Volume

1

QUANTUM SCIENTIFIC IMAGING

500, 600 and RS Series CCD Imaging Cameras

COM API Reference v7.2

WINDOWS® OPERATING SYSTEMS

API Reference Manual

© Quantum Scientific Imaging, Inc.
12 Coteau Dr.
Poplarville, MS 39470
Phone 888.774.4223 • Fax 888.774.4223

Chapter

1

Introduction

Using the QSI COM API to control your QSI camera

The QSI COM API provides a programming interface on Windows® platforms to capture images and control QSI cameras. Any COM compliant programming language can be used. These languages include Microsoft VB.net, VBA, VBScript, C# .net, C++.

The QSI API is compliant with the ASCOM interface definitions for camera and filter wheels. The API includes methods and properties beyond the ASCOM standard to fully expose the capabilities of the QSI camera.

The QSI API COM library exposes the ICameraEx, ICamera, IFilterWheelEx and IFilterWheel interfaces. Each interface is composed of method calls and properties. A method call is made by the application to perform an action, such as start an exposure. A property is information about the camera that can be retrieved or set to a new value. An example of a property is the temperature set point of the ccd cooler, SetCCDTemperature. When this value is read by the program, the API returns the current setting of the cooler. When this value is written to, a new cooler set point is assigned.

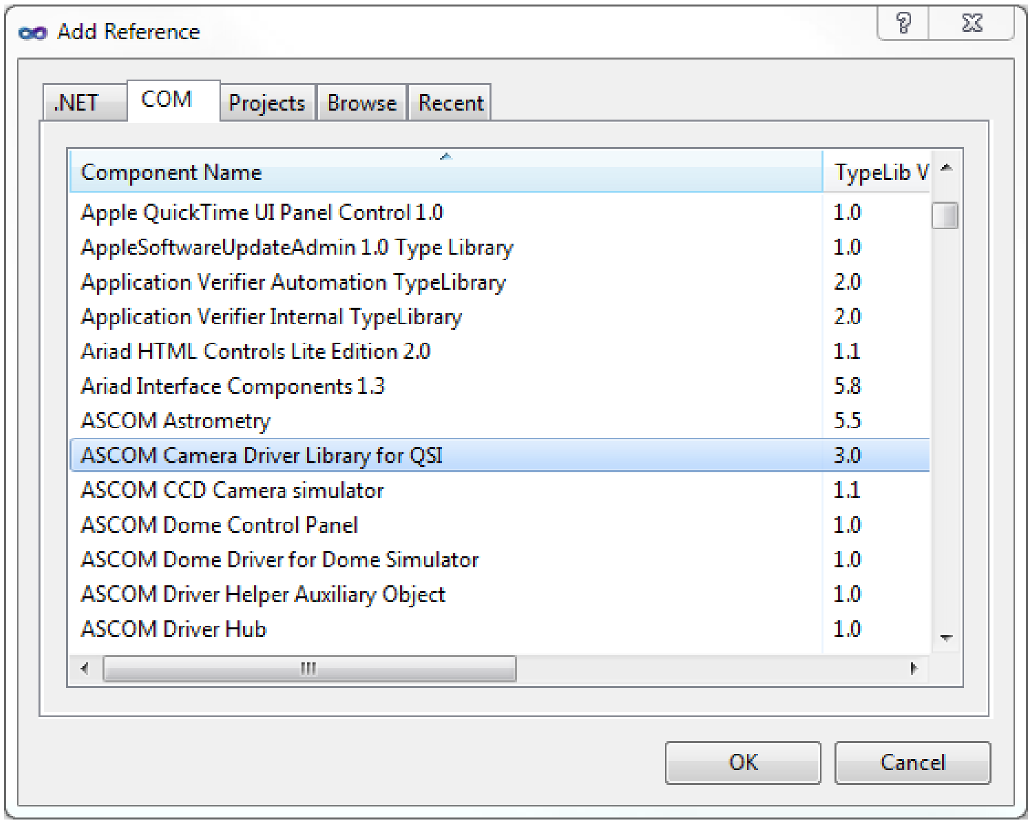
The COM server that implements the API is included on the standard QSI installation disc in the form of a Windows® dll and is automatically installed and registered when the QSI device drivers are installed on the target system. The QSI API COM object CCDCamera is implemented in the QSICamera.dll file. The dll is located in the C:\Windows\System32 directory on 32 bit platforms, or the C:\Windows\System32 directory on 64 bit platforms for 64 bit applications and the C:\Windows\SysWow64 directory on 64 bit platforms for 32 bit applications.

How to Use the QSI Camera COM object

You must first install the QSI Camera software and drivers using the QSI Installation CD-ROM that came with your camera. The QSI installer is also available for download at: http//www.qsimaging.com/support/downloads.html.

The ProgID is the standard way a program locates a COM object using the registry on Windows® systems. The ProgID for the QSI API is QSICamera.CCDCamera.

When using Visual Studio and .net languages, a reference to the "ASCOM Camera Driver for QSI" COM component should be added to your project and you should include the QSICameraLib in your source:



Component Name	TypeLib V
Apple QuickTime UI Panel Control 1.0	1.0
AppleSoftwareUpdateAdmin 1.0 Type Library	1.0
Application Verifier Automation TypeLibrary	2.0
Application Verifier Internal TypeLibrary	2.0
Ariad HTML Controls Lite Edition 2.0	1.1
Ariad Interface Components 1.3	5.8
ASCOM Astrometry	5.5
ASCOM Camera Driver Library for QSI	3.0
ASCOM CCD Camera simulator	1.1
ASCOM Dome Control Panel	1.0
ASCOM Dome Driver for Dome Simulator	1.0
ASCOM Driver Helper Auxiliary Object	1.0
ASCOM Driver Hub	1.0

VB:

```
Imports QSICameraLib
```

C#

```
using QSICameraLib;
```

You must also create an instance of the QSI CCDCamera class for each camera you wish to control:

VB:

```
Private cam As QSICameraLib.CCDCameraClass
cam = New CCDCameraClass()
```

C#

```
QSICameraLib.CCDCameraClass cam;
cam = new CCDCameraClass();
```

You are now ready to start implementing you camera control program.

CCD Camera Supporting Information

CCD Imager Geometry

The CCD imager geometry is organized in rows and columns. The number of columns is the number of pixels in the image in the X (horizontal) direction, starting from the left of the image. The number of rows is the number of pixels in the Y (vertical) direction, starting at the top. The image array is returned as single dimension array. The array is ordered by column then row. The array begins with the pixels of the first row, beginning with the first column pixel and ending in the first row, last column pixel. Row two follows in the same order, column zero to n, followed by rows three, four, etc. until the last row is returned.

The API provides the total number of columns as the property CameraXSize and the total number of rows as the property CameraYSize. These properties are valid only when the camera is in the connected state.

The frame to be captured is defined by four properties, StartX, StartY, which define the upper left corner of the frame, and NumX and NumY the define the binned size of the frame.

Pixel binning combines CCD pixels into groups with each group representing one image pixel.

Restrictions on binning are:

The properties BinX and BinY specify the number of bits per bin for each axis. If CanAsymmetricBin is False, BinX must equal BinY.

MaxXBin and MaxYBin specify the maximum number of bits per bin allowed by the camera for each axis. Therefore, BinX <=MaxXBin and BinY <= MaxYBin.

The total number of bits in a image frame must not exceed the CCD dimension. Therefore, (StartX + NumX) * BinX <= CameraXSize and (StartY + NumY) * BinY <= CameraYSize

Shutterless Cameras

The second parameter in StartExposure(Duration, Light) determines the shutter state during exposure. If the Light argument is True, the shutter is opened during the exposure period. If the property HasShutter is False, the Light parameter is ignored and the shutter is always in the open state.

Camera with Filter Wheels

Some QSI camera models contain an integral filter wheel. The property HasFilterWheel will return True if the camera contains a internal filter wheel. This property will return False if the camera has no internal wheel and is not affected by the existence of an external filter wheel. If you have an external filter wheel, use the ASCOM driver provided by the filter wheel manufacturer to control the wheel.

Image copy efficiency

The API provides three properties to retrieve image data: ImageArray, ImageArrayVariant, ImageArrayLong, and ImageArrayDouble.

ImageArrayLong returns a SAFEARRAY of long values. This is the most efficient and preferred way to return the image data. It avoids the large per value memory overhead of a Variant type. This method is not part of the ASCOM specification; ASCOM users should use ImageArray or ImageArrayVariant.

ImageArray returns a Variant with a vartype of SAFEARRAY long as per the ASCOM v5 specification.

Unfortunately, the SAFEARRAY long data type is not available to simple scripting languages, such as VBScript. In this case, use ImageArrayVariant, which returns a VARIANT of varitype SAFEARRAY of Variants.

ASCOM notes

The ASCOM interface specification states that if a property or method is not supported or the object encounters an error in execution, the API will throw an exception. This is required because there is no provision for a property or method to return an error code. The calling application should wrap calls to the properties/methods that can throw an exception with a try/catch block.

For example:

VB:

```
Dim eGain as Double
Try
  eGain = cam.ElectronsPerADU
Catch [error] As Exception
  Console.WriteLine([error].ToString())
End Try
```

C#:

```
double eGain;
try
{
  eGain = cam.ElectronsPerADU;
}
catch (Exception error)
{
  Console.WriteLine(error.ToString());
}
```

ASCOM specifies a separate COM interface for the Filter Wheel. The QSICamera.CCDCamera class implements both the ICamera and IFilterWheel interfaces. The IFilterWheel interface is only usable when the camera contains an internal filter wheel. The ICameraEx property HasFilterWheel will return true if the connected camera has an internal filter wheel.

The IFilterWheel properties and methods have been included in the ICameraEx interface for convenience. However, there is one property name and one method name that are defined in both interfaces: the property Connected and the method SetupDialog(). Fortunately, the Connected property and the SetupDialog method perform the same function for both ICameraEx and IFilterWheel, as the filter wheel is contained in the camera. If you wish to explicitly access the filter wheel property and method there are two ways to accomplish this.

(1) The IFilterWheel property Connected is exposed in ICameraEx as FilterWheelConnected, and the IFilterWheel method SetupDialog() is exposed in ICameraEx as FilterWheelSetupDialog().

(2) You can preface the propery and method with the interface name. i.e. IFilterWheel_Connected and IFilterWheel_SetupDialog().

In either case the net effect internally in the API is the IFilterWheel Connected property delegates the call to the camera Connected property and the IFilterWheel SetupDialog() method delegates to the camera SetupDialog() method. There is no need to use the filter wheel variants. They are included strictly for compatibility with the ASCOM specification.

Upgrading to release 5.3.0 and beyond.

With Release 5.3.0, the location of the QSIAPI library dll has been moved from the Program Files\QSI directory to the Windows\System32 directory on 32 bit version of Windows®. On 64 bits versions of Windows®, there are two dlls, one for 32 bit processes and one for 64 bit processes. Both dlls are named "QSICamera.dll" and are located in the Windows\SysWOW64 and Windows\System32 directories respectively.

The QSI setup program will install and properly register both versions of the library. If you wish to manually register the dlls on a 64 bits system, you must use the regsvr32.exe program located in Windows\SysWOW64 to register the 32 bit versions and use the regsvr32.exe program located in Windows\System32 to register the 64 bit version.

Early Binding Notes

Starting with QSI API Release 4.5.0 there are two new interfaces, ICameraEx and IFilterWheelEx, which inherit from ICamera and IFilterWheel. All QSI specific properties and methods (those not specified by the ASCOM specification) have been moved to the "Ex" version of each interface. Also, the ICamera and IFilterWheel interfaces have new GUIDs assigned and have been re-ordered to conform to the ASCOM V5 early binding specification.

As a result, code that was compiled with versions earlier than 4.5.0 must be recompiled after API version 4.5.0 has been installed. Any interface pointers that reference QSI specific properties and methods must be changed to the "Ex" type (i.e. ICamera ptr; changes to ICameraEx ptr;). ICameraEx inherits from ICamera so that an interface pointer to ICameraEx has access to all of the ICamera methods also. The same relationship holds for IFilterWheelEx.

Upgrading from prior QSI API releases to release 4.5.0.

With the introduction of release 4.5.0, the typlib for the API has changed from version 2.0 to version 3.0 to accommodate the new interfaces and ASCOM dictated changes to the argument type of ImageArray and ImageArrayVariant. You must recompile any clients that were compiled for an earlier version of the API. For Microsoft Visual Studio® users, a change in the typlib version requires that you re-establish the reference in your project to the new API version.

To re-establish the reference, first install the QSI API and drivers. Start Visual Studio and open the client project. In the solution explorer window, References section, delete the old reference to "QSICameraLib". Now right click on "References" and select "Add Reference…". Select the COM tab. Scroll down to find ASCOM Camera Driver Library for QSI Version 3.0. Select it and click OK. Now recompile the client.

Simple C++ Example

Here is a very simple example on how to use the QSI API with MS Visual Studio® 2010 C++

```
// QSICPP.cpp
//

#include "stdafx.h"
#include "stdlib.h"
#include "resource.h"

#import "progid:QSICamera.CCDCamera"

using namespace std;
using namespace QSICameraLib;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
        int nRetCode = 0;
        char temp[256];
        char szInfo[1024];
        QSICameraLib::ICameraEx *     pCam;

        cout << "Hello World\n";

        HRESULT h = ::CoInitialize(NULL);

        h = ::CoCreateInstance( __uuidof (QSICameraLib::CCDCamera), NULL,
        CLSCTX_INPROC_SERVER, __uuidof( QSICameraLib::ICamera ) , (void **)
        &pCam);

        if (h != S_OK)
        {
                cout << "\nCoCreateInstance Failed.\n";
                cin >> temp;    //wait for input
                exit(0);
        }

        try
        {
                pCam->Connected = true;
        }
        catch (_com_error & e)
        {
                cout << "\nConnection Failed\n";
                cin >> temp;    //wait for input
                exit(0);
        }

        try
        {
                sprintf_s(szInfo, 256, "Temp: %5.1fC Cooler Power: %3.0f%%\n",
                pCam->CCDTemperature, pCam->CoolerPower);
        }
        catch (_com_error & e)
        {
                szInfo[0] = 0;
        }

        std::cout << szInfo;


        try
        {
                pCam->StartExposure(0.1, true);
                while (!pCam->ImageReady)
```

```
{
        Sleep(100);
}

SAFEARRAY *sa = pCam->ImageArrayLong;
int dims = SafeArrayGetDim(sa);

//Get Upper and Lower Bound for each dimensions
long lBound,uBound;
int dim_size[3];

for(int i = 1; i <= dims; i++)
{
        if(SafeArrayGetLBound(sa ,i ,&lBound) < 0 ||
        SafeArrayGetUBound(sa ,i ,&uBound) < 0)
                throw "Can't get bounds";
        dim_size[i-1] = uBound - lBound + 1;
}

HRESULT lResult;
long HUGEP *arr;

lResult = SafeArrayAccessData(sa,(void HUGEP**)&arr);
if( lResult != S_OK)
        throw "Can't lock safearray";

//dim_size[0] = image rows
//dim_size[1] = image cols

long ** image;
image = new long* [ dim_size[0] ];
for( int i = 0; i < dim_size[0]; i++)
{
        image[i] = new long[dim_size[1]];
}

//Convert image from row first to column first
for(int i = 0; i < dim_size[0]; i++)
{
        int j = i;
        for (int k = 0; k < dim_size[1]; k++)
        {
                image[i][k] = arr[j];
                j += dim_size[0];
        }
}

//display 1st row of image, first 5 cols
for (int i = 0; i < 5; i++)
{
        std::cout << image[0][i] << "\n";
}

// TODO do your processing on the image here
// Example, add 5 to each pixel

for(int i = 0 ;i < dim_size[0]; i++)         //rows
{
        for(int j = 0; j < dim_size[1]; j++)  //columns
        {
                image[i][j] = image[i][j]+5;
        }
}

//Unlock SafeArray
lResult = SafeArrayUnaccessData(sa);
if (lResult != S_OK)
```

```
                        throw "Can't unlock safearry";

                SafeArrayDestroy(sa);

                for(register int i = 0; i < dim_size[0]; i++)
                        delete [] image[i];
                delete [] image;

                cout << "\nImage complete\n";
        }
        catch (_com_error & e)
        {
                cout << "\nImage exception\n";
                cin >> temp;    //wait for input
                exit(0);
        }

        try
        {
                pCam->Connected = false;
        }
        catch (_com_error & e) {}

        try
        {
                pCam->Release();
        }
        catch (_com_error & e) {}

        cin >> temp;    //wait for input

        return 0;
}
```

Chapter

2

API Reference

QSICameraLib.CCDCamera properties and methods

Properties

ICamera

BinX
BinY
CameraState
CameraXSize
CameraYSize
CanAbortExposure
CanAsymmetricBin
CanGetCoolerPower
CanPulseGuide
CanSetCCDTemperature
CanStopExposure
CCDTemperature
Connected
CoolerOn
CoolerPower
Description
DriverInfo
ElectronsPerADU
FilterOffsets
FullWellCapacity
HasShutter
HeatSinkTemperature
ImageArray
ImageArrayVariant
ImageReady
IsPulseGuiding
LastError
LastExposureDuration
LastExposureStartTime
MaxADU
MaxBinX
MaxBinY
NumX
NumY
PixelSizeX
PixelSizeY
SerialNumber
SetCCDTemperature
ShutterMode
StartX
StartY

ICameraEx

AntiBlooming
CameraGain
CanSetGain
FanMode
FilterPostionTrim
FlushCycles
HasFilterWheel
HasFilterWheelTrim
HostTimedExposure
ImageArrayLong
IsMainCamera
LEDEnabled
ManualShutterMode
ManualShutterOpen
MaskPixels
MaxExposureTime
MinExposureTime
ModelNumber
Name
Named Filter Wheels
PowerOfTwoBinning
PreExposureFlush
ReadoutSpeed
ShutterPriority
SoundEnabled

<u>IFilterWheel</u>

Connected
FocusOffset
Names
Position
SetupDialog

<u>IFilterWheelEx</u>

FilterWheelConnected
FilterWheelSetupDialog

Methods

<u>ICamera</u>

AbortExposure
PulseGuide
SetupDialog
StartExposure
StopExposure

<u>ICameraEx</u>

GetPixelMask
SetPixelMask
External Trigger Input

ICamera Properties

AntiBlooming

Property

CCDCamera.AntiBlooming (enum AntiBloomingEnum)

Syntax

CCDCamera.AntiBlooming = AntiBloomingHigh;

Exceptions

Throws an exception if the AntiBlooming property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
   cam.AntiBlooming = AntiBloomingEnum.AntiBloomHigh
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.AntiBlooming = AntiBloomingEnum.AntiBloomHigh;
}
catch (Exception error)
{}
```

Remarks

Controls the amount of anti-blooming in anti-blooming cameras.

Symbolic Constants

The symbolic values for AntiBloomingEnum are:

Constant	Value
AntiBloomNormal	0
AntiBloomHigh	1

BinX

Property

CCDCamera.BinX (Short)

Syntax

CCDCamera.BinX [= Short]

Exceptions

Throws an exception for illegal binning values

Examples:

VB:

```
Dim binX as Short
binX = cam.BinX
cam.BinX = 2
```

C#:

```
short binX;
binX = cam.BinX;
cam.BinX = 2;
```

Remarks

Sets the binning factor for the X axis. Also returns the current value. Defaults to 1 when the camera link is established. Note: the driver does not check for compatible sub-frame values when this value is set; rather they are checked upon StartExposure.

BinY

Property

CCDCamera.BinY (Short)

Syntax

CCDCamera.BinY [= Short]

Exceptions

Throws an exception for illegal binning values

Examples

VB:

```
    Dim binY as Short
    binX = cam.BinY
    cam.BinY = 2
    cam.BinY = cam.BinX
```

C#:

```
    short binY;
    binY = cam.BinY;
    cam.BinY = 2;
    cam.BinY = cam.BinX;
```

Remarks

Sets the binning factor for the Y axis Also returns the current value. Defaults to 1 when the camera link is established. Note: The driver does not check for compatible sub-frame values when this value is set; rather they are checked upon StartExposure.

CameraGain

Property

CCDCamera.CameraGain (enum CameraGainEnum)

Syntax

CCDCamera.CameraGain = CameraGainHigh;

Exceptions

Throws an exception if the CameraGain property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
   cam.CameraGain = CameraGainEnum.CameraGainHigh
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.CameraGain = CameraGainEnum.CameraGainHigh;
}
catch (Exception error)
{}
```

Remarks

Controls the amount of camera gain in adjustable gain cameras. Sets the gain of the camera to maximize dynamic range. High gain is the default and provides the greatest sensitivity. Low gain is useful when binning an image where the binned pixels contain more electrons than a normal un-binned pixel. Sensitivity is lower, but the full dynamic range of the binned image can be captured. High = 0.75e-/ADU, Low = 1.5e-/ADU. When set to Auto, the camera will use high gain if x and y bin equals one, and low gain in all other cases.

Symbolic Constants

The symbolic values for CameraGainEnum are:

Constant	Value
CameraGainHigh	0
CameraGainLow	1
CameraGainAuto	2

CameraState

Property

CCDCamera.CameraState (read only, enumeration)

Syntax

CCDCamera.CameraState

Exceptions

Throws an exception if the camera status is unavailable.

Examples

VB:

```
If cam.CameraState = CameraStateEnum.CameraError Then
   Return
End If
```

C#:

```
if (cam.CameraState == CameraStateEnum.CameraError)
{
  return;
}
```

Remarks

Returns one of the following status information enumeration values:

Value	State	Meaning
0	CameraIdle	At idle state, available to start exposure
1	CameraWaiting	Exposure started but waiting (for shutter, trigger, filter wheel, etc.)
2	CameraExposing	Exposure currently in progress
3	CameraReading	CCD array is being read out (digitized)
4	CameraDownload	Downloading data to PC
5	CameraError	Camera error condition serious enough to prevent further operations (link fail, etc.).

CameraXSize

Property

CCDCamera.CameraXSize (read only, Long)

Syntax

CCDCamera.CameraXSize

Exceptions

Thows exception if the value is not known

Example

VB:

```
    Dim x as Long
    x = cam.CameraXSize
```

C#:

```
    long x;
    x = cam.CameraXSize;
```

Remarks

Returns the width of the CCD camera chip in un-binned pixels.

CameraYSize

Property

CCDCamera.CameraYSize (read only, Long)

Syntax

CCDCamera.CameraYSize

Exceptions

Throws exception if the value is not known

Examples:

VB:

```
    Dim y as Long
    y = cam.CameraYSize
```

C#:

```
    long y;
    y = cam.CameraYSize;
```

Remarks

Returns the height of the CCD camera chip in un-binned pixels.

CanAbortExposure

Syntax

CCDCamera.CanAbortExposure (read only, Boolean)

Syntax

CCDCamera.CanAbortExposure

Exceptions

None

Examples:

VB:

```
If cam.CanAbortExposure Then
   cam.AbortExposure()
End If
```

C#:

```
if (cam.CanAbortExposure)
   cam.AbortExposure();
```

Remarks

Returns True if the camera can abort exposures; False if not.

CanAsymmetricBin

Property

CCDCamera.CanAsymmetricBin (read only, Boolean)

Syntax

CCDCamera.CanAsymmetricBin

Exceptions

Throws an exception if the value is not known.

Examples:

VB:

```
Try
   If cam.AsymmetricBin Then
      cam.BinX = 1
      cam.BinY = 2
   End If
Catch [error] As Exception
End Try
```

C#:

```
try
{
   if (cam.AsymmetricBin)
   {
       cam.BinX = 1;
       cam.BinY = 2;
   }
}
catch (Exception error)
{
}
```

Remarks

If True, the camera can have different binning on the X and Y axes, as determined by BinX and BinY. If False, the binning must be equal on the X and Y axes.

CanGetCoolerPower

Property

CCDCamera.CanGetCoolerPowere (read only, Boolean)

Syntax

CCDCamera.CanGetCoolerPower

Exceptions

None

Examples:

VB:

```
If cam.CanGetCoolerPower Then

End If
```

C#:

```
if (cam.CanGetCoolerPower)
{

}
```

Remarks

If True, the camera's cooler set point can be adjusted. If False, the camera either uses open-loop cooling or does not have the ability to adjust temperature from software, and setting the TemperatureSetpoint property has no effect.

CanPulseGuide

Property

CCDCamera.CanPulseGuide (read only, Boolean)

Syntax

CCDCamera.CanPulseGuide

Exceptions

None

Examples:

VB:

```
If cam.CanPulseGuide Then
   cam.PulseGuide(GuideDirections.guideWest, 10)
End If
```

C#:

```
if (cam.CanPulseGuide)
{
   cam.PulseGuide(GuideDirections.guideWest, 10);
}
```

Remarks

Returns True if the camera can send auto-guider pulses to the telescope mount; False if not. (Note: this does not provide any indication of whether the auto-guider cable is actually connected.)

CanSetCCDTemperature

Property

CCDCamera.CanSetCCDTemperature (read only, Boolean)

Syntax

CCDCamera.CanSetCCDTemperature

Exceptions

None

Examples:

VB:

```
If cam.CanSetCCDTemperature Then
   cam.SetCCDTemperature(-10.7)
End If
```

C#:

```
if (cam.CanSetCCDTemperature)
{
   cam.SetCCDTemperature(-10.7);
}
```

Remarks

If True, the camera's cooler set point can be adjusted. If False, the camera either uses open-loop cooling or does not have the ability to adjust temperature from software, and setting the TemperatureSetpoint property has no effect.

CanSetGain

Syntax

CCDCamera.CanSetGain (read only, Boolean)

Syntax

CCDCamera.CanSetGain

Exceptions

None

Examples:

VB:

```
    If cam.CanSetGain Then
       cam.CameraGain = CameraGainEnum.CameraGainHigh
    End If
```

C#:

```
    if (cam.CanSetGain)
       cam.CameraGain = CameraGainEnum.CameraGainHigh;
```

Remarks

Returns True if the camera has adjustable gain settings; False if not.

CanStopExposure

Syntax

CCDCamera.CanStopExposure (read only, Boolean)

Syntax

CCDCamera.CanStopExposure

Exceptions

Throws an exception if not supported. Throws an exception if an error condition such as link failure is present.

Examples:

VB:

```
If cam.CanStopExposure Then
   cam.StopExposure()
End If
```

C#:

```
if (cam.CanStopExposure)
{
   cam.StopExposure();
}
```

Remarks

Some cameras support StopExposure, which allows the exposure to be terminated before the exposure timer completes, but will still read out the image. Returns True if StopExposure is available, False if not.

CCDTemperature

Property

CCDCamera.CCDTemperature (read only, Double)

Syntax

CCDCamera.CCDTemperature

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
    Dim temp as Double
    temp = cam.CCDTemerature
```

C#:

```
    double temp;
    temp = cam.CCDTemperature;
```

Remarks

Returns the current CCD temperature in degrees Celsius. Only valid if CanControlTemperature is True.

Connected

Property

CCDCamera.Connected (Boolean)

Syntax

CCDCamera.Connected [= Boolean]

Exceptions

Throws an exception if unsuccessful.

Examples:

VB:

```
if cam.Connected = False Then
   cam.Connected = True
EndIf
```

C#:

```
if (cam.Connected == false)
   cam.Connected = true;
```

Remarks

Controls the link between the driver and the camera. Set True to enable the link. Set False to disable the link (this does not switch off the cooler). You can also read the property to check whether it is connected. The camera must be connected before using properties and methods on the camera object that relate to camera capabilities.

CoolerOn

Property

CCDCamera.CoolerOn (Boolean)

Syntax

CCDCamera.CoolerOn [= Boolean]

Exceptions

Throws an exception if not supported. Throws an exception if an error condition such as link failure is present

Examples:

VB:

```
If cam.CoolerOn = False Then
   cam.CoolerOn = True
End If
```

C#:

```
if (cam.CoolerOn == false)
   cam.CoolerOn = true;
```

Remarks

Turns on and off the camera cooler, and returns the current on/off state. Warning: turning the cooler off when the cooler is operating at high delta-T (typically >20C below ambient) may result in thermal shock. Repeated thermal shock may lead to damage to the sensor or cooler stack.

CoolerPower

Property

CCDCamera.CoolerPower (read only, Double)

Syntax

CCDCamera.CoolerPower

Exceptions

Throws an exception if not supported by the camera. Throws an exception if an error condition such as link failure is present.

Examples:

VB:

```
Dim power as Double
If cam.CoolerOn = True Then
   power = cam.CoolerPower;
End If
```

C#:

```
double power;
if (cam.CoolerOn == true)
   power = cam.CoolerPower;
```

Remarks

Returns the present cooler power level, in percent. Returns zero if CoolerOn is False.

Description

Property

CCDCamera.Description (read only, String)

Syntax

CCDCamera.Description

Exceptions

Throws an exception if description unavailable

Examples:

VB:

```
Try
   Dim desc As String
   desc = cam.Description
Catch [error] As Exception
   ' No Description available from this device
End Try
```

C#:

```
try
{
   string desc;
   desc = cam.Description;
}
Catch (Exception error)
{
   // No Description available from this device
}
```

Remarks

Returns a description of the camera model, such as manufacturer and model number. The string shall not exceed 68 characters (for compatibility with FITS headers).

DriverInfo

Property

CCDCamera.DriverInfo (read only, String)

Syntax

CCDCamera.DriverInfo

Exceptions

Throws an exception if description unavailable

Examples:

VB:

```
Try
   Dim info As String
   info = cam.DriverInfo
Catch [error] As Exception

End Try
```

C#:

```
try
{
   string info;
   info = cam.DriverInfo;
}
Catch (Exception error)
{

}
```

Remarks

Returns revision information of the loaded driver. The string shall not exceed 68 characters (for compatibility with FITS headers).

ElectronsPerADU

Property

CCDCamera.ElectronsPerADU (read only, Double)

Syntax

CCDCamera.ElectronsPerADU

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Try
   Dim epa As Double
    epa = cam.ElectronsPerADU
Catch [error] As Exception
   ' ElectronsPerADU not available from this device
End Try
```

C#:

```
try
{
   double epa;
   epa = cam.ElectroncsPerADU;
}
catch (Exception error)
{
   // ElectronsPerADU not available from this device
}
```

Remarks

Returns the gain of the camera in photoelectrons per A/D unit.

EnableShutterStatusOutput

Property

QSICamera.EnableShutterStatusOutput (Boolean)

Syntax

CCDCamera.EnableShutterStatusOutput = true;

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
enabled = cam.EnableShutterStatusOutput
if (enabled == false)
   cam.EnableShutterStatusOutput = true;
```

Remarks

The guider port can be configured to provide a shutter open/close indication using one of the guider port outputs. To enable this feature the property EnableShutterStatusOutput should be set to true. In this mode, the camera will use the the "up" output (pin 2) to reflect the shutter state. The camera will pull pin 2 to the common pin 5 while the shutter is open.

The guider ports are opto isolated open collector outputs. Each output is capable of sinking 50ma, 50 VDC maximum. The common pin must be at ground potential and the "up" output must be pulled up by an external resistor to V+.

This signal is meaningful only when the mechanical shutter is in start/stop mode (an exposure time of greater than 300 milliseconds).

To disable this feature and re-enable guiding output, set the EnableShutterStatusOutput property to false. The EnableShutterStatusOutput defaults to false on camera power up and is not saved between camera connections or power off.

FanMode

Property

CCDCamera.FanMode (enum FanModeEnum)

Syntax

CCDCamera.FanMode = FanModeEnum.FanQuiet;

Exceptions

Throws an exception if the FanMode property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
   cam.FanMode = FanModeEnum.FanFull
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.FanMode = FanModeEnum.FanFull;
}
catch (Exception error)
{

}
```

Remarks

Controls the speed of the camera's cooling fans.

Symbolic Constants

The symbolic values for FanModeEnum are:

```
Constant     Value   Description
FanOff        0      Fans off
FanQuiet      1      Fans slow speed, full if cooling requires.
FanFull       2      Fans full speed.
```

FlushCycles

No longer supported. See PreExpsoureFlush.

FullWellCapacity

Property

CCDCamera.FullWellCapacity (read only, Double)

Syntax

CCDCamera.FullWellCapacity

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Try
   Dim fwc As Double
   fwc = cam.FullWellCapacity
Catch [error] As Exception
   ' FullWellCapacity not available from this device
End Try
```

C#:

```
try
{
   double fwc;
   fwc = cam.FullWellCapacity;
}
catch (Exception error)
{
   // FullWellCapacity not available from this device
}
```

Remarks

Reports the full well capacity of the camera in electrons, at the current camera settings (binning, SetupDialog settings, etc.)

HasFilterWheel

Property

CCDCamera.HasFilterWheel (read only, Boolean)

Syntax

CCDCamera.HasFilterWheel

Exceptions

Throws an exception if no camera is connected.

Examples:

VB:

```
    If cam.HasFilterWheel Then

    End If
```

C#:

```
    if (cam.HasFilterWheel)
    {

    }
```

Remarks

If True, the camera has an internal filter wheel. If False, the camera does not have an internal filter wheel.

HasShutter

Property

CCDCamera.HasShutter (read only, Boolean)

Syntax

CCDCamera.HasShutter

Exceptions

Throws an exception if no camera is connected.

Examples:

VB:

```
If cam.HasShutter Then

End If
```

C#:

```
if (cam.HasShutter)
{

}
```

Remarks

If True, the camera has a mechanical shutter. If False, the camera does not have a shutter. If there is no shutter, the StartExposure command will ignore the Light parameter.

HeatSinkTemperature

Property

CCDCamera.HeatSinkTemperature (read only, Double)

Syntax

CCDCamera.HeatSinkTemperature

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
    Dim temp as Double
    temp = cam.CCDTemerature
```

C#:

```
    double temp;
    temp = cam.CCDTemperature
```

Remarks

Returns the current heat sink temperature (The ambient air temperature as it enters that fans on the heatsink) in degrees Celsius. Only valid if CanControlTemperature is True.

HostTimedExposure

Property

CCDCamera.HostTimedExposure (read only, bool)

Syntax

```
cam. HostTimedExposure = true;
```

Exceptions

Throws an exception if the HostTimedExposure property is unsupported, if the camera is not connected, or if the command is unsuccessful.

Examples:

VB:

```
cam.HostTimedExposure = true
```

C#:

```
cam.HostTimedExpsoure = true;
```

Remarks

On Interline Transfer CCDs, like the KAI-2020M in the QSI 520i, every other column of the CCD is masked to prevent light from striking the underlying pixels. To read an image from an Interline Transfer CCD, the active pixels are transferred to the masked pixels and then shifted out of the CCD.

In normal operation, the active pixels are “flushed” at the beginning of an exposure in order to remove any charge that had built up in the pixels since the last exposure. “HostTimedExposure” mode eliminates this flush allowing you to begin integrating the next exposure while the previous exposure is being transferred to the computer.

This special mode is generally only useful when taking a rapid sequence of short exposures. If significant time is allowed to pass between exposures in this mode, dark current will likely saturate the CCD. With “HostTimedExposure” mode enabled, it is possible to take small subframes at a rate of multiple images per second, while capturing the majority of the light striking the CCD. This can be useful for some rapid guiding applications.

Note: The “Pre-Exposure Flush” options in the Advanced Dialog box are ignored when in “HostTimedExposure” mode. The only flushing that occurs in this mode is to flush the masked columns prior to transferring the image into them for reading by the computer.

ImageArray

Property

CCDCamera.ImageArray (read only, SAFEARRAY of Long)

Syntax

CCDCamera.ImageArray

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Imports System.Threading
While Not cam.ImageReady
   Thread.Sleep(100)
End While
Dim image As Array = cam.ImageArray
Dim sizeX As Integer = image.GetLength(0)
Dim sizeY As Integer = image.GetLength(1)
Dim pixel As Long = CType(image.GetValue(0, 1), Int32)
```

C#:

```
using System.Threading;
while (!cam.ImageReady) { Thread.Sleep(100); }
Array image = cam.ImageArray;
int sizeX = image.GetLength(0);
int sizeY = image.GetLength(1);
long pixel = (Int32)image.GetValue(0, 1);
```

Remarks

Returns a SAFEARRAY of Long of size NumX * NumY containing the pixel values from the last exposure. The application must inspect the SAFEARRAY parameters to determine the dimensions. Note: if NumX or NumY is changed after a call to StartExposure it will have no effect on the size of this array. This is the preferred method for programs (not scripts) to download images since it requires much less memory.

For color or multi-spectral cameras, will produce an array of NumX * NumY * NumPlanes. If the application cannot handle multi-spectral images, it should use just the first plane.

The caller must test the property ImageReady and received a true result prior to using this property, to insure that the transfer of the image from the camera is complete.

ImageArrayVariant

Property

CCDCamera.ImageArrayVariant (read only, SAFEARRAY of Variant)

Syntax

CCDCamera.ImageArrayVariant

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Imports System.Threading
While Not cam.ImageReady
   Thread.Sleep(100)
End While
Dim image As Array = cam.ImageArrayVariant
Dim sizeX As Integer = image.GetLength(0)
Dim sizeY As Integer = image.GetLength(1)
Dim pixel As Long = CType(image.GetValue(0, 1), Int32)
```

C#:

```
using System.Threading;
while (!cam.ImageReady) { Thread.Sleep(100); }
Array image = cam.ImageArrayVariant;
int sizeX = image.GetLength(0);
int sizeY = image.GetLength(1);
long pixel = (Int32)image.GetValue(0, 1);
```

Remarks

Returns a SAFEARRAY of Variant of size NumX * NumY containing the pixel values from the last exposure. The application must inspect the SAFEARRAY parameters to determine the dimensions. Note: if NumX or NumY is changed after a call to StartExposure it will have no effect on the size of this array. This property should only be used from scripts due to the extremely high memory utilization on large image arrays (26 bytes per pixel). Pixels values should be in Short, Long, or Double format.

For color or multi-spectral cameras, will produce an array of NumX * NumY *NumPlanes. If the application cannot handle multi-spectral images, it should use just the first plane.

The caller must test the property ImageReady and received a true result prior to using this property, to insure that the transfer of the image from the camera is complete.

ImageReady

Property

CCDCamera.ImageReady (read only, Boolean)

Syntax

CCDCamera.ImageReady

Exceptions

Throws an exception if hardware or communications link error has occurred.

Examples:

VB:

```
Imports System.Threading
While Not cam.ImageReady
   Thread.Sleep(100)
End While
Dim image As Array = cam.ImageArrayVariant
Dim sizeX As Integer = image.GetLength(0)
Dim sizeY As Integer = image.GetLength(1)
Dim pixel As Long = CType(image.GetValue(0, 1), Int32)
```

C#:

```
using System.Threading;
while (!cam.ImageReady) { Thread.Sleep(100); }
Array image = cam.ImageArrayVariant;
int sizeX = image.GetLength(0);
int sizeY = image.GetLength(1);
long pixel = (Int32)image.GetValue(0, 1);
```

Remarks

If True, there is an image from the camera available. If False, no image is available and attempts to use the ImageArray method will produce an exception.

IsMainCamera

Property

CCDCamera.IsMainCamera (Boolean)

Syntax

CCDCamera.IsMainCamera

Exceptions

Throws an exception if the camera is connected when attempting to set the property.

Examples:

VB:

```
Try
   Cam.IsConnected = False
   If Not cam.IsMainCamera Then
      cam.IsMainCamera = True
   End If
   Cam.IsConnected = True
Catch [error] As Exception
End Try
```

C#:

```
try
{
   Cam.IsConnected == false;
   if (!cam.IsMainCamera)
   {
         cam.IsMainCamera = true;
   }
   Cam.IsConnected = true;
}
catch (Exception error) {}
```

Remarks

If True, the camera is in the main camera role; if False, the camera is in the guider role. The camera must be in the disconnected state prior to changing the camera role (when setting this property).

IsPulseGuiding

Property

CCDCamera.IsPulseGuiding (read only, Boolean)

Syntax

CCDCamera.IsPulseGuiding

Exceptions

Throws an exception if hardware or communications link error has occurred.

Examples:

VB:

```
If cam.IsPulseGuiding Then

End If
```

C#:

```
if (cam.IsPulseGuiding)
{

}
```

Remarks

If True, pulse guiding is in progress.

LastError

Property

CCDCamera.LastError (read only, String)

Syntax

CCDCamera.LastError

Exceptions

Throws an exception if no error condition.

Examples:

VB:

```
    Dim error as String
    error = cam.LastError
```

C#:

```
    string error;
    error = cam.LastError;
```

Remarks

Reports the last error condition reported by the camera hardware or communications link. The string may contain a text message or simply an error code.

LastExposureDuration

Property

CCDCamera.LastExposureDuration (read only, Double)

Syntax

CCDCamera.LastExposureDuration

Exceptions

Throws an exception if not supported or no exposure has been taken

Examples:

VB:

```
Dim time As Double
Try
   time = cam.LastExposureDuration
Catch err As Exception
End Try
```

C#:

```
double time;
try
{
   time = cam.LastExposureDuration;
}
catch( Exception err)
{

}
```

Remarks

Reports the actual exposure duration in seconds (i.e. shutter open time). This may differ from the exposure time requested due to shutter latency, camera timing precision, etc.

LastExposureStartTime

Property

CCDCamera.LastStartTime (read only, String)

Syntax

CCDCamera.LastStartTime

Exceptions

Throws an exception if not supported or no exposure has been taken.

Examples:

VB:

```
    Dim time As String
    Try
       time = cam.LastExposureStartTime
    Catch err As Exception
    End Try
```

C#:

```
    string time;
    try
    {
      time = cam.LastExposureStartTime;
    }
    catch( Exception err)
    {

    }
```

Remarks

Reports the actual exposure start in the FITS-standard CCYY-MM-DDThh:mm:ss[.sss...] format.

LEDEnabled

Property

QSICamera.put_LEDEnabled (Boolean)

Syntax

CCDCamera LEDEnabled = true;

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
enabled = cam.LEDEnabled;
if (enabled == false)
   cam.LEDEnabled = true;
```

Remarks

Enable or disable the camera status LED. The camera must be in the connected state. This setting is recorded in the system registry and will be maintained between connections to the device. This setting is unique for each device serial number.

ManualShutterMode

Property

CCDCamera.ManualShutterMode (bool ManualShutterMode)

Syntax

CCDCamera.ManualShutterMode = FanMode.ShutterQuiet;

bool mode = CCDCamera.ManualShutterMode;

Exceptions

Throws an exception if the ManualShutterMode property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
   cam.ManualShutterMode = true
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.ManualShutterMode = true;
}
catch (Exception error)
{}
```

Remarks

Enables the manual control (using the API) of the mechanical shutter, independent of the StartExposure command. See ManualShutterOpen for further details.

ManualShutterOpen

Property

CCDCamera.ManualShutterOpen (Boolean)

Syntax

CCDCamera.ManualShutterOpen = true;

Exceptions

Throws an exception if the ManualShutterOpen property is unsupported, if the camera is not connected, if the ManualShutterMode is not enabled, or if the command is unsuccessful.

Examples:

VB:

```
Try
   cam.ManualShutterOpen = true
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.ManualShutterOpen = true;
}
catch (Exception error)
{}
```

Remarks

Manually open (if set true) or closes (if set false) the mechanical shutter. The ManualShutterMode must be set true before issuing ManualShutterOpen commands.

MaskPixels

Property

CCDCamera.MaskPixels (Boolean)

Syntax

CCDCamera.MaskPixels [= Boolean]

Exceptions

Throws an exception if unsuccessful or if the camera is not connected.

Examples:

VB:

```
    if cam.MaskPixels = False Then
       cam.PaskPixels = True
    EndIf
```

C#:

```
    if (cam.MaskPixels == false)
       cam.MaskPixels = true;
```

Remarks

The QSI camera driver provides the capability to individually mask pixels, replacing the ccd value with a fixed value of 200 ADU. This is intended to mask out hot pixels that may interfere with othe post processing activities, such as auto guiding. Pixels to be masked are identified by their un-binned x,y location. Use the GetPixelMask and SetPixelMask to read or update the pixels to be masked. The MaskPixel property enables or disabled the post exposure masking of pixels.

The PixelMaks array and the status of the MaskPixel propery are stored in the registry by camera serial number and are automatically restored each time the camera is opened.

If the MaskPixel is enabled, each pixel listed in the PixelMask array is replaced with a fixed value, typically 200ADU. The actual value used depends on the zero target for the camera as set by the camera firmware. This value is 200 ADU on all QSI 500 series models.

MaxADU

Property

CCDCamera.MaxADU (read only, Long)

Syntax

CCDCamera.MaxADU

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Try
   Dim adu As Long
   adu = cam.MaxADU
Catch [error] As Exception

End Try
```

C#:

```
try
{
   long adu;
   adu = cam.MaxADU;
}
catch (Exception error)
{

}
```

Remarks

Reports the maximum ADU value the camera can produce.

MaxBinX

Property

CCDCamera.MaxBinX (read only, Short)

Syntax

CCDCamera.MaxBinX

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
    Dim maxBinX As Short
    maxBinX = cam.MaxBinX
```

C#:

```
    short maxBinX;
    maxBinX = cam.MaxBinX;
```

Remarks

If AsymmetricBinning = False, returns the maximum allowed binning factor. If AsymmetricBinning = True, returns the maximum allowed binning factor for the X axis.

MaxBinY

Property

CCDCamera.MaxBinY (read only, Short)

Syntax

CCDCamera.MaxBinY

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
    Dim maxBinY As Short
    maxBinY = cam.MaxBinY
```

C#:

```
    short maxBinY;
    maxBinY = cam.MaxBinY;
```

Remarks

If AsymmetricBinning = False, equals MaxBinX. If AsymmetricBinning = True, returns the maximum allowed binning factor for the Y axis.

MaxExposureTime

Property

CCDCamera.MaxExposureTime (read only, double)

Syntax

CCDCamera.MaxExposureTime

Exceptions

Throws an exception if data unavailable or if the camera is not connected.

Examples:

VB:

```
    Dim maxExp As Double
    maxExp = cam.MaxExposureTime
```

C#:

```
    double maxExp;
    maxExp = cam.MaxExposureTime;
```

Remarks

Returns the maximum exposure time in seconds for the connected camera.

MinExposureTime

Property

CCDCamera.MinExposureTime (read only, double)

Syntax

CCDCamera.MinExposureTime

Exceptions

Throws an exception if data unavailable or if the camera is not connected.

Examples:

VB:

```
Dim minExp As Double
minExp = cam.MinExposureTime
```

C#:

```
double minExp;
minExp = cam.MinExposureTime;
```

Remarks

Returns the minimum exposure time in seconds for the connected camera.

ModelNumber

Property

CCDCamera.ModelNumber (read only, String)

Syntax

CCDCamera.ModelNumber

Exceptions

Throws an exception if description unavailable

Examples:

VB:

```
Try
   Dim model As String
    model = cam.ModelNumber
Catch [error] As Exception

End Try
```

C#:

```
try
{
   string model;
   model = cam.ModelNumber;
}
Catch (Exception error)
{

}
```

Remarks

Returns the model number of the camera. The string shall not exceed 68 characters (for compatibility with FITS headers).

Name

Property

CCDCamera.Name (read only, String)

Syntax

CCDCamera.Name

Exceptions

Throws an exception if description unavailable

Examples:

VB:

```
Try
  Dim name As String
  name = cam.Name
Catch [error] As Exception

End Try
```

C#:

```
try
{
  string name;
  name = cam.Name;
}
Catch (Exception error)
{

}
```

Remarks

Returns the model name of the camera. The string shall not exceed 68 characters (for compatibility with FITS headers).

Named Filter Wheels

Properties

```
QSICamera. HasFilterWheelTrim(read-only, boolean);
QSICamera. FilterPositionTrim( SAFEARRAY of short);
QSICamera. FilterWheelNames( read-only SAFEARRAY of String);
QSICamera. SelectedFilterWheel(String);
QSICamera.NewFilterWheel(String);
QSICamera.DeleteFilterWheel(String);
```

Syntax

```
bollean hasWheel = cam.HasFilterWheelTrim;
short[] trims = (short[])cam. FilterPositionTrim;
cam. FilterPositionTrim = trims;
String[] names = (String[])cam. FilterWheelNames;
String name = cam. SelectedFilterWheel(&strName);
cam. SelectedFilterWheel = “CYMKWheel”;
cam.NewFilterWheel(“LRGBWHEEL”);
cam.DeleteFilterWheel(“CYMKWheel”);
```

Exceptions

If exceptions are enabled, throws an exception if the camera is not in the connected state.

Example:

```
if (cam.FilterWheelConnected )
{
  cam.NewFilterWheel("Test3");
  cam.SelectedFilterWheel = "Test3";
  String[] Names = (String[])cam.Names;
  if (Names.Length >= 5)
    {
     Names[0] = "Blue";
     Names[1] = "Green";
     Names[2] = "Red";
     Names[3] = "Luminence";
     Names[4] = "Empty";
    }
  cam.Names = Names;
  cam.NewFilterWheel("Test4");
  String[] FilterNames = (String[])cam.FilterWheelNames;
  cam.DeleteFilterWheel("Test3");
  String CurrentWheel = cam.SelectedFilterWheel;
  cam.Names = Names;
  cam.SelectedFilterWheel = "Test4";
  CurrentWheel = cam.SelectedFilterWheel;
}
```

Remarks

Filter wheels can now be referenced by name, to track the individual settings of the filter name, focus offset and trim setting on a per filter wheel basis. Filter wheel names are ascii character strings. The name "Default" is reserved for system use, in the case of no user defined wheels. The filter name databse is kept in the system registry.

NumX

Property

CCDCamera.NumX (Long)

Syntax

CCDCamera.NumX [= Long]

Exceptions

None

Examples:

VB:

```
Dim numX As Long
numX = cam.NumX
```

C#:

```
long numX;
numX = cam.NumX;
```

Remarks

Sets the sub-frame width. Also returns the current value. If binning is active, value is in binned pixels. No error check is performed when the value is set. Defaults to CameraXSize.

NumY

Property

CCDCamera.NumY (Long)

Syntax

CCDCamera.NumY [= Long]

Exceptions

None

Examples:

VB:

```
Dim numY As Long
numY = cam.NumY
```

C#:

```
long numY;
numY = cam.NumY;
```

Remarks

Sets the sub-frame height. Also returns the current value. If binning is active, value is in binned pixels. No error check is performed when the value is set. Default to CameraYSize.

PixelSizeX

Property

CCDCamera.PixelSizeX(read only, Double)

Syntax

CCDCamera.PixelSizeX

Exceptions

Throws exception if data unavailable.

Examples:

VB:

```
Try
   Dim x As Double
   x = cam.PixelSizeX
Catch [error] As Exception

End Try
```

C#:

```
try
{
   double x;
   x = cam.PixelSizeX;
}
catch (Exception error)
{

}
```

Remarks

Returns the width of the CCD chip pixels in microns, as provided by the camera driver.

PixelSizeY

Property

CCDCamera.PixelSizeY(read only, Double)

Syntax

CCDCamera.PixelSizeY

Exceptions

Throws an exception if data unavailable.

Examples:

VB:

```
Try
   Dim y As Double
   y = cam.PixelSizeY
Catch [error] As Exception

End Try
```

C#:

```
try
{
   double y;
   y = cam.PixelSizeY;
}
catch (Exception error)
{

}
```

Remarks

Returns the height of the CCD chip pixels in microns, as provided by the camera driver.

PowerOfTwoBinning

Property

CCDCamera.PowerOfTwoBinning (read only, Boolean)

Syntax

CCDCamera.PowerOfTwoBinning

Exceptions

None

Examples:

VB:

```
    If cam.PowerOfTwoBinning Then

    End If
```

C#:

```
    if (cam.PowerOfTwoBinning)
    {

    }
```

Remarks

If True, the camera bins in powers of two. If False, the camera bins in increments of one.

PreExposureFlush

Property

CCDCamera.PreExposureFlush (enum PreExposureFlushEnum)

Syntax

CCDCamera. PreExposureFlush = PreExposureFlushNormal;

Exceptions

Throws an exception if the PreExposureFlush property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
   cam.PreExposureFlush = PreExposureFlushEnum.FlushNormal
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.PreExposureFlush = PreExposureFlushEnum.FlushNormal;
}
catch (Exception error)
{}
```

Remarks

Controls the amount of pre-exposure flushing. Flushes any previously accumulated Dark Current from the CCD imager.

None – No flushing performed. Image will contain any dark current that had accumulated since the last exposure. This mode allows for the fastest back-to-back exposures.

The next 4 modes, Modest, Normal, Aggressive and Very Aggressive, provide increasingly higher levels of flushing by employing a number of different strategies. Higher levels of flushing take more time to execute.

For KAF based cameras the flush cycles for each setting are:

None – no flush cycles
Modest – 1 flush cycle
Normal – 2 flush cycles
Aggressive – 4 flush cycles
Very Aggressive – 8 flush cycles.

Symbolic Constants

The symbolic values for PreExposureFlushEnum are:

Constant	enum Value
FlushNone	0
FlushModest	1
FlushNormal	2
FlushAggresive	3
FlushVeryAggressive	4

ReadoutSpeed

Property

CCDCamera.ReadoutSpeed (enum)

Syntax

cam.ReadoutSpeed = FastReadout;

Exceptions

If exceptions are enabled, throws an exception if unsuccessful or if camera does not support this feature.

Examples:

VB:

```
cam.ReadoutSpeed = ReadoutSpeedEnum.FastReadout
```

C#:

```
cam.ReadoutSpeed = ReadoutSpeedEnum.FastReadout;
```

Remarks

Controls the readout speed of cameras that have read out speed control capability. Readout speed selection is a tradeoff between high image quality and fast image readout and download. Typically FastReadout is used during focusing and other setup operations and HighQualityImage is used during the final image capture.

The connection to the camera must be in the "connected" state to change the readout speed.

ReadoutSpeed will return an error or throw an exception if this setting is not available on the connected camera.

ReadoutSpeedEnum:

```
HighImageQuality = 0,
FastReadout = 1
```

SerialNumber

Property

CCDCamera.SerialNumber (read only, String)

Syntax

CCDCamera.SerialNumber

Exceptions

Throws an exception if description unavailable

Examples:

VB:

```
Try
  Dim num As String
   num = cam.SerialNumber
Catch [error] As Exception

End Try
```

C#:

```
try
{
  string num;
  num = cam.SerialNumber;
}
Catch (Exception error)
{

}
```

Remarks

Returns the serial number of the camera. The string shall not exceed 68 characters (for compatibility with FITS headers).

SetCCDTemperature

Property

CCDCamera.SetCCDTemperature (Double)

Syntax

CCDCamera.SetCCDTemperature [= Double]

Exceptions

Throws an exception if command not successful. Throws throw exception if CanSetCCDTemperature is False.

Examples:

VB:

```
Try
  Dim temp As Double
  temp = cam.SetCCDTemperature
  cam.SetCCDTemperature = 10.0
Catch [error] As Exception

End Try
```

C#:

```
try
{
  double temp;
  temp = cam.SetCCDTemperature;
  cam.SetCCDTemperature = 10.0;
}
catch (Exception error)
{

}
```

Remarks

Sets the camera cooler set-point in degrees Celsius, and returns the current set-point.

ShutterMode

No longer supported. See ShutterPriority.

ShutterPriority

Property

CCDCamera.ShutterPriority (enum ShutterPriorityEnum)

Syntax

CCDCamera. ShutterPriority = ShutterPriorityMechanical;

Exceptions

Throws an exception if the ShutterPriority property is unsupported, if the camera is not connected, or if the command is unsuccessful. This setting is stored in the system Registry and will be maintained between camera connections.

Examples:

VB:

```
Try
  cam. ShutterPriority = _
  ShutterPriorityEnum.ShutterPriorityMechanical
Catch [error] As Exception
End Try
```

C#:

```
try
{
  cam.ShutterPriority = ShutterPriorityEnum.
  ShutterPriorityMechanical;
}
catch (Exception error)
{}
```

Remarks

Controls the shutter Priority in dual shutter cameras. Only enabled when the camera has a mechanical shutter.

Mechanical – Shutter is closed between exposures and only opens for exposures that are taking an image, i.e. closed for Darks and Bias images. This mode prevents the CCD imager from being flooded with light in between exposures. This can reduce or eliminate 'ghost' or residual images when imaging bright objects.

Electronic – Shutter is open between exposures and only closes when taking Darks and Bias images. This mode allows the fastest back-to-back image exposures.

Symbolic Constants

The symbolic values for ShutterPriorityEnum are:

Constant	Value
ShutterPriorityMechanical	0
ShutterPriorityElectronic	1

SoundEnabled

Property

QSICamera.put_SoundEnabled (Boolean)

Syntax

CCDCamera SoundEnabled = true;

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
enabled = cam.SoundEnabled;
if (enabled == false)
   cam.SoundEnabled = true;
```

Remarks

Enable or disable the camera status beeper. The camera must be in the connected state. This setting is recorded in the system registry and will be maintained between connections to the device. This setting is unique for each device serial number.

StartX

Property

CCDCamera.StartX (Long)

Syntax

CCDCamera.StartX [= Long]

Exceptions

None

Examples:

VB:

```
    Dim x As Long
    x = cam.StartX
    cam.StartX = 0
```

C#:

```
    long x;
    x = cam.StartX;
    cam.StartX = 0;
```

Remarks

Sets the sub-frame start position for the X axis (0 based). Also returns the current value. If binning is active, value is in binned pixels.

StartY

Property

CCDCamera.StartY (Long)

Syntax

CCDCamera.StartY [= Long]

Exceptions

None

Examples:

VB:

```
    Dim y As Long
    y = cam.StartY
    cam.StartY = 0
```

C#:

```
    long y;
    y = cam.StartY;
    cam.StartY = 0;
```

Remarks

Sets the sub-frame start position for the Y axis (0 based). Also returns the current value. If binning is active, value is in binned pixels.

ICamera Methods

AbortExposure

Syntax

CCDCamera.AbortExposure()

Parameters

None.

Returns

Nothing

Exceptions

Throws an exception if camera is not idle and abort is unsuccessful (or not possible, e.g. during download). Throws an exception if hardware or communications error occurs.

Examples:

VB:

```
Try
   cam.AbortExposure()
Catch [error] As Exception

End Try
```

C#:

```
try
{
   cam.AbortExposure();
}
catch (Exception error)
{

}
```

Remarks

Aborts the current exposure, if any, and returns the camera to Idle state.

PulseGuide

Syntax

CCDCamera.PulseGuide(Direction, Duration)

Parameters

GuideDirections Direction - direction in which the guide-rate motion is to be made

Long Duration - Duration of the guide-rate motion (milliseconds)

Returns

Nothing.

Exceptions

Throws an exception if PulseGuide command is unsupported or the command is unsuccessful.

Examples:

VB:

```
Try
   cam.PulseGuide(GuideDirections.guideWest, 10)
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.PulseGuide(GuideDirections.guideWest, 10);
}
catch (Exception error)
{

}
```

Remarks

This method may return immediately after the move has started, in which case back-to-back dual axis pulse-guiding can be supported. Use the IsPulseGuiding property to detect when all moves have completed.

The pulse duration resolution provided by the camera is in 10ms increments. This duration argument to this function is in millisecond increments.

Symbolic Constants

The (symbolic) values for GuideDirections are:

Constant	Value	Description
guideNorth	0	North (+ declination/elevation)
guideSouth	1	South (- declination/elevation)
guideEast	2	East (+ right ascension/azimuth)
guideWest	3	West (- right ascension/azimuth)

Note: directions are nominal and may depend on exact mount wiring.

SetupDialog

Syntax

CCDCamera.SetupDialog

Parameters

None.

Returns

Nothing.

Exceptions

Must throw an exception if Setup dialog is unavailable.

Examples:

VB:

```
    cam.SetupDialog()
```

C#:

```
    cam.SetupDialog();
```

Remarks

Launches a configuration dialog box for the camera. The call will not return until the user clicks OK or cancel manually. The camera Connected state must be false to change setup values for the camera.

StartExposure

Syntax

CCDCamera.StartExposure (Duration, Light)

Parameters

Double Duration - Duration of exposure in seconds

Boolean Light - True for light frame, False for dark frame (ignored if no shutter)

Returns

Nothing.

Exceptions

Throws an exception if NumX, NumY, XBin, YBin, StartX, StartY, or Duration parameters are invalid. Throws an throw exception if CanAsymmetricBin is False and BinX <> BinY. Throws an exception if the exposure cannot be started for any reason, such as a hardware or communications error.

Examples:

VB:

```
Try
   cam.StartExposure(1.00, True)
Catch [error] As Exception
End Try
```

C#:

```
try
{
   cam.StartExposure(1.00, true);
}
catch (Exception error)
{

}
```

Remarks

Starts an exposure. You must use ImageReady to check when the exposure is complete.

StopExposure

Syntax

CCDCamera.StopExposure()

Parameters

None.

Returns

Nothing.

Exceptions

Throws an exception if CanStopExposure is False. Throws an exception if no exposure is in progress. Throws an exception if the camera or link has an error condition. Throws an exception if for any reason if no image readout will be available.

Examples:

VB:

```
Try
   If cam.CanStopExposure Then
      cam.StopExposure()
   End If
Catch [error] As Exception
End Try
```

C#:

```
try
{
   if (cam.CanStopExposure)
         cam.StopExposure();
}
catch (Exception error)
{

}
```

Remarks

Stops the current exposure, if any. If an exposure is in progress, the readout process is initiated. Ignored if readout is already in process.

ICameraEx Methods

GetPixelMask / SetPixelMask

Method

CCDCamera.GetPixelMask(SAFEARRAY<INT>* x, SAFEARRAY<INT>* y)
CCDCamera.SetPixelMask(SAFEARRAY<INT> x, SAFEARRAY<INT> y)

Syntax

CCDCamera.GetPixelMask(&x, &y);
CCDCamera.SetPixelMask(x, y);

Exceptions

Throws an exception if unsuccessful or if the camera is not connected.

Example:

VB:

```
Dim X As Array
Dim Y As Array
cam.GetPixelMask(X, Y)
. . .
Dim x As Int32() = New Int32(2)
Dim y As Int32() = New Int32(2)
x(0) = 23
y(0) = 40
x(1) = 107
y(1) = 200

cam.SetPixelMask(x, y)
cam.MaskPixels = True
```

C#:

```
Array X;
Array Y;
cam.GetPixelMask(out X, out Y); // Get prior settings
  . . .
Int32[] x = new Int32[2];
Int32[] y = new Int32[2];
x[0] = 23;    // Pixel (23,40)
y[0] = 40;
x[1] = 107;   // Pixel (107,200)
y[1] = 200;

cam.SetPixelMask(x, y); // Set the pixels to mask
cam.MaskPixels = true;  // Enable masking
```

Remarks

The QSI camera driver provides the capability to individually mask pixels, replacing the ccd pixel value with a fixed value of 200 ADU. This is intended to mask out hot pixels that may interfere with other post processing activities, such as auto guiding. Pixels to be masked are identified by their un-binned x, y location.

Use the GetPixelMask and SetPixelMask methods to read or update the pixel addresses to be masked. Pixel addresses are expressed as un-binned x and y addresses, starting at (0,0). The x argument is a SAFEARRAY of INT values that specifies the x location for each pixel. The nth entry of the array is the x address of the nth pixel to mask. The same holds true for the y array. The x and the y arrays must always be of the same size. Empty arrays are acceptable.

Use the MaskPixels property to enable or disable the post exposure masking of pixels.

The PixelMask array and the status of the MaskPixels properly are stored in the registry by the connected camera serial number and are automatically restored each time the camera is opened.

If MaskPixels is enabled, each pixel listed in the PixelMask array is replaced with a fixed value, typically 200 ADU. The actual value used depends on the zero target for the camera as set by the camera's firmware. This value is 200 ADU on all QSI 500 series models.

External Trigger Input

Methods

CCDCamera.EnableTriggerMode(TriggerModeEnum, TriggerPolarityEnum)
CCDCamera.TerminatePendingTrigger(void)
CCDCamera.CancelTriggerMode(void)

```
enum
{
        ShortWait = 4,
        LongWait = 6
} TriggerModeEnum;


enum
{
        HighToLow = 0,
        LowToHigh = 1
} TriggerPolarityEnum;
```

Syntax

CCDCamera.SetTriggerMode(LongWait, HighToLow);
CCDCamera.TerminatePendingTrigger();
CCDCamera.CancelTriggerMode();

Exceptions

Throws an exception if unsupported on the camera, unsuccessful, or if the camera is not connected.

Example:

C#:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using QSICameraLib;

namespace QSIExtTrigger
{
    class Program
    {
        static void Main(string[] args)
        {
            CCDCamera cam = new CCDCamera();
            int count = cam.QSIDeviceCount;
            int numCam = cam.QSISerialNumbers.Length;
            if (numCam > 0)
            {
                string serialNumber = (string)cam.QSISerialNumbers.GetValue(0);
                cam.QSISelectedDevice = serialNumber;
            }
            else
            {
                Console.WriteLine("No Cameras Found\n");
                return;
            }
```

```
try
{
    cam.Connected = true;
}
catch (Exception e)
{
    Console.WriteLine("Unable to connect to camera\n");
}
////////////////////////////////////////////////////////////////////////////
// Short Wait Trigger Mode
////////////////////////////////////////////////////////////////////////////

// Set Short Wait External Trigger (4 seconds max), Pos to Neg polarity
cam.EnableTriggerMode(TriggerModeEnum.ShortWait,TriggerPolarityEnum.HighToLow);
try
{
    cam.StartExposure(0.3, true);
  // Short wait, so this will return with an image, or will timeout
    while (!cam.ImageReady)
    {
        Thread.Sleep(1);
    }
    int x = cam.NumX;
    int y = cam.NumY;
    int len = x * y;
    ushort[] data = new ushort[len];

    // Copy the SAFEARRAY to a array to improve speed
    int[,] pixels = new int[x, y];
    Array.Copy((System.Array)cam.ImageArray, pixels, x * y);
}
catch (Exception e)
{
    // Timeout comes here

}

// turn off external trigger mode
cam.CancelTriggerMode();

/////////////////////////////////////////////////////////////////////////
// Long Wait Trigger Mode
////////////////////////////////////////////////////////////////////////

cam.EnableTriggerMode(TriggerModeEnum.LongWait, TriggerPolarityEnum.LowToHigh);
try
{
    // Start a long wait exposure
    cam.StartExposure(0.3, true);
    // Sleep for 5 seconds as a demo
    Thread.Sleep(5000);
    //
    // Demo cancelling of the pending trigger
    // This would not normally be done...
    cam.TerminatePendingTrigger();
}
catch (Exception e)
{

}

 // Start a new exposure with long wait trigger.
 //Note that the trigger mode remains in effect until canceled.
try
{
     cam.StartExposure(0.3, true);
    while (!cam.ImageReady)
    {
        Thread.Sleep(1);
    }
    int x = cam.NumX;
    int y = cam.NumY;
    int len = x * y;
    ushort[] data = new ushort[len];

    // Copy the SAFEARRAY to a array to improve speed
    int[,] pixels = new int[x, y];
    Array.Copy((System.Array)cam.ImageArray, pixels, x * y);
```

```
                }
                catch (Exception e)
                {

                }

                ////////////////////////////////////////////////////////////////////
                // Cancel Trigger Mode
                ////////////////////////////////////////////////////////////////////
                cam.CancelTriggerMode();
                cam.Connected = false;
            }
        }
    }
```

Remarks

Introduction:

The External Trigger Input is used to configure the camera to trigger exposures using an externally generated electronic signal.

Some QSI camera models support electronic input trigger through the opto-isolated signals on the control port. The camera is placed into ExternalTriggerMode with the ExternalTriggerMode method. The first parameter selects short or long wait mode, and the second parameter specifies the polarity of the trigger pulse that will fire the shutter. Shutter open timing is controlled by the StartExposure command.

When the camera first receives a StartExposure command and when in the External Trigger mode, a number of operations must be performed. Two of these operations can take a sizable amount of time to complete.

- CCD Imager Flush - QSI cameras have numerous flushing modes, from None to Very Aggressive. This can insert up to 1000 msec of delay from the receipt of the Start Exposure command and the camera becoming Armed and ready to receive an External Trigger. Additionally, camera models with smaller sensors (e.g. KAI-2020 vs KAI-04022) flush faster due to the smaller image array. Use a less aggressive flush for the shortest latency.

- Mechanical Shutter - If the camera is equipped with a mechanical shutter and it is closed at the time of the receipt of the StartExposure command, the shutter must be opened this can take as much as 200msec depending on the beginning shutter state.

Consequently, the time to arm the camera is not a fixed value. Ordinarily, the device generating the external trigger should validate the CameraArmed output from the camera before allowing an ExternalTrigger input to be sent. If the trigger arrives before the camera is armed, then the camera will immediately trigger as soon as it enters the armed state.

As soon as the camera is triggered, the CameraArmed output is returned to the unarmed state by the camera.

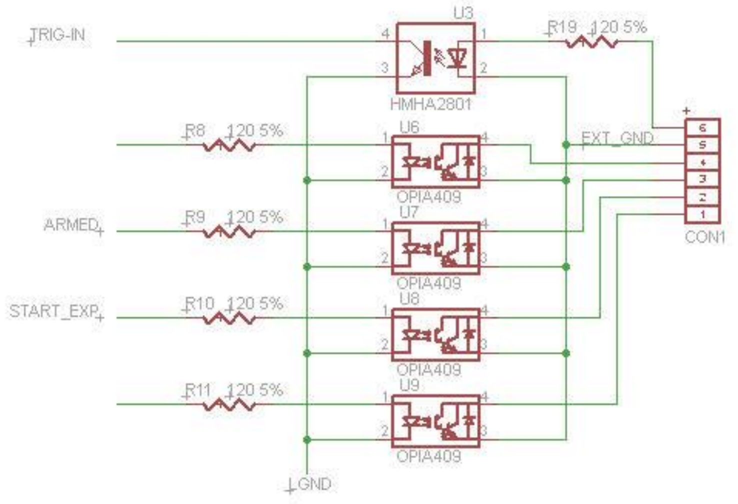
Electrical Interface:

On the control port RJ-11 connector, pin 6 is the trigger signal input (3.3V LVCMOS, +5V maximum input @30mA) and pin 5 is the common ground return. Minimum trigger pulse length is 30µs in short wait mode and 10ms in long wait mode (see below for an explanation of short and long wait modes).

When armed, the CameraArmed output (pin 3 of the control port) will be pulled down to the local ground level (pin 5) via an opto-isolator output. The opto-isolator outputs require positive signals relative to ground and are limited to 60V Vce, 60mA Ice. Vcesat is approximately 1V @ 2mA with maximum dissipation of 150mW.

Due to the nature of the opto-isolator trigger electronics, the low-to-high transition mode triggers the exposure with less latency compared to the high-to-low transition mode. Typical 3.3Vtrigger times (in Short Wait mode) are between 5µs and 11µs when using low-to-high triggering and between 20µs and 28µs when using high-to-low triggering.

The Control port schematic for QSI cameras with external trigger control is shown below.



Note: Specific parts numbers are subject to change without notice.

Programming the ExternalTriggerMode:

To initiate an exposure using the External Trigger Input, execute the EnableTriggerMode method, selecting the mode and polarity desired. The camera will remain in the selected external trigger mode until the mode is cancelled using the CancelExternalTrigger() method. On initial connection of the API to the camera, the external trigger mode always resets to the disabled state.

Once in the external trigger mode, exposures are taken in the normal API programming sequence, ie:

StartExposure(),
Loop unitl ImageReady is true,
Retrieve image using the ImageArray property.

The camera will pause during or after the StartExposure (depending on the wait mode) and will wait for the external trigger signal before firing the shutter and completing the exposure process. A pending long wait mode exposure may be cancelled by issuing the TerminatePendingExposure() method. A short wait exposure occurs during the StartExposure method and therefore cannot be cancelled. It will either complete with a trigger, or timeout.

The EnableTriggerMode has two modes of operation:

ShortWait – ShortWait mode provides the shortest latency and highest repeatability between the trigger transition and the start of the exposure. ShortWait mode is only available on supported camera models with interline transfer sensors, such as the KAI-2020 and the KAI-04022. In ShortWait mode, background processing in the camera (e.g. processing get temperature requests from the host) is suspended while waiting for the trigger. The maximum length of "Short" mode is 4 seconds.

If no trigger is received during the timeout period, (maximum 4 seconds) the camera reports an error to the API and an exception is thrown in the API. Maximum latency between receiving the Input Trigger and beginning the exposure in ShortWait mode is 10µs. A short wait occurs DURING the StartExposure method and that method will not return to the user until (1) the trigger is fired, or (2) the 4 seconds timeout elapses and the exception is thrown.

The calling program should always surround the StartExposure with a try/catch block to catch the timeout condition. If the timeout error is thrown, the calling application should not attempt to retrieve an image, as there will be none available. No further action is required before the next exposure may be attempted.

LongWait – LongWait mode has higher latency than ShortWait mode but allows the camera to wait for the Input Trigger indefinitely. Maximum latency between receiving the Input Trigger and beginning the exposure in LongWait varies depending on the camera model and whether a mechanical shutter is being used to time the exposure. With interline transfer sensors, the maximum latency is LongWait mode mode is 5ms. With full-frame sensors which utilize the mechanical shutter, maximum latency before the exposure begins is variable up to 600ms depending on shutter latencies.

Once the camera is in LongWait mode, the calling program should issue a StartExposure and then poll the camera status waiting for the ImageReady property to come true. The calling API may also poll for camera temperature during this wait loop. Other than CancelTriggerMode, no other commands may be issued to the camera until the exposure completes, or is canceled by the calling program.

When the external trigger is triggered, the camera will perform the selected preexposure flush, fire the shutter, complete the exposure, and set ImageReady true. The application can then retrieve an image using the ImageArray property as usual. The camera will stay in LongWait mode for each subsequent exposure until explicitly canceled by the CancelTriggerMode() method, or when the camera is re-connected by the API.

In LongWait mode, it is up to the calling application to time out the exposure request if desired. Following a StartExposure in long wait mode, the exposure may be cancelled by calling the TerminatePendingExposure method. No further action is required before the next exposure may be attempted.

If the TerminatePendingExposure command arrives at the camera just after an exposure triggered, the camera will necessarily complete timing of the current exposure. If the exposure shutter time is greater than 0.2 seconds, the caller may issue an AbortExposure to prematurely complete the shutter timing. A new exposure may be started without reading out the image.

IFilterWheel Properties

Connected

Property

FilterWheel.Connected (Boolean)

Syntax

FilterWheel.Connected [= Boolean]

Exceptions

Throughs an exception if connection attempt fails

Examples:

VB:

```
    if cam.IFilterWheel_Connected = False Then
       cam.FilterWheel_Connected = True
    EndIf

End If
```

C#:

```
    if (cam.IFilter_Connected == false)
       cam.IFilterWheel_Connected = true;
```

Remarks

Controls the link between the driver and the filter wheel. Set True to enable the link. Set False to disable the link. You can also read the property to check whether it is connected. This method is provided for compatibility. Use the preferred cam.Connected instead (member of ICamera).

SetupDialog

Syntax

FilterWheel.SetupDialog

Parameters

None.

Returns

Nothing.

Exceptions

Throws an exception if no dialog box available

Examples:

VB:

```
    cam.IFilterWheel_SetupDialog()
```

C#:

```
    cam.IFilterWheel_SetupDialog();
```

Remarks

Launches a configuration dialog box for the driver. The call will not return until the user clicks OK or cancel manually. The dialog is the same as cam.SetupDialog(). This method is provided for compatibility. Use the preferred cam.SetupDialog() instead (member of ICamera).

Names

Property

FilterWheel.Names (Array of String)

Syntax

FilterWheel.Names(i)

Exceptions

Throws an exception if filter number (i) is invalid

Examples:

VB:

```
    Dim name As String = CStr(cam.Names.GetValue(1))
    cam.Names.SetValue("Green", 1)
```

C#:

```
    string name = (string)cam.Names.GetValue(1);
    cam.Names.SetValue("Green", 1);
```

Remarks

For each valid slot number (from 0 to N-1), reports the name given to the filter position. These names would usually be set up by the user via the SetupDialog. The number of slots N can be determined from the length of the array. If filter names are not available, then it should report back "Filter 1", "Filter 2", etc.

Position

Property

FilterWheel.Position (Short)

Syntax

FilterWheel.Position [= Short]

Exceptions

Throws an exception if filter number is invalid. Throw an exception if link error occurs.

Examples:

VB:

```
    Dim pos As Short = cam.Position
    cam.Position = 2
```

C#:

```
    short pos = cam.Position;
    cam.Position = 2;
```

Remarks

Write number between 0 and N-1, where N is the number of filter slots (see Filter.Names). Starts filter wheel rotation immediately when written. Reading the property gives current slot number (if wheel stationary) or -1 if wheel is moving.

FocusOffsets

Property

FilterWheel.FocusOffsets (Array of Long)

Syntax

FilterWheel.FocusOffsets(i)

Exceptions

Throws an exception if filter number (i) is invalid

Examples:

VB:

```
    Dim offset As Long = CLng(cam.FocusOffset.GetValue(1))
    cam.FocusOffset.SetValue(230, 1)
```

C#:

```
    long offset = (long)cam.FocusOffset.GetValue(1);
    cam.FocusOffset.SetValue(230, 1);
```

Remarks

For each valid slot number (from 0 to N-1), reports the focus offset for the given filter position. These values are focuser- and filter -dependent, and would usually be set up by the user via the SetupDialog. The number of slots N can be determined from the length of the array. If focuser offsets are not available, then it should report back 0 for all array values.

Index