obtain more accurate exposure start and end times.

The specific operation of VCOX calibration is to adjust the VCOX frequency so that the PPS counter value is close to 1000000. When the PPS signal is lost, the PPS counter value will become 10000500. Therefore, do not let the PPS counter value exceed 10000500 to avoid confusion.

LED calibration is performed by changing Position A and Position B to obtain the exact time of exposure start and end.The specific operation is to observe the LED lamp next to the CMOS. When Position A is added and it is found that the LED pulse is never visible, it is the shutter start time and this Position is the starting Position.When Pposition B is added, the LED pulse is found to change from visible to invisible. This position is the end

position.Each time the exposure time and USBTRAFFIC change, it is necessary to recalibrate Position A and Position B. Below is the setting value of Position A and Position B under different exposure time tested in advance.

8BIT USBTRAFFIC=0			16BIT USBTRAFFIC=0		
EXPOSURE （ms）	START POS （B）	END POS （A）	EXPOSURE （ms）	START POS （B）	END POS （A）
1.0	486080	2850	1.0	902840	4190
2.0	411270	2850	2.0	602930	4190
5.0	185930	2850	5.0	185930	4190
7.0	36290	2850	10.0	228050	4190
8.0	6280	2850	20.0	10930	4190
10.0	6280	2850	30.0	10930	4190
20.0	6280	2850	40.0	10930	4190
40.0	6280	2850	100.0	10930	4190
100.0	6280	2850	200.0	10930	4190
200.0	6280	2850	500.0	10930	4190
500.0	6280	2850			

In theory, both single-frame and continuous modes generate time information, and the received GPS data is stored in the 44th data of the image data.The following is an explanation of the data and the calculation method:

GPS 数据结构：

Frame number			
0	Sequence Number MSB	1	Sequence Number
2	Sequence Number	3	Sequence Number LSB
4	temporary Sequence Number (Normally no use)		
Image Width			
5	Image Width MSB	6	Image Width LSB
Image Height			
7	Image Height MSB	8	Image Height LSB
Latitude			
9	latitude MSB	10	latitude
11	latitude	12	latitude LSB
Longitude			
13	longitude MSB	14	longitude
15	longitude	16	longitude LSB
Shutter start time (JS)			
17	Start_Flag	18	Start Second MSB
19	Start Second	20	Start Second
21	Start Second LSB	22	Start micro second MSB
23	Start micro second	24	Start micro second LSB
Shutter end time (JS)			
25	End flag	26	End Second MSB
27	End Second	28	End Second
29	End Second LSB	30	End micro second MSB
31	End micro second	32	End micro second LSB
State of the GPS			
33	now flag		
Current time (CMOS sensor vSYNC time, not exact shutter start/close time)			

34	now second MSB	35	now second
36	now second	37	now second LSB
38	now micro second MSB	39	now micro second
40	now micro second LSB		
PPS counter value			
41	count of PPS MSB	42	count of PPS
43	count of PPS LSB		

Data conversion method:

```
//The frame serial number
int seqNumber, tempNumber;
seqNumber = 256*256*256*imageHead[0]+256*256*imageHead[1]+256*imageHead[2]+imageHead[3];
tempNumber = imageHead[4];
//The width of the image
int width;width = 256*imageHead[5]+imageHead[6];
//Height of the image
int height;height = 256*imageHead[7]+imageHead[8];
//latitude
int temp, deg, min, south;
double fractMin, latitude;
temp = 256*256*256*imageHead[9]+256*256*imageHead[10]+256*imageHead[11]+imageHead[12];
south = temp > 1000000000;
deg = (temp % 1000000000) / 10000000;min = (temp % 10000000) / 100000;
fractMin = (temp % 100000) / 100000.0;
latitude = (deg + (min + fractMin) / 60.0) * (south==0 ? 1 : -1);

//longitude
int temp, deg, min, west;
double fractMin, longitude;
temp = 256*256*256*imageHead[13]+256*256*imageHead[14]+256*imageHead[15]+imageHead[16];
west = temp > 1000000000;
deg = (temp % 1000000000) / 1000000;min = (temp % 1000000) / 10000;
fractMin = (temp % 10000) / 10000.0;
longitude = (deg + (min + fractMin) / 60.0) * (west==0 ? 1 : -1);
//Shutter start time
int start_flag, start_sec, start_us;
start_flag = imageHead[17];
start_sec = 256*256*256*imageHead[18]+256*256*imageHead[19]+256*imageHead[20]+imageHead[21];
start_us = 256*256*imageHead[22]+256*imageHead[23]+imageHead[24];

//Shutter end time
int end_flag, end_sec, end_us;end_flag = imageHead[25];
end_sec = 256*256*256*imageHead[26]+256*256*imageHead[27]+256*imageHead[28]+imageHead[29];
end_us = 256*256*imageHead[30]+256*imageHead[31]+imageHead[32];
//State of the GPS
int now_flag;
now_flag = (imageHead[33] / 16 ) %4;
//The current time
int now_sec, now_us;
now_sec = 256*256*256*imageHead[34]+256*256*imageHead[35]+256*imageHead[36]+imageHead[37];
now_us = 256*256*imageHead[38]+256*imageHead[39]+imageHead[40];
//PPS count
int pps;pps = 256*256*imageHead[41]+256*imageHead[42]+imageHead[43];
```

```
//Exposure time
double exposeexposure = (end_sec - start_sec) * 1000 * 1000 + (end_us - start_us);
```

Time format conversion:

Start_sec, end_sec, and now_sec are the total number of seconds from October 10, 1995 to the present, which can be converted to the format of year, month, day, hour, minute, and second. The conversion codes are as follows:

```
typedef struct
{
uint16_t year;
uint16_t month;uint16_t date;
uint16_t hour;uint16_t min;
uint16_t sec;uint16_t week;
}drive_time,*pdrive_time;

drive_time UTC;

//Initialization time
drive_time struct_time =
{
.year = 1995,
.month = 10,
.date = 10,
.hour = 0,
.min = 0,
.sec = 0,
};

bool isLeapYear( int year )
{
	if (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0))
		return true;

	return false;
}

//To get utc time, add 8 hours to convert to Beijing time
int get_UTC(unsigned long second, pdrive_time UTC)
{
	const char Leap_Year_day[2][12] =
{ {31,28,31,30,31,30,31,31,30,31,30,31},{31,29,31,30,31,30,31,31,30,31,30,31} };
	int Leap_Year = 0;
	int month_day = 0;

	Leap_Year = isLeapYear(struct_time.year);
	month_day = Leap_Year_day[Leap_Year][struct_time.month-1];

	UTC->year = struct_time.year;
	UTC->month = struct_time.month;
	UTC->date = struct_time.date;
	UTC->hour = struct_time.hour +(second / 3600 % 24);
```

```
    UTC->min = struct_time.min+ (second / 60 % 60);
    UTC->sec = struct_time.sec +(second % 60);

    uint16_t count_days = second / 86400;

    if(UTC->sec >=60)
    {
        UTC->sec = UTC->sec%60;
        (UTC->min) ++;
    }

    if(UTC->min >=60)
    {
        UTC->min = UTC->min%60;
        (UTC->hour) ++;
    }
    if(UTC->hour >=24)
    {
        UTC->hour = UTC->hour%24;
        (count_days) ++;
    }

    for(int i = 0 ; i < count_days; i++ )
    {
        Leap_Year = isLeapYear(UTC->year);
        month_day = Leap_Year_day[Leap_Year][(UTC->month)-1];

        (UTC->date) ++;
        if((UTC->date) > month_day)
        {
            (UTC->date) = 1;
            (UTC->month) ++;
            if((UTC->month) > 12)
            {
                (UTC->month) = 1;
                (UTC->year) ++;
                if( ( (UTC->year) - (struct_time.year) ) >100)
                return -1;
            }
        }
    }

    return    0 ;
}

get_UTC(start_sec, &UTC);
printf("%d %d %d %d %d %d\n", UTC.year, UTC.month, UTC.date, UTC.hour, UTC.min, UTC.sec);
```

This function can be used after the camera is initialized with the InitQHYCCD function, but GPS information can only be obtained after the image data is obtained.The functions used are SetQHYCCDParam, SetQHYCCDGPSVCOXFreq, SetQHYCCDGPSLedCalMode, SetQHYCCDGPSPOSA, SetQHYCCDGPSPOSB, and SetQHYCCDGPSMasterSlaVe, SetQHYCCDGPSSlaveModeParameter, continuous mode is below 200 frames, GPS data and calculate the sample code:

```
int retVal = QHYCCD_ERROR;
unsigned char gps[44] = { 0 };
int length = 0;
int imagew, imageh, bits, channels;
int seqNumber, tempNumber, width, height;
int status, pps;
int temp, deg, min, south, west;
double fractMin, latitude, longitude;
int start_sec, start_us, end_sec, end_us, now_sec, now_us;double exposure;

//Check whether the GPS function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CAM_GPS);
if(retVal == QHYCCD_SUCCESS)
{
	printf("The camera can use GPS function.\n");
}

//Enable THE GPS function
retVal = SetQHYCCDParam(camHandle, CAM_GPS, 1.0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS on successfully.\n");
}

//This setting needs to be adjusted according to the actual situation to make the PPS count close to 1000000. Only QHY174-GPS requires this setting
retVal = SetQHYCCDGPSVCOXFreq(camHandle, 2);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set GPS VCOX frequency successfully.\n");
}

//Set position A and position B of the calibration pulse according to the actual situation. This parameter is required only for QHY174-GPS
SetQHYCCDGPSPOSA(camHandle, 0, pos, 40);
SetQHYCCDGPSPOSB(camHandle, 0, pos, 40);

length = GetQHYCCDMemLength(camHandle);
ImgData = (uint8_t *)malloc(length);

int num = 0;
//Start shooting in continuous mode
retVal = BeginQHYCCDLive(camHandle);
while(num < 200)
{
```

```
    retVal = GetQHYCCDLiveFrame(camHandle, &imagw, &imageh, &bits, &channels, ImgData);
    if(retVal == QHYCCD_SUCCESS)
    {
        retVal = QHYCCD_ERROR;
        num ++;

        memcpy(gps, ImgData, 44);
        //State of the GPS
        now_flag = (gps[33] / 16 ) %4;
        //PPS count
        pps = 256*256*gps[41] + 256*gps[42] + gps[43];
        //Frame number
        seqNumber = 256*256*256*gps[0] + 256*256*gps[1] + 256*gps[2] + gps[3];
        //The width of the image
        width = 256*gps[5] + gps[6];
        //Height of the image
        height = 256*gps[7] + gps[8];
        //latitude
        temp = 256*256*256*gps[9] + 256*256*gps[10] + 256*gps[11] + gps[12];
        south = temp > 1000000000;
        deg = (temp % 1000000000) / 10000000;
        min = (temp % 10000000) / 100000;
        fractMin = (temp % 100000) / 100000.0;
        latitude = (deg + (min + fractMin) / 60.0) * (south==0 ? 1 : -1);
        //longitude
        temp = 256*256*256*gps[13] + 256*256*gps[14] + 256*gps[15] + gps[16];
        west = temp > 1000000000;
        deg = (temp % 1000000000) / 1000000;
        min = (temp % 1000000) / 10000;
        fractMin = (temp % 10000) / 10000.0;
        longitude = (deg + (min + fractMin) / 60.0) * (west==0 ? 1 : -1);
        //Shutter start time
        start_sec = 256*256*256*gps[18] + 256*256*gps[19] + 256*gps[20] + gps[21];
        start_us = 256*256*gps[22] + 256*gps[23] + gps[24];
        //Shutter end time
        end_sec = 256*256*256*gps[26] + 256*256*gps[27] + 256*gps[28] + gps[29];
        end_us = 256*256*gps[30] + 256*gps[31] + gps[32];
        //The current time
        now_sec = 256*256*256*gps[34] + 256*256*gps[35] + 256*gps[36] + gps[37];
        now_us = 256*256*gps[38] + 256*gps[39] + gps[40];
        //Exposure time
        exposure = (end_sec - start_sec) * 1000 * 1000 + (end_us - start_us);
    }
}

//Stop shooting in continuous mode
retVal = StopQHYCCDLive(camHamdle);
if(retVal == QHYCCD_SUCCESS)
{
    printf("Stop live capture successfully.\n");
```

```
}

//Turn off LED and GPS functions
retVal = SetQHYCCDGPSLedCalMode(camHandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS LED off successfully.\n");
}
retVal = SetQHYCCDParam(camHandle, CAM_GPS, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS off successfully.\n");
}
```

Single frame mode using GPS mode and continuous mode similar to the image data of 44 before data is used for calculation, but the difference is that a single frame mode taken only one frame at a time, so if you want to get real time GPS information requires constant, for a frame under the single frame image and calculate the GPS information of the sample code:

```
int retVal = QHYCCD_ERROR;
unsigned char gps[44] = { 0 };
int length = 0;
int imagew, imageh, bits, channels;
int seqNumber, tempNumber, width, height;int status, pps;
int temp, deg, min, south, west;double fractMin, latitude, longitude;
int start_sec, start_us, end_sec, end_us, now_sec, now_us;double exposure;

//Check whether the GPS function is supported
retVal = IsQHYCCDControlAvailable(camHandle, CAM_GPS);
if(retVal == QHYCCD_SUCCESS)
{
	printf("The camera can use GPS function.\n");
}

//Enable THE GPS function
retVal = SetQHYCCDParam(camHandle, CAM_GPS, 1.0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS on successfully.\n");
}

//This setting needs to be adjusted according to the actual situation. The value of PPS is close to 1000000, which is required only for QHY174-GPS
retVal = SetQHYCCDGPSVCOXFreq(camHandle, 2);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set GPS VCOX frequency successfully.\n");
}

//Set position A and position B of the calibration pulse according to the actual situation. This parameter is required only for QHY174-GPS
```

```
SetQHYCCDGPSPOSA(camHandle, 0, pos, 40);
SetQHYCCDGPSPOSB(camHandle, 0, pos, 40);

length = GetQHYCCDMemLength(camHandle);
ImgData = (uint8_t *)malloc(length);

int num = 0;
//Start shooting in continuous mode
retVal = ExpQHYCCDSingleFrame(camHandle);
retVal = GetQHYCCDSingleFrame(camHandle, &imagw, &imageh, &bits, &channels, ImgData);
if(retVal == QHYCCD_SUCCESS)
{
	retVal = QHYCCD_ERROR;
	num ++;

	memcpy(gps, ImgData, 44);
	//State of the GPS
	now_flag = (gps[33] / 16 ) %4;
	//PPS count
	pps = 256*256*gps[41] + 256*gps[42] + gps[43];
	//Frame number
	seqNumber = 256*256*256*gps[0] + 256*256*gps[1] + 256*gps[2] + gps[3];
	//The width of the image
	width = 256*gps[5] + gps[6];
	// Image height
	height = 256*gps[7] + gps[8];
	//latitude
	temp = 256*256*256*gps[9] + 256*256*gps[10] + 256*gps[11] + gps[12];
	south = temp > 1000000000;
	deg = (temp % 1000000000) / 10000000;
	min = (temp % 10000000) / 100000;
	fractMin = (temp % 100000) / 100000.0;
	latitude = (deg + (min + fractMin) / 60.0) * (south==0 ? 1 : -1);
	//longitude
	temp = 256*256*256*gps[13] + 256*256*gps[14] + 256*gps[15] + gps[16];
	west = temp > 1000000000;
	deg = (temp % 1000000000) / 1000000;
	min = (temp % 1000000) / 10000;
	fractMin = (temp % 10000) / 10000.0;
	longitude = (deg + (min + fractMin) / 60.0) * (west==0 ? 1 : -1);
	//Shutter start time
	start_sec = 256*256*256*gps[18] + 256*256*gps[19] + 256*gps[20] + gps[21];
	start_us = 256*256*gps[22] + 256*gps[23] + gps[24];
	//Shutter end time
	end_sec = 256*256*256*gps[26] + 256*256*gps[27] + 256*gps[28] + gps[29];
	end_us = 256*256*gps[30] + 256*gps[31] + gps[32];
	//The current time
	now_sec = 256*256*256*gps[34] + 256*256*gps[35] + 256*gps[36] + gps[37];
	now_us = 256*256*gps[38] + 256*gps[39] + gps[40];
	//Exposure time
	exposure = (end_sec - start_sec) * 1000 * 1000 + (end_us - start_us);
}

//Stop shooting in continuous mode
```

```
retVal = CancelQHYCCDExposingAndReadout(camHamdle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Stop live capture successfully.\n");
}

//Turn off LED and GPS functions
retVal = SetQHYCCDGPSLedCalMode(camHandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS LED off successfully.\n");
}
retVal = SetQHYCCDParam(camHandle, CAM_GPS, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Turn GPS off successfully.\n");
}
```

47.Set the AntiRBI mode

Set the AntiRBI mode of the camera, which can eliminate the effect of residual image.After this mode is enabled, the camera will automatically start shooting with a short exposure frame and a normal exposure interval. From the view of the image, the camera will output the image in a light and dark mode. This function can only be used in continuous mode.

Use function for SetQHYCCDEnableLiveModeAntiRBI, for example code:

```
int ret = QHYCCD_ERROR;
int num = 0, length = 0;
int imagew, imageh, bits, channels;

//Enable the AntiRBI mode
ret = SetQHYCCDEnableLiveModeAntiRBI(camHandle, 1);
if(ret == QHYCCD_SUCCESS)
{
	printf("Enable live mode AntiRBI successfully.\n");
}

//Get image data memory length and open up memory space
length = GetQHYCCDMemLength(camHandle);
ImgData = (uint8_t *)malloc(length);

//Start continuous mode exposure
ret = BeginQHYCCDLive(camHandle);
if(ret == QHYCCD_SUCCESS)
{
	printf("Begin live successfully.\n");
}

while(num < 100)
{
	//Get continuous image data
	ret = GetQHYCCDLiveFrame(camHandle, &imagew, &imageh, &bits, &channels, ImgData);
	if(ret == QHYCCD_SUCCESS)
	{
```

```
            printf("Get live frame successfully.\n");
            num ++;
        }
}
```

48.Setting Burst Mode

Burst mode is a sub-mode of continuous mode. In this mode, the camera will not continuously output image data, but wait for the signal to start shooting. After receiving the signal, the camera will continuously output several frames of image, and the number of frames of output image can be set.

In addition, it should be noted that,

1.When Burst mode is used in optical fiber mode, there will be one less Burst shot for the first time. For example, if start End is set to 1, 6, output 2, 3, 4, 5 will be normal, but in fact, only output 3, 4, 5, and 2 will not be received. Burst images 2, 3, 4 and 5 can be normally obtained in the second and subsequent shooting.This issue will be fixed later.

2.2.QHY2020,QHY4040: when short expose, the index of output frames is [start+1,end-1], when long exposure,the index of output frames is [start+2,end].

3. If the value of "End" is set to a large value when the camera is first connected, the image will be directly generated when the camera enters burst mode. Therefore, it is necessary to set the start End and related burst operations after the camera enters IDLE state.

This feature can be used in continuous mode,Use the function of EnableQHYCCDBurstMode, EnableQHYCCDBurstCountFun, ResetQHYCCDFrameCounter, SetQHYCCDBurstModeStartEnd, SetQHYCCDBurstIDLE, ReleaseQHYCCDBurstIDLE,

The sample code SetQHYCCDBurstModePatchNumber, Burst mode is below:

```
int ret = QHYCCD_ERROR;
int num = 0, length = 0;
int start, end;
int imagew, imageh, bits, channels;

//Get image data memory length and open up memory space
length = GetQHYCCDMemLength(camHandle);
ImgData = (uint8_t *)malloc(length);

//Enable Burst Mode
ret = EnableQHYCCDBurstMode(camHandle, true);
if(ret == QHYCCD_SUCCESS)
{
    printf("Enable burst mode successfully.\n");
}

//Set the position of the start frame and end frame. The camera will output the middle frame
start = 1;
end = 3;
ret = SetQHYCCDBurstModeStartEnd(camHandle, start, end);
if(ret == QHYCCD_SUCCESS)
{
    printf("Set burst mode start end successfully.\n");
}

//Set the amount of supplementary packet data to avoid the image data cannot be output
ret = SetQHYCCDBurstModePatchNumber(camhandle, 32001);
if(ret == QHYCCD_SUCCESS)
{
```

```
        printf("Set burst mode patch number successfully.\n");
}

//Start continuous mode exposure
ret = BeginQHYCCDLive(camHandle);
if(ret == QHYCCD_SUCCESS)
{
        printf("Begin live successfully.\n");
}

//SetQHYCCDBurstIDLE SetQHYCCDBurstIDLE SetQHYCCDBurstIDLE SetQHYCCDBurstIDLE SetQHYCCDBurstIDLE
ret = SetQHYCCDBurstIDLE(camHandle);
if(ret == QHYCCD_SUCCESS)
{
        printf("Set burst mode IDLE successfully.\n");

        ret = ReleaseQHYCCDBurstIDLE(camHandle);
        if(ret == QHYCCD_SUCCESS)
        {
                printf("Release burst mode IDLE successfully.\n");
        }
}

//Get continuous image data
while(true)
{
        retVal = GetQHYCCDLiveFrame(camHandle, &imagew, &imageh, &bits, &channels, ImgData);
        if(retVal == QHYCCD_SUCCESS)
        {
                printf("Get live frame successfully.\n");
                num ++;
        }
}

//Note: It is not necessary to end continuous mode after shooting burst mode images, but to end continuous
mode after all shooting is completed. If continuous mode is not finished, you can also reset start and End
```

49.Set external trigger mode

Set the trigger function of the camera. The trigger function is divided into two kinds: trigger input and trigger output.

Trigger input function, from the outside sends a trigger signal to control the camera, when enabled will not be able to trigger mode by invoking the original the exposure function to control the camera shooting task, at this moment need input from outside trigger signal to the camera, the camera after receiving the trigger signal will start to exposure, exposure after the completion of calls to functions of image data can obtain image data,Trigger signals can be generated by software or hardware.

Trigger output function: when the camera takes an image, it outputs the pulse related to exposure, according to which exposure information can be obtained.

Trigger functions can be used in both single frame and continuous mode. Trigger functions vary from camera to camera. Trigger input and trigger output functions can be used at the same time, or only one of them can be used.

This function can be used when the camera is connected, but you must set BIN, resolution, bit and color mode before shooting tasks.The functions used are IsQHYCCDControlAvailable, GetQHYCCDParamMinMaxStep, SetQHYCCDTrigerFunction, SetQHYCCDTrigerMode. The following is an example code for using this function in single frame mode:

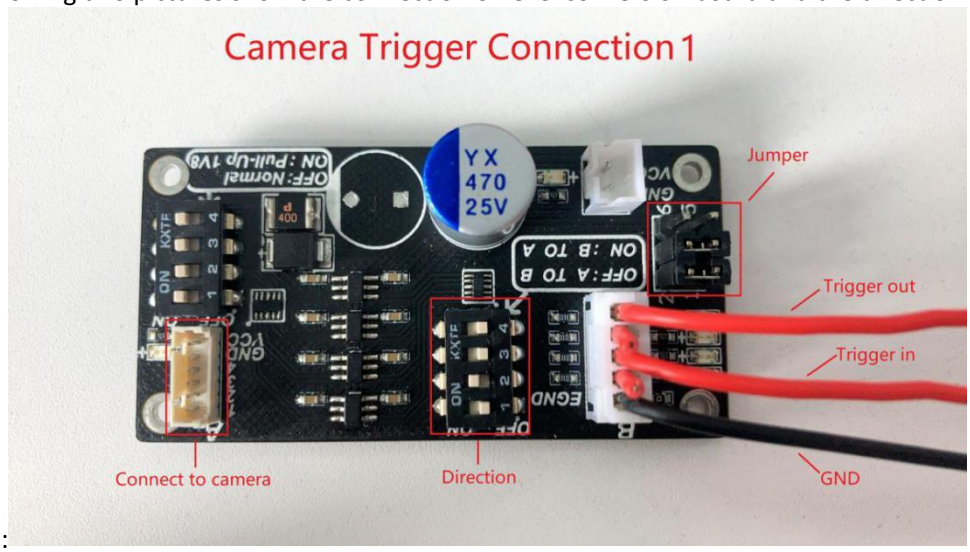
```
int retVal = QHYCCD_ERROR;
double min, max, step;

//Check whether the camera supports triggering
retVal = IsQHYCCDControlAvailable(camHandle, CAM_TRIGER_INTERFACE);
if(retVal == QHYCCD_SUCCESS)
{
	printf("This camera can use trigger function.\n");
}

//Gets the number and name of the trigger interface
uint32_t num = 0;
retVal = GetQHYCCDTrigerInterfaceNumber(camhandle, &num);
if(num > 1)
{
	char name[40] = { 0 };
	for(int i = 0; i < num; i ++)
	{
		retVal = GetQHYCCDTrigerInterName(camhandle, i, name);
		if(retVal == QHYCCD_SUCCESS)
		{
			printf("Get triger interface successfully.\n");
		}
	}
}

//Setting the trigger Interface
retVal = SetQHYCCDTrigerInterface(camhandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set interface successfully.\n");
}

//The trigger function was enabled
retVal = SetQHYCCDTrigerFunction(camHandle, true);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Enable trigger mode successfully.\n");
}

//Enable the trigger output function
retVal = EnableQHYCCDTrigerOut(camhandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Enable trigger out successfully.\n");
}

//Start single frame shooting
retVal = ExpQHYCCDSingleFrame(camHandle);
```

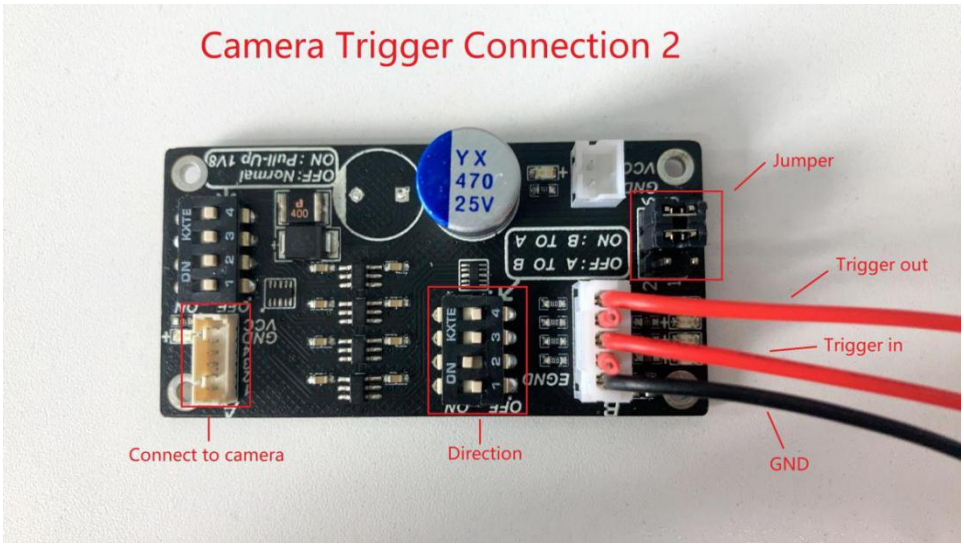
```
if(retVal != QHYCCD_ERROR)
{
	printf("Start single expose successfully.\n");
}

//Block to get single frame data
retVal = GetQHYCCDSingleFrame(camHandle, &imagew, &imageh, &bpp, &channels, ImgData);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Get single frame successfully.\n");
}

//End single frame shooting
retVal = CancelQHYCCDExposingAndReadout(camhandle);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Stop single capture successfully.\n");
}
```

Trigger control hardware description:

At present, there are two kinds of camera trigger interface, one is GPIO interface, the other is SMA interface.

If GPIO interface is used, it is recommended to use the matching level conversion board for control.QHY4040, QHY4040PRO, QHY600, QHY268, QHY461 and QHY411 can be controlled by GPIO interface. These cameras are all triggered by falling edge, but the trigger level standard is different. QHY4040 is triggered by 2.5V.QHY4040PRO, QHY600, QHY268, QHY461 and QHY411 are 1.8V trigger. Taking cameras with 6PIN GPIO interface as an example, the following two pictures show the connection of level conversion board and the direction of topless switch:





Connection1's Trigger Out level is fixed at 5V, while Connection2's Trigger Out level is 1.8V or 2.5V.QHY4040PRO, QHY600, QHY268, QHY461, QHY411 are 1.8V, QHY4040 is 2.5V.It should also be noted that QHY4040 and QHY4040PRO can only use either Trigger In or Trigger Out at the same time due to hardware reasons.Level conversion board description:http://note.youdao.com/noteshare?id=4b265780689cadb651452299be5b19dd&sub=6D04551AD06E41548FE9D341BC9DC48E

There are currently cameras that use SMA trigger interfaces are QHY4040、QHY411ERIS、QHY461、QHY550、QHY990、QHY991、QHY21、QHY22, QHY23, A series cameras, all use the same level standard triggered by SMA interface, 1.8~2.5V, less than 20mA power input, trigger mode is falling edge trigger, THE inner core of SMA interface is positive, the outer copper port is GND.In addition, some cameras that use the SMA Trigger interface have only one Trigger In interface and no Trigger Out interface, so only the Trigger In function can be used.https://note.youdao.com/share/?token=B19CB8A8576141279B0AB6E16913792D&gid=7234866

Ⅳ.sample program

1.Single frame mode

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
    int num = 0;
    qhyccd_handle *camhandle = NULL;
    int ret = QHYCCD_ERROR;
    char id[32];
    int found = 0;
    unsigned int w,h,bpp,channels;
    unsigned char *ImgData;
    double chipw,chiph,pixelw,pixelh;
```

```
//To initialize the SDK
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get the camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open the camera and get the device handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}

	//Set read mode
	ret = SetQHYCCDReadMode(camhandle, 0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set read mode failed.\n");
```

```
        goto failure;
}

//Set the single frame mode
ret = SetQHYCCDStreamMode(camhandle, 0);
if(ret == QHYCCD_ERROR)
{
        printf("Set stream mode failed.\n");
        goto failure;
}

//Initializing the camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
        printf("Init camera failed.\n",ret);
        goto failure;
}

//Obtain camera hardware information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
        printf("GetQHYCCDChipInfo success!\n");
        printf("CCD/CMOS chip information:\n");
        printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
        printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
        printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
        printf("GetQHYCCDChipInfo fail\n");
        goto failure;
}

//Set the BIN
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
        printf("Set BIN mode failed.\n");
        goto failure;
}

//Set resolution
```

```
ret = SetQHYCCDResolution(camhandle, 0, 0, w, h);
if(ret == QHYCCD_ERROR)
{
	printf("Set resolution failed.\n");
	goto failure;
}

//Set the Color
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
	printf("This is a color camera.\n");
	ret = SetQHYCCDDebayerOnOff(camhandle,false);
	if(ret == QHYCCD_ERROR)
	{
		printf("Setup color failed.\n");
		goto failure;
	}
}

//Set the Bits
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set bits failed.\n");
		goto failure;
	}
}

//Set up the Traffic
ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
	if(ret == QHYCCD_ERROR)
	{
		printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
		goto failure;
	}
}
```

```
//Set the Gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gain failed.\n");
		goto failure;
	}
}

//Set the Offset
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set offset failed.\n");
		goto failure;
	}
}

//Set the DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Set exposure time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 1000000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
}
```

```
        //Obtain image data memory length, open up memory space
        uint32_t length = GetQHYCCDMemLength(camhandle);
        if(length > 0)
        {
            ImgData = (unsigned char *)malloc(length);
            memset(ImgData,0,length);
        }
        else
        {
            printf("Get the min memory space length failure \n");
            goto failure;
        }

        //Expose
        ret = ExpQHYCCDSingleFrame(camhandle);
        if( ret == QHYCCD_ERROR )
        {
            printf("Start single expose failed.\n");
            goto failure;
        }

        //Get image data
        ret = GetQHYCCDSingleFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
        if(ret == QHYCCD_SUCCESS)
        {
            printf("Get single frame succeessfully.\n");
            //show the image,this function need you do in software
        }
        else
        {
            printf("Get single frame failed.\n",ret);
        }

        delete(ImgData);
    }
    else
    {
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
    }

    if(camhandle)
    {
        //Stop capture
```

```
            ret = CancelQHYCCDExposingAndReadout(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("Cancel exposing and readout failed.\n");
                goto failure;
            }

            //Close camera
            ret = CloseQHYCCD(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("Close camera failed.\n");
                goto failure;
            }
        }

        //Release SDK Resource
        ret = ReleaseQHYCCDResource();
        if(ret == QHYCCD_ERROR)
        {
            printf("Rlease SDK resource failed.\n");
            goto failure;
        }

        return 0;

failure:
        printf("Some fatal error happened.\n");
        return 1;
}
```

2.Live Mode

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
        int num = 0;
        qhyccd_handle *camhandle = NULL;
        int ret = QHYCCD_ERROR;
```

```
char id[32];
int found = 0;
unsigned int w,h,bpp,channels;
unsigned char *ImgData;
double chipw,chiph,pixelw,pixelh;

//Initialize SDK
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open camera,get handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}
```

```
//Setup read mode
ret = SetQHYCCDReadMode(camhandle, 0);
if(ret == QHYCCD_ERROR)
{
	printf("Set read mode failed.\n");
	goto failure;
}

//Set live mode
ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
	printf("Set stream mode failed.\n");
	goto failure;
}

//Initialize camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("Init camera failed.\n",ret);
	goto failure;
}

//Get chip information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo success!\n");
	printf("CCD/CMOS chip information:\n");
	printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
	printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
	printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
	printf("GetQHYCCDChipInfo fail\n");
	goto failure;
}

//Setup BIN mode
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
```

```
        printf("Set BIN mode failed.\n");
        goto failure;
    }

    //Setup resolution
    ret = SetQHYCCDResolution(camhandle, 0, 0, w, h);
    if(ret == QHYCCD_ERROR)
    {
        printf("Set resolution failed.\n");
        goto failure;
    }

    //Setup color mode
    ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
    if(ret == QHYCCD_SUCCESS)
    {
        printf("This is a color camera.\n");
        ret = SetQHYCCDDebayerOnOff(camhandle,false);
        if(ret == QHYCCD_ERROR)
        {
            printf("Setup color failed.\n");
            goto failure;
        }
    }

    //Setup bits mode
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set bits failed.\n");
            goto failure;
        }
    }

    //Setup gain
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
        if(ret == QHYCCD_ERROR)
        {
```

```
            printf("Set gain failed.\n");
            goto failure;
        }
    }

    //Setup offset
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set offset failed.\n");
            goto failure;
        }
    }

    //Setup traffic
    ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
        if(ret == QHYCCD_ERROR)
        {
            printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
            goto failure;
        }
    }

    //Setup red balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBR);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_WBR,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set red balance failed.\n");
            goto failure;
        }
    }

    //Setup green balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBG);
    if(ret == QHYCCD_SUCCESS)
```

```
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBG,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set green balance failed.\n");
		goto failure;
	}
}

//Setup blue balance
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBB);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBB,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set blue balance failed.\n");
		goto failure;
	}
}

//设置 brightness
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_BRIGHTNESS);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_BRIGHTNESS,0.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set brightness failed.\n");
		goto failure;
	}
}

//Setup contrast
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_CONTRAST);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_CONTRAST,0.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set contrast failed.\n");
		goto failure;
	}
}
```

```
//Setup Gamma
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAMMA);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAMMA,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gamma failed.\n");
		goto failure;
	}
}

//Setup DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Setup expose time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 10000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
}

//Get image data length and get memory space
uint32_t length = GetQHYCCDMemLength(camhandle);
if(length > 0)
{
	ImgData = (unsigned char *)malloc(length);
	memset(ImgData,0,length);
}
else
{
	printf("Get the min memory space length failure \n");
	goto failure;
```

```
        }

        //Start to expose
        ret = BeginQHYCCDLive(camhandle);
        if( ret == QHYCCD_ERROR )
        {
            printf("Start live expose failed.\n");
            goto failure;
        }

        int fame = 0;
        //Get image data
        while(frame < 100)
        {
            ret = QHYCCD_ERROR;
            while(ret != QHYCCD_SUCCESS)
            {
                ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
            }

            printf("Get live frame succeessfully.\n");
            frame ++;
            //show the image,this function need you do in software
        }

        delete(ImgData);
}
else
{
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
}

if(camhandle)
{
        //Stop capture
        ret = StopQHYCCDLive(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Stop live capture failed.\n");
            goto failure;
        }

        //Close camera
```

```
            ret = CloseQHYCCD(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("Close camera failed.\n");
                goto failure;
            }
        }

        //Release SDK resource
        ret = ReleaseQHYCCDResource();
        if(ret == QHYCCD_ERROR)
        {
            printf("Rlease SDK resource failed.\n");
            goto failure;
        }

        return 0;

failure:
        printf("Some fatal error happened.\n");
        return 1;
}
```

3. Live mode switches between readout mode, BIN mode, and data format

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
        int num = 0;
        qhyccd_handle *camhandle = NULL;
        int ret = QHYCCD_ERROR;
        char id[32];
        int found = 0;
        int fame = 0;
        unsigned int w,h,bpp,channels;
        unsigned char *ImgData;
        double chipw,chiph,pixelw,pixelh;

        //Initialize SDK
```

```
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open camera,get camera handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}

	//Setup read mode
	ret = SetQHYCCDReadMode(camhandle, 0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set read mode failed.\n");
		goto failure;
	}
```

```
//Setup live mode
ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
	printf("Set stream mode failed.\n");
	goto failure;
}

//Initialize camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("Init camera failed.\n",ret);
	goto failure;
}

//Get chip information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo success!\n");
	printf("CCD/CMOS chip information:\n");
	printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
	printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
	printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
	printf("GetQHYCCDChipInfo fail\n");
	goto failure;
}

//Setup BIN mode
cambinx = 1;
cambiny = 1;
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
	printf("Set BIN mode failed.\n");
	goto failure;
}
ret = SetQHYCCDResolution(camhandle, 0, 0, w/cambinx, h/cambiny);
if(ret == QHYCCD_ERROR)
```

```
{
	printf("Set resolution failed.\n");
	goto failure;
}

//Setup color mode
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
	printf("This is a color camera.\n");
	ret = SetQHYCCDDebayerOnOff(camhandle,false);
	if(ret == QHYCCD_ERROR)
	{
		printf("Setup color failed.\n");
		goto failure;
	}
}

//Setup bits mode
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set bits failed.\n");
		goto failure;
	}
}

//Setup traffic
ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
	if(ret == QHYCCD_ERROR)
	{
		printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
		goto failure;
	}
}

//Setup gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
```

```
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gain failed.\n");
		goto failure;
	}
}

//Setup offset
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set offset failed.\n");
		goto failure;
	}
}

//Setup DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Setup expose time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 1000000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
}

//Get image data length,get memory space
uint32_t length = GetQHYCCDMemLength(camhandle);
```

```
if(length > 0)
{
	ImgData = (unsigned char *)malloc(length);
	memset(ImgData,0,length);
}
else
{
	printf("Get the min memory space length failure \n");
	goto failure;
}

//Start to expose
ret = BeginQHYCCDLive(camhandle);
if( ret == QHYCCD_ERROR )
{
	printf("Start live expose failed.\n");
	goto failure;
}

//Get image data
while(frame < 100)
{
	ret = QHYCCD_ERROR;
	while(ret != QHYCCD_SUCCESS)
	{
		ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
	}

	printf("Get live frame succeessfully.\n");
	frame ++;
	//show the image,this function need you do in software
}

//Stop capture
ret = StopQHYCCDLive(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("Stop live capture failed.\n");
	goto failure;
}

//Close camera
ret = CloseQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
```

```
{
    printf("Close camera failed.\n");
    goto failure;
}

/*****************************************************************************/
/********** 从新获取句柄，并设置读出模式、BIN 模式、数据格式等相关参数 **********/
/*****************************************************************************/
//Open camera
camhandle = OpenQHYCCD(id);
if(camhandle == NULL)
{
    printf("Open QHYCCD fail \n");
    goto failure;
}

//Setup read mode
ret = SetQHYCCDReadMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
    printf("Set read mode failed.\n");
    goto failure;
}

//Setup live mode
ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
    printf("Set stream mode failed.\n");
    goto failure;
}

//Initialize camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
    printf("Init camera failed.\n",ret);
    goto failure;
}

//Setup BIN mode
cambinx = 2;
cambiny = 2;
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
```

```
if(ret == QHYCCD_ERROR)
{
	printf("Set BIN mode failed.\n");
	goto failure;
}
ret = SetQHYCCDResolution(camhandle, 0, 0, w/cambinx, h/cambiny);
if(ret == QHYCCD_ERROR)
{
	printf("Set resolution failed.\n");
	goto failure;
}

//Setup color mode
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
	printf("This is a color camera.\n");
	ret = SetQHYCCDDebayerOnOff(camhandle, false);
	if(ret == QHYCCD_ERROR)
	{
		printf("Setup color failed.\n");
		goto failure;
	}
}

//Setup bits mode
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 8);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set bits failed.\n");
		goto failure;
	}
}

//Setup gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
	if(ret == QHYCCD_ERROR)
	{
```

```
            printf("Set gain failed.\n");
            goto failure;
        }
    }

    //Setup offset
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set offset failed.\n");
            goto failure;
        }
    }

    //Setup traffic
    ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
        if(ret == QHYCCD_ERROR)
        {
            printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
            goto failure;
        }
    }

    //Setup red balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBR);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_WBR,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set red balance failed.\n");
            goto failure;
        }
    }

    //Setup green balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBG);
    if(ret == QHYCCD_SUCCESS)
```

```
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBG,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set green balance failed.\n");
		goto failure;
	}
}

//Setup blue balance
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBB);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBB,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set blue balance failed.\n");
		goto failure;
	}
}

//Setup brightness
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_BRIGHTNESS);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_BRIGHTNESS,0.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set brightness failed.\n");
		goto failure;
	}
}

//Setup contrast
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_CONTRAST);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_CONTRAST,0.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set contrast failed.\n");
		goto failure;
	}
}
```

```
//Setup gamma
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAMMA);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAMMA,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gamma failed.\n");
		goto failure;
	}
}

//Setup DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Setup expose time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 10000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
}

//Start to capture
ret = BeginQHYCCDLive(camhandle);
if( ret == QHYCCD_ERROR )
{
	printf("Start live expose failed.\n");
	goto failure;
}

//Get image data
while(frame < 100)
{
```

```
            ret = QHYCCD_ERROR;
            while(ret != QHYCCD_SUCCESS)
            {
                ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
            }

            printf("Get live frame succeessfully.\n");
            frame ++;
            //show the image,this function need you do in software
        }

        //Stop capture
        ret = StopQHYCCDLive(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Stop live capture failed.\n");
            goto failure;
        }

        delete(ImgData);
    }
    else
    {
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
    }

    if(camhandle)
    {
        //Close camera
        ret = CloseQHYCCD(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Close camera failed.\n");
            goto failure;
        }

        //Release SDK resource
        ret = ReleaseQHYCCDResource();
        if(ret == QHYCCD_ERROR)
        {
        printf("Rlease SDK resource failed.\n");
        goto failure;
        }
```

```
    }

    return 0;

failure:
    printf("Some fatal error happened.\n");
    return 1;
}
```

4.Camera refrigeration and humidity pressure sensor

Automatic mode

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include "qhyccd.h"

qhyccd_handle *camhandle = NULL;

int main(int argc,char *argv[])
{
    uint32_t ret = QHYCCD_ERROR;
    int num = 0;
    int found = 0;
    char id[32];
    double nowTemp, nowPWM;
    int time = 0;

    ret = InitQHYCCDResource();
    if(ret == QHYCCD_ERROR)
    {
        printf("Initialize SDK resource failed.\n");
        goto failure;
    }

    num = ScanQHYCCD();
    if(num <= 0)
    {
        printf("Not Found QHYCCD,please check the usblink or the power.\n");
        goto failure;
    }
```

```
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("Connected to the first camera from the list,id is %s.\n",id);
		found = 1;
	}
}

if(found == 1)
{
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open camera failed.\n");
		goto failure;
	}

	ret = SetQHYCCDStreamMode(camhandle,0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set stream mode failed.\n");
		goto failure;
	}

	ret = InitQHYCCD(camhandle);
	if(ret == QHYCCD_ERROR)
	{
		printf("Init camera failed.\n");
		goto failure;
	}

	/*********************************************************************************/
	/*********************CMOS cameras can be set only once or repeatedly. *********************/
	/******************Automatic mode and manual mode cannot be used ***********************/
	/*********************at the same time.                                    *********************/
	/*********************************************************************************/
	//Cycle control camera cooling for 10min
	while(time < 600)
	{
		//Set the target temperature to -10 degrees
		ret = SetQHYCCDParam(camhandle, CONTROL_COOLER, -10.0);
		if(ret == QHYCCD_ERROR)
```

```
            {
                printf("Set target temperature failed.\n");
            }

            nowTemp = GetQHYCCDParam(camhandle,CONTROL_CURTEMP);//Get current temperature
            nowPWM = GetQHYCCDParam(camhandle,CONTROL_CURPWM);//Get current power
            printf("Temperature:%.1f°  C,PWM:%.1f%%.\n",nowTemp,nowPWM/255.0 * 100);

            time ++;
            sleep(1000); //Delay 1 s
        }

        //Turn off camera cooling
        ret = SetQHYCCDParam(camhandle, CONTROL_MANULPWM, 0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Close camera cooler failed!(%d)\n",ret);
        }
    }

    if(camhandle)
    {
        ret = CloseQHYCCD(camhandle);//Close the camera
        if(ret == QHYCCD_ERROR)
        {
            printf("Close camera failed.\n");
            goto failure;
        }

        ret = ReleaseQHYCCDResource();//Release camera resources
        if(ret == QHYCCD_ERROR)
        {
            printf("Rlease SDK resource failed.\n");
            goto failure;
        }
    }

    return 0;

failure:
    printf("some fatal error happened\n");
    return 1;
}
```

Manual mode

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include "qhyccd.h"

qhyccd_handle *camhandle = NULL;

int main(int argc,char *argv[]){
	int ret = QHYCCD_ERROR;
	int found = 0, num = 0;
	char id[32];

	ret = InitQHYCCDResource();
	if(ret == QHYCCD_ERROR)
	{
		printf("Initialize SDK resource failed.\n");
		goto failure;
	}

	num = ScanQHYCCD();
	if(num <= 0)
	{
		printf("Not Found QHYCCD,please check the usblink or the power.\n");
		goto failure;
	}

	for(int i = 0;i < num;i++)
	{
		ret = GetQHYCCDId(i,id);
		if(ret == QHYCCD_SUCCESS)
		{
			printf("Connected to the first camera from the list,id is %s.\n",id);
			found = 1;
		}
	}

	if(found == 1)
	{
		camhandle = OpenQHYCCD(id);
		if(camhandle == NULL)
		{
```

```
        printf("Open camera failed.\n");
        goto failure;
    }

    ret = SetQHYCCDStreamMode(camhandle,0);
    if(ret == QHYCCD_ERROR)
    {
        printf("Set stream mode failed.\n");
        goto failure;
    }

    ret = InitQHYCCD(camhandle);
    if(ret == QHYCCD_ERROR)
    {
        printf("Init camera failed.\n");
        goto failure;
    }

    int time = 0;
    double nowTemp, nowPWM;

    /*****************************************************************************/
    /***********Manual mode, this mode needs to set the cooling power, ******************/
    /***********CCD camera and CMSO camera only need to set once, *********************/
    /***********manual mode and automatic mode can not be used at**********************/
    /*********** the same time.                                          **********************/
    /*****************************************************************************/
    //Set the cooling power to 70%
    double pwm = 70.0;
    ret = SetQHYCCDParam(camhandle, CONTROL_MANULPWM, pwm / 100.0 * 255.0);
    if(ret == QHYCCD_ERROR)
    {
        printf("Set camera PWM failed.\n");
    }

    //Continuously obtain the camera temperature for 10min
    while(time < 600)
    {
        nowTemp = GetQHYCCDParam(camhandle,CONTROL_CURTEMP);//Get current temperature
        nowPWM = GetQHYCCDParam(camhandle,CONTROL_CURPWM);//Get current power
        printf("Temperature:%.1f°  C,PWM:%.1f%%.\n",nowTemp,nowPWM/255.0 * 100);

        time ++;
        sleep(1000); //Delay 1s
```

```
            }

            //Turn off camera cooling
            ret = SetQHYCCDParam(camhandle, CONTROL_MANULPWM, 0);
            if(ret == QHYCCD_ERROR)
            {
                printf("Close camera cooler failed!(%d)\n",ret);
            }
        }
        if(camhandle)
        {
            ret = CloseQHYCCD(camhandle);//Close the camera
            if(ret == QHYCCD_ERROR)
            {
                printf("Close QHYCCD failed.\n");
                goto failure;
            }

            ret = ReleaseQHYCCDResource();//Release camera resources
            if(ret == QHYCCD_ERROR)
            {
                printf("Rlease SDK resource failed.\n");
                goto failure;
            }
        }

        return 0;

failure:
        printf("some fatal error happened\n");
        return 1;
}
```

5.Filter wheel control

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include "qhyccd.h"

qhyccd_handle *camhandle = NULL;
```

```
int main(int argc,char *argv[])
{
	int ret = QHYCCD_ERROR;
	int found = 0, num = 0;
	char id[32];
	double status;

	ret = InitQHYCCDResource();//Initialize SDK resources
	if(ret == QHYCCD_ERROR)
	{
		printf("Initialize SDK resource failed.\n");
		goto failure;
	}

	num = ScanQHYCCD();//Scan camera
	if(num <= 0)
	{
		printf("Not Found QHYCCD,please check the usblink or the power.\n");
		goto failure;
	}

	for(int i = 0;i < num;i++)
	{
		ret = GetQHYCCDId(i,id);//Get the camera ID
		if(ret == QHYCCD_SUCCESS)
		{
			printf("Connected to the first camera from the list,id is %s.\n",id);
			found = 1;
		}
	}

	if(found == 1)
	{
		camhandle = OpenQHYCCD(id);//Open the camera
		if(camhandle == NULL)
		{
			printf("Open camera failed.\n");
			goto failure;
		}

		ret = SetQHYCCDReadMode(camhandle,0);//Set read mode
		if(ret == QHYCCD_ERROR)
		{
			printf("Set read mode failed.\n");
```

```
            goto failure;
        }

        ret = SetQHYCCDStreamMode(camhandle,0);//Set to single frame mode
        if(ret == QHYCCD_ERROR)
        {
            printf("Set stream mode failed.\n");
            goto failure;
        }

        ret = InitQHYCCD(camhandle);//Initializing the camera
        if(ret == QHYCCD_ERROR)
        {
            printf("Init camera failed.\n");
            goto failure;
        }

        ret = IsQHYCCDCFWPlugged(camhandle);//Check the filter wheel connection status
        if(ret == QHYCCD_SUCCESS)
        {
            printf("CFW is plugged.\n");

            num = (int)GetQHYCCDParam(camhandle, CONTROL_SLOTSNUM);//Get the number of filter wheel holes
            if(num != 0)
            {
                ret = SetQHYCCDParam(camhandle, CONTROL_CFWPORT, 50.0);//Set the target hole position as the third hole
                if(ret == QHYCCD_SUCCESS)
                {
                    while(status != 50.0)//Loop to get the position and determine whether to turn to the target position
                    {
                        status = GetQHYCCDParam(camHandle, CONTROL_CFWPORT);//Get current location
                        sleep(500);//Delay 500 ms
                    }
                }
            }
        }
    }

    if(camhandle)
    {
        ret = CloseQHYCCD(camhandle);//Close the camera
```

```
            if(ret == QHYCCD_ERROR)
            {
                printf("Close QHYCCD failed.\n");
                goto failure;
            }

            ret = ReleaseQHYCCDResource();//Release camera resources
            if(ret == QHYCCD_ERROR)
            {
                printf("Rlease SDK resource failed.\n");
                goto failure;
            }
        }

        return 0;

failure:
        printf("some fatal error happened\n");
        return 1;
}
```

6.GPS

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
        int num = 0;
        qhyccd_handle *camhandle = NULL;
        int ret = QHYCCD_ERROR;
        char id[32];
        int found = 0;
        unsigned int w,h,bpp,channels;
        unsigned char *ImgData;
        double chipw,chiph,pixelw,pixelh;
        unsigned char gps[44] = { 0 };
        int seqNumber, tempNumber, width, height;
        int status, pps;
        int temp, deg, min, south, west;
        double fractMin, latitude, longitude;
        int start_sec, start_us, end_sec, end_us, now_sec, now_us;
```

```
double exposure;

//To initialize the SDK
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get the camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open the camera and get the device handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}

	//Set read mode
	ret = SetQHYCCDReadMode(camhandle, 0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set read mode failed.\n");
```

```
        goto failure;
}

//Set continuous mode
ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
        printf("Set stream mode failed.\n");
        goto failure;
}

//Initializing the camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
        printf("Init camera failed.\n",ret);
        goto failure;
}

//Obtain camera hardware information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
        printf("GetQHYCCDChipInfo success!\n");
        printf("CCD/CMOS chip information:\n");
        printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
        printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
        printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
        printf("GetQHYCCDChipInfo fail\n");
        goto failure;
}

//Set the BIN
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
        printf("Set BIN mode failed.\n");
        goto failure;
}

//Set resolution
```

```
ret = SetQHYCCDResolution(camhandle, 0, 0, w, h);
if(ret == QHYCCD_ERROR)
{
	printf("Set resolution failed.\n");
	goto failure;
}

//Set the Color
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
	printf("This is a color camera.\n");
	ret = SetQHYCCDDebayerOnOff(camhandle,false);
	if(ret == QHYCCD_ERROR)
	{
		printf(“Setup color failed.\n”);
		goto failure;
	}
}

//Set the Bits
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set bits failed.\n");
		goto failure;
	}
}

//Set the Gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gain failed.\n");
		goto failure;
	}
}
```

```
//Set the Offset
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set offset failed.\n");
		goto failure;
	}
}

//Set up the Traffic
ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
	if(ret == QHYCCD_ERROR)
	{
		printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
		goto failure;
	}
}

//Set the red balance
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBR,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set red balance failed.\n");
		goto failure;
	}
}

//Set up green balance
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBG);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_WBG,30.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set green balance failed.\n");
```

```
            goto failure;
        }
    }

    //Set the blue balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBB);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_WBB,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set blue balance failed.\n");
            goto failure;
        }
    }

    //Set the brightness
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_BRIGHTNESS);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_BRIGHTNESS,0.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set brightness failed.\n");
            goto failure;
        }
    }

    //Set the contrast
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_CONTRAST);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_CONTRAST,0.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set contrast failed.\n");
            goto failure;
        }
    }

    //Set the Gamma
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAMMA);
    if(ret == QHYCCD_SUCCESS)
    {
```

```
        ret = SetQHYCCDParam(camhandle,CONTROL_GAMMA,1.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set gamma failed.\n");
            goto failure;
        }
    }

    //Set the DDR
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set DDR failed.\n");
            goto failure;
        }
    }

    //Set exposure time
    ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 10000);
    if(ret == QHYCCD_ERROR)
    {
        printf("Set expose time failed.\n");
        goto failure;
    }

    //Obtain image data memory length, open up memory space
    uint32_t length = GetQHYCCDMemLength(camhandle);
    if(length > 0)
    {
        ImgData = (unsigned char *)malloc(length);
        memset(ImgData,0,length);
    }
    else
    {
        printf("Get the min memory space length failure \n");
        goto failure;
    }

    //Check whether the GPS function is supported
    ret = IsQHYCCDControlAvailable(camHandle, CAM_GPS);
    if(ret == QHYCCD_SUCCESS)
    {
```

```
        printf("The camera can use GPS function.\n");
    }

    //Enable THE GPS function
    ret = SetQHYCCDParam(camHandle, CAM_GPS, 1.0);
    if(ret == QHYCCD_SUCCESS)
    {
        printf("Turn GPS on successfully.\n");
    }

    //This setting needs to be adjusted according to the actual situation to make the PPS count close to 1000000. Only QHY174-GPS requires this setting
    ret = SetQHYCCDGPSVCOXFreq(camHandle, 2);
    if(ret == QHYCCD_SUCCESS)
    {
        printf("Set GPS VCOX frequency successfully.\n");
    }

    //Set position A and position B of the calibration pulse according to the actual situation. This parameter is required only for QHY174-GPS
    SetQHYCCDGPSPOSA(camHandle, 0, pos, 40);
    SetQHYCCDGPSPOSB(camHandle, 0, pos, 40);

    //Exposure
    ret = BeginQHYCCDLive(camhandle);
    if( ret == QHYCCD_ERROR )
    {
        printf("Start live expose failed.\n");
        goto failure;
    }

    int fame = 0;
    //Obtaining image data
    while(frame < 100)
    {
        ret = QHYCCD_ERROR;
        while(ret != QHYCCD_SUCCESS)
        {
            ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
        }

        printf("Get live frame succeessfully.\n");
        frame ++;
        //show the image,this function need you do in software
```

```
            memcpy(gps, ImgData, 44);
            //GPS Status
            now_flag = (gps[33] / 16 ) %4;
            //PPS count value
            pps = 256*256*gps[41] + 256*gps[42] + gps[43];
            //Frame number
            seqNumber = 256*256*256*gps[0] + 256*256*gps[1] + 256*gps[2] + gps[3];
            //Image width
            width = 256*gps[5] + gps[6];
            //Image height
            height = 256*gps[7] + gps[8];
            //Latitude
            temp = 256*256*256*gps[9] + 256*256*gps[10] + 256*gps[11] + gps[12];
            south = temp > 1000000000;
            deg = (temp % 1000000000) / 10000000;
            min = (temp % 10000000) / 100000;
            fractMin = (temp % 100000) / 100000.0;
            latitude = (deg + (min + fractMin) / 60.0) * (south==0 ? 1 : -1);
            //Longitude
            temp = 256*256*256*gps[13] + 256*256*gps[14] + 256*gps[15] + gps[16];
            west = temp > 1000000000;
            deg = (temp % 1000000000) / 1000000;
            min = (temp % 1000000) / 10000;
            fractMin = (temp % 10000) / 10000.0;
            longitude = (deg + (min + fractMin) / 60.0) * (west==0 ? 1 : -1);
            //Shutter start time
            start_sec = 256*256*256*gps[18] + 256*256*gps[19] + 256*gps[20] + gps[21];
            start_us = 256*256*gps[22] + 256*gps[23] + gps[24];
            //Shutter stop time
            end_sec = 256*256*256*gps[26] + 256*256*gps[27] + 256*gps[28] + gps[29];
            end_us = 256*256*gps[30] + 256*gps[31] + gps[32];
            //Current time
            now_sec = 256*256*256*gps[34] + 256*256*gps[35] + 256*gps[36] + gps[37];
            now_us = 256*256*gps[38] + 256*gps[39] + gps[40];
            //Expose time
            exposure = (end_sec - start_sec) * 1000 * 1000 + (end_us - start_us);
        }

        delete(ImgData);
    }
    else
    {
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
    }

    if(camhandle)
    {
        //Stop capture
        ret = StopQHYCCDLive(camhandle);
```

```
            if(ret == QHYCCD_ERROR)
            {
                printf("Stop live capture failed.\n");
                goto failure;
            }

            //Close camera
            ret = CloseQHYCCD(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("Close camera failed.\n");
                goto failure;
            }
        }

        //Release SDK resource
        ret = ReleaseQHYCCDResource();
        if(ret == QHYCCD_ERROR)
        {
            printf("Rlease SDK resource failed.\n");
            goto failure;
        }

        return 0;

failure:
        printf("Some fatal error happened.\n");
        return 1;
}
```

7.AntiRBI function

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
        int num = 0;
        qhyccd_handle *camhandle = NULL;
        int ret = QHYCCD_ERROR;
        char id[32];
```

```
int found = 0;
unsigned int w,h,bpp,channels;
unsigned char *ImgData;
double chipw,chiph,pixelw,pixelh;

//To initialize the SDK
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get the camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open the camera and get the device handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}

	//Set read mode
```

```
ret = SetQHYCCDReadMode(camhandle, 0);
if(ret == QHYCCD_ERROR)
{
	printf("Set read mode failed.\n");
	goto failure;
}

//Set continuous mode
ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
	printf("Set stream mode failed.\n");
	goto failure;
}

//Initializing the camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("Init camera failed.\n",ret);
	goto failure;
}

//Obtain camera hardware information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo success!\n");
	printf("CCD/CMOS chip information:\n");
	printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
	printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
	printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
	printf("GetQHYCCDChipInfo fail\n");
	goto failure;
}

//Set the BIN
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
	printf("Set BIN mode failed.\n");
```

```
        goto failure;
}

//Set resolution
ret = SetQHYCCDResolution(camhandle, 0, 0, w, h);
if(ret == QHYCCD_ERROR)
{
        printf("Set resolution failed.\n");
        goto failure;
}

//Set the Color
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
        printf("This is a color camera.\n");
        ret = SetQHYCCDDebayerOnOff(camhandle,false);
        if(ret == QHYCCD_ERROR)
        {
                printf("Setup color failed.\n");
                goto failure;
        }
}

//Set the Bits
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
        ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
        if(ret == QHYCCD_ERROR)
        {
                printf("Set bits failed.\n");
                goto failure;
        }
}

//Set the Gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
        ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
        if(ret == QHYCCD_ERROR)
        {
                printf("Set gain failed.\n");
```

```
            goto failure;
        }
    }

    //Set the Offset
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set offset failed.\n");
            goto failure;
        }
    }

    //Set up the Traffic
    ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
        if(ret == QHYCCD_ERROR)
        {
            printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
            goto failure;
        }
    }

    //Set the red balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBR);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_WBR,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set red balance failed.\n");
            goto failure;
        }
    }

    //Set up green balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBG);
    if(ret == QHYCCD_SUCCESS)
    {
```

```
        ret = SetQHYCCDParam(camhandle,CONTROL_WBG,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set green balance failed.\n");
            goto failure;
        }
    }

    //Set the blue balance
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_WBB);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_WBB,30.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set blue balance failed.\n");
            goto failure;
        }
    }

    //Set the brightness
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_BRIGHTNESS);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_BRIGHTNESS,0.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set brightness failed.\n");
            goto failure;
        }
    }

    //Set the contrast
    ret = IsQHYCCDControlAvailable(camhandle,CONTROL_CONTRAST);
    if(ret == QHYCCD_SUCCESS)
    {
        ret = SetQHYCCDParam(camhandle,CONTROL_CONTRAST,0.0);
        if(ret == QHYCCD_ERROR)
        {
            printf("Set contrast failed.\n");
            goto failure;
        }
    }
```

```
//Set the Gamma
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAMMA);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAMMA,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gamma failed.\n");
		goto failure;
	}
}

//Set the DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Set exposure time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 10000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
}

//Enable the AntiRBI mode
ret = SetQHYCCDEnableLiveModeAntiRBI(camHandle, 1);
if(ret == QHYCCD_SUCCESS)
{
	printf("Enable live mode AntiRBI successfully.\n");
}

//Obtain image data memory length, open up memory space
uint32_t length = GetQHYCCDMemLength(camhandle);
if(length > 0)
{
	ImgData = (unsigned char *)malloc(length);
	memset(ImgData,0,length);
```

```
        }
        else
        {
            printf("Get the min memory space length failure \n");
            goto failure;
        }

        //Start expose
        ret = BeginQHYCCDLive(camhandle);
        if( ret == QHYCCD_ERROR )
        {
            printf("Start live expose failed.\n");
            goto failure;
        }

        int fame = 0;
        //Obtaining image data
        while(frame < 100)
        {
            ret = QHYCCD_ERROR;
            while(ret != QHYCCD_SUCCESS)
            {
                ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
            }

            printf("Get live frame succeessfully.\n");
            frame ++;
            //show the image,this function need you do in software
        }

        delete(ImgData);
}
else
{
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
}

if(camhandle)
{
        //End camera shooting
        ret = StopQHYCCDLive(camhandle);
        if(ret == QHYCCD_ERROR)
        {
```

```
            printf("Stop live capture failed.\n");
            goto failure;
        }

        //Close the camera
        ret = CloseQHYCCD(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Close camera failed.\n");
            goto failure;
        }
    }

    //Releasing SDK Resources
    ret = ReleaseQHYCCDResource();
    if(ret == QHYCCD_ERROR)
    {
        printf("Rlease SDK resource failed.\n");
        goto failure;
    }

    return 0;

failure:
    printf("Some fatal error happened.\n");
    return 1;
}
```

8.Burst function

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <qhyccd.h>
#include <sys/time.h>

int main(int argc,char *argv[])
{
    int num = 0;
    qhyccd_handle *camhandle;
    int ret;
    char id[32];
    int length;
```

```
int found = 0;
unsigned int w,h,bpp,channels;
unsigned int ex, ey, sizex, sizey;
unsigned char *ImgData;
double chipw,chiph,pixelw,pixelh;

unsigned int imagew;
unsigned int imageh;

ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("InitQHYCCDResource failed.\n");
	return;
}

num = ScanQHYCCD();
if(num <= 0)
{
	printf("ScanQHYCCD no camera be found.\n");
	return;
}

for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("GetQHYCCDId id = %s\n", id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("OpenQHYCCD failed.\n");
		return;
	}

	ret = SetQHYCCDReadMode(camhandle, 0);
```

```
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDReadMode failed.\n");
}

ret = SetQHYCCDStreamMode(camhandle, 1);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDStreamMode failed.\n");
}

ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("InitQHYCCD failed.\n");
}

ret = GetQHYCCDEffectiveArea(camhandle, &ex, &ey, &sizex, &sizey);
if(ret == QHYCCD_ERROR)
{
	printf("GetQHYCCDEffectiveArea failed.\n");
}

ret = GetQHYCCDChipInfo(camhandle, &chipw, &chiph, &imagew, &imageh, &pixelw, &pixelh, &bpp);
if (ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo:\n");
	printf("Chip    size width x height         : %.3f x %.3f [mm]\n", chipw, chiph);
	printf("Pixel size width x height          : %.3f x %.3f [um]\n", pixelw, pixelh);
	printf("Image size width x height          : %d x %d\n", imagew, imageh);
}
else
{
	printf("GetQHYCCDChipInfo failed.\n");
	return;
}

ret = SetQHYCCDBinMode(camhandle, 1, 1);
if(ret == QHYCCD_SUCCESS)
{
	printf("SetQHYCCDBinMode failed.\n");
}

ret = SetQHYCCDResolution(camhandle, 0, 0, imagew, imageh);
```

```
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDResolution failed.\n");
}

ret = SetQHYCCDDebayerOnOff(camhandle, false);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDDebayerOnOff failed.\n");
}

ret = SetQHYCCDParam(camhandle,CONTROL_TRANSFERBIT,16);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDParam BITS failed.\n");
}

ret = SetQHYCCDParam(camhandle,CONTROL_BRIGHTNESS,0);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDParam BRIGHTNESS failed.\n");
}

ret = SetQHYCCDParam(camhandle,CONTROL_CONTRAST,0);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDParam CONTRAST failed.\n");
}

ret = SetQHYCCDParam(camhandle,CONTROL_GAMMA,1.0);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDParam GAMMA failed.\n");
}

ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 0);
if(ret == QHYCCD_ERROR)
{
	printf("SetQHYCCDParam TRAFFIC failed.\n");
}

ret = SetQHYCCDParam(camhandle, CONTROL_DDR, 1.0);
if(ret == QHYCCD_ERROR)
{
```

```
        printf("SetQHYCCDParam DDR failed.\n");
    }

    ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 200000);
    if(ret == QHYCCD_ERROR)
    {
        printf("SetQHYCCDParam EXPOSURE failed.\n");
    }

    length = GetQHYCCDMemLength(camhandle);
    printf("GetQHYCCDMemLength length = %d\n", length);
    if(length > 0)
    {
        ImgData = (unsigned char *)malloc(length);
        memset(ImgData,0,length);
    }

    //begin live capture,it will start live thread in SDK
    ret = BeginQHYCCDLive(camhandle);
    printf("BeginQHYCCDLive ret = %d\n", ret);

    //Setup start and end frame,after setting this,camera will output middle frames when capture,for example setup 1 and 3,camera will output 2 frame
    ret = SetQHYCCDBurstModeStartEnd(camhandle, 1, 3);
    //Add patch data,it ca avoid data not enough that can't output image
    ret = SetQHYCCDBurstModePatchNumber(camhandle, 2000);

    //Enable Burst Mode
    ret = EnableQHYCCDBurstMode(camhandle, true);
    usleep(10000);

    //Start Burst Mode capture,it need add a delay between two functions
    ret = SetQHYCCDBurstIDLE(camhandle);
    usleep(20000);
    ret = ReleaseQHYCCDBurstIDLE(camhandle);

    //Get images
    ret = QHYCCD_ERROR;
    while(ret != QHYCCD_SUCCESS)
    {
        ret = GetQHYCCDLiveFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
        //printf("GetQHYCCDLiveFrame ret= %d\n", ret);
    }
    printf("w = %d h = %d bpp = %d channels = %d\n", w, h, bpp, channels);
```

```
            ret = StopQHYCCDLive(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("StopQHYCCDLive failed.\n");
            }
        }

        delete(ImgData);

        if(camhandle)
        {
            ret = CloseQHYCCD(camhandle);
            if(ret == QHYCCD_ERROR)
            {
                printf("CloseQHYCCD failed.\n");
            }
        }

        ret = ReleaseQHYCCDResource();
        if(ret == QHYCCD_ERROR)
        {
            printf("ReleaseQHYCCDResource failed.\n");
        }

        return 0;
}
```

9.Trigger function

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "qhyccd.h"

int main(int argc,char *argv[])
{
        int num = 0;
        qhyccd_handle *camhandle = NULL;
        int ret = QHYCCD_ERROR;
        char id[32];
        int found = 0;
        unsigned int w,h,bpp,channels;
```

```
unsigned char *ImgData;
double chipw,chiph,pixelw,pixelh;
double min, max, step;

//To initialize the SDK
ret = InitQHYCCDResource();
if(ret == QHYCCD_ERROR)
{
	printf("Init SDK failed!\n");
	goto failure;
}

//Scan camera
num = ScanQHYCCD();
if(num <= 0)
{
	printf("Not Found QHYCCD camera.\n\n");
	goto failure;
}

//Get the camera ID
for(int i = 0;i < num;i++)
{
	ret = GetQHYCCDId(i,id);
	if(ret == QHYCCD_SUCCESS)
	{
		printf("connected a camera,id is %s\n",id);
		found = 1;
		break;
	}
}

if(found == 1)
{
	//Open the camera and get the device handle
	camhandle = OpenQHYCCD(id);
	if(camhandle == NULL)
	{
		printf("Open QHYCCD fail \n");
		goto failure;
	}

	//Set read mode
	ret = SetQHYCCDReadMode(camhandle, 0);
	if(ret == QHYCCD_ERROR)
```

```
{
	printf("Set read mode failed.\n");
	goto failure;
}

//Set the single frame mode
ret = SetQHYCCDStreamMode(camhandle, 0);
if(ret == QHYCCD_ERROR)
{
	printf("Set stream mode failed.\n");
	goto failure;
}

//Initializing the camera
ret = InitQHYCCD(camhandle);
if(ret == QHYCCD_ERROR)
{
	printf("Init camera failed.\n",ret);
	goto failure;
}

//Obtain camera hardware information
ret = GetQHYCCDChipInfo(camhandle,&chipw,&chiph,&w,&h,&pixelw,&pixelh,&bpp);
if(ret == QHYCCD_SUCCESS)
{
	printf("GetQHYCCDChipInfo success!\n");
	printf("CCD/CMOS chip information:\n");
	printf("Chip width %3f mm,Chip height %3f mm\n",chipw,chiph);
	printf("Chip pixel width %3f um,Chip pixel height %3f um\n",pixelw,pixelh);
	printf("Chip Max Resolution is %d x %d,depth is %d\n",w,h,bpp);
}
else
{
	printf("GetQHYCCDChipInfo fail\n");
	goto failure;
}

//Set the BIN
ret = SetQHYCCDBinMode(camhandle, cambinx, cambiny);
if(ret == QHYCCD_ERROR)
{
	printf("Set BIN mode failed.\n");
	goto failure;
}
```

```
//Set resolution
ret = SetQHYCCDResolution(camhandle, 0, 0, w, h);
if(ret == QHYCCD_ERROR)
{
	printf("Set resolution failed.\n");
	goto failure;
}

//Set the Color
ret = IsQHYCCDControlAvailable(camhandle,CAM_IS_COLOR);
if(ret == QHYCCD_SUCCESS)
{
	printf("This is a color camera.\n");
	ret = SetQHYCCDDebayerOnOff(camhandle,false);
	if(ret == QHYCCD_ERROR)
	{
		printf(“Setup color failed.\n”);
		goto failure;
	}
}

//Set the Bits
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_TRANSFERBIT);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_TRANSFERBIT, 16);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set bits failed.\n");
		goto failure;
	}
}

//Set up the Traffic
ret = IsQHYCCDControlAvailable(camhandle, CONTROL_USBTRAFFIC);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle, CONTROL_USBTRAFFIC, 30);
	if(ret == QHYCCD_ERROR)
	{
		printf("SetQHYCCDParam CONTROL_USBTRAFFIC failed.\n");
		goto failure;
	}
```

```
}

//Set the Gain
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_GAIN);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_GAIN,30);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set gain failed.\n");
		goto failure;
	}
}

//Set the Offset
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_OFFSET);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_OFFSET,10);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set offset failed.\n");
		goto failure;
	}
}

//Set the DDR
ret = IsQHYCCDControlAvailable(camhandle,CONTROL_DDR);
if(ret == QHYCCD_SUCCESS)
{
	ret = SetQHYCCDParam(camhandle,CONTROL_DDR,1.0);
	if(ret == QHYCCD_ERROR)
	{
		printf("Set DDR failed.\n");
		goto failure;
	}
}

//Set the exposure time
ret = SetQHYCCDParam(camhandle, CONTROL_EXPOSURE, 1000000);
if(ret == QHYCCD_ERROR)
{
	printf("Set expose time failed.\n");
	goto failure;
```

```
}

//Obtain image data memory length, open up memory space
uint32_t length = GetQHYCCDMemLength(camhandle);
if(length > 0)
{
	ImgData = (unsigned char *)malloc(length);
	memset(ImgData,0,length);
}
else
{
	printf("Get the min memory space length failure \n");
	goto failure;
}

//Check whether the camera supports triggering
retVal = IsQHYCCDControlAvailable(camHandle, CAM_TRIGER_INTERFACE);
iif(retVal == QHYCCD_SUCCESS)
{
	printf("This camera can use trigger function.\n");
}

//Gets the number and name of the trigger interface
uint32_t num = 0;
retVal = GetQHYCCDTrigerInterfaceNumber(camhandle, &num);
if(num > 1)
{
	char name[40] = { 0 };
	for(int i = 0; i < num; i ++)
	{
		retVal = GetQHYCCDTrigerInterName(camhandle, i, name);
		if(retVal == QHYCCD_SUCCESS)
		{
			printf("Get triger interface successfully.\n");
		}
	}
}

//Set the default trigger interface to 0
retVal = SetQHYCCDTrigerInterface(camhandle, 0);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Set interface successfully.\n");
}

//The trigger function was enabled
retVal = SetQHYCCDTrigerFunction(camHandle, true);
if(retVal == QHYCCD_SUCCESS)
{
	printf("Enable trigger mode successfully.\n");
}
```

```
        //Enable the trigger output function
        retVal = EnableQHYCCDTrigerOut(camhandle);
        if(retVal == QHYCCD_SUCCESS)
        {
            printf("Enable trigger out successfully.\n");
        }

        //Start expose
        ret = ExpQHYCCDSingleFrame(camhandle);
        if( ret == QHYCCD_ERROR )
        {
            printf("Start single expose failed.\n");
            goto failure;
        }

        //Now you need send a trigger signal to camera,this signal can generated by software or hardware

        //Obtaining image data
        ret = GetQHYCCDSingleFrame(camhandle,&w,&h,&bpp,&channels,ImgData);
        if(ret == QHYCCD_SUCCESS)
        {
            printf("Get single frame succeessfully.\n");
            //show the image,this function need you do in software
        }
        else
        {
            printf("Get single frame failed.\n",ret);
        }

        delete(ImgData);
}
else
{
        printf("The camera is not QHYCCD camera or other error. \n");
        goto failure;
}

if(camhandle)
{
        //End camera shooting
        ret = CancelQHYCCDExposingAndReadout(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Cancel exposing and readout failed.\n");
            goto failure;
```

```
        }

        //Close the camera
        ret = CloseQHYCCD(camhandle);
        if(ret == QHYCCD_ERROR)
        {
            printf("Close camera failed.\n");
            goto failure;
        }
    }

    //Releasing SDK Resources
    ret = ReleaseQHYCCDResource();
    if(ret == QHYCCD_ERROR)
    {
        printf("Rlease SDK resource failed.\n");
        goto failure;
    }

    return 0;

failure:
    printf("Some fatal error happened.\n");
    return 1;
}
```

V. Image data structure

When the image data is obtained by GetQHYCCDSingleFrame and GetQHYCCDLiveFrame functions, the SDK will read the pixel data of the 2d image in order and store it in the one-dimensional array. The order of reading pixel is Z-shaped, that is, from left to right and from top to bottom.

According to the different image data format, the data structure of each pixel will also change. RAW8 image data stores one pixel of data for an eight-bit variable;RAW16 image data stores one pixel of data for two 8-bit brightening. At this time, the high and low bits of data need to be considered. The low bits are first and the high bits are second.RGB24 image data stores one pixel of data for three eight-bit data.