Volume

1

QUANTUM SCIENTIFIC IMAGING

500, 600, and RS Series CCD Imaging Cameras

QSI Linux API Reference v7.6

Revision Date: 29 March 2017 8:14 PM

LINUX

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 Linux API to control your QSI camera

The QSI Linux API provides a programming interface on Linux platforms to capture images and control QSI cameras using the C++ programming language.

The QSI Linux API exposes a QSI Camera object for interacting with the camera. The application calls methods on the object to perform various actions, such as starting an exposure. The camera object provides get_ and put_ property methods to control information about the camera that can be retrieved or set to a new value.

How to Use the QSI Camera Linux API

Requirements

The API requires Linux kernel 2.6 or beyond and is fully tested with 32 bit Fedora 24. See the README file in the qsiapi package for the latest information.

The kernel should be configured for USB support.

The 500 and 600 series camera use a USB interface shipset from FTDI, Inc. There are two driver library packages available to support FTDI chipsets on Linux. One is a proprietary library from FTDI called libftd2xx and an open source library from www.intra2net.com called libftdi-1.0.

You must also install the user mode libftd2xx driver library for FTDI (NOT the VCP drivers) **or** the open source libftdi, and configure the QSI build to use which ever driver library you choose. The libftd2xx is the default configured driver (but you still must obtain the driver from ww.ftdichip.com and install it).

The libftd2xx driver library is certified and supported by FTDI (www.ftdichip.com). The open source solution is preferred in cases where the release is built on platforms not supported by the FTDI libftd2xx drivers. You may want to try each library (one at a time) and see which version best suits the needs of your application.

For more information, see:

http://www.ftdichip.com/Drivers/D2XX.htm

http://www.intra2net.com/en/developer/libftdi/download.php

For the api to be used by users without root privileges, the raw USB devices *(/dev/bus/usb/*)* **must be configured for the appropriate user read/write permissions.** See Appendix A in this document, and the udev manual pages for further information.

Installation

Insure that the kernel is configured for USB support and the device permissions are correct. Connect the QSI camera to AC power and connect the USB cable from the camera to an available USB port on the computer. Use lsusb to verify proper USB installation (with the camera plugged in and connected via usb).

Installing the support libraries for the FTDI USB chipset.

With root permissions, select one of the two options below:

If you choose the FTDI LIBFTD2XX driver library, download, modify, compile, and install libftd2xx from FTDI.

(Available at: www.ftdichip.com)

Follow the installation instructions provided with the libftd2xx distribution. You must install version 1.1.12 or later. Insure no earlier versions are installed. Look in /usr/lib or /usr/local/lib, etc. for prior installations. See the included README for the latest installation instructions.

Briefly:

As user root:

Extract and install the driver files.

tar –xvf libftd2xx1.1.12.tar.gz

The extraction process should create a directory call ftd2xx1.0.4

cd ftd2xx.1.1.12

Follow the installation instructions in the README.DAT file:

Installation:

1. unzip and untar the file given to a suitable directory

tar -xvf libftd2xx1.1.12.tar

2. Change directory to the required architecture subdirectory, build/i386 for 32-bit or build/x86_64 for 64-bit.

3. As root user copy the following files to /usr/local/lib
cp libftd2xx.so.1.1.12 /usr/local/lib

3. Change directory to /usr/local/lib
cd /usr/local/lib

4. make symbolic links to these files using the following commands:
ln -s libftd2xx.so.1.1.12 libftd2xx.so

5. Change directory to /usr/lib
cd /usr/lib

6. make symbolic links to these files using the following commands:
ln -s /usr/local/lib/libftd2xx.so.1.1.12 libftd2xx.so

This completes the FTDI libftd2xx driver installation.

If you chose the open source drivers, install the ftdi1 library.

Use you package manager (rpm/dnf/app-get) to install the libftdi and libftdi-devel packages.

As an alternative, you can download and install the package manually: See http://www.intra2net.com/en/developer/libftdi/download.php for download and installation instructions. Be sure to install the latest 1.0 versions of the library and not the deprecated 0.20 version.

Look in /usr/lib or /usr/local/lib, etc. for prior installations of libftdi.

This completes the FTDI USB driver library installation.

If you have problems with this check with lsusb to check the usb file system is mounted properly.

Other problems will be related to the ftdi_sio driver loading. You must unload this driver (and usbserial) if it is attached to your device ("rmmod ftdi_sio" and "rmmod usbserial" as root user).

Note on the Cypress USB3 driver stack available in this release:

As of release 7.4, QSI cameras using the Cypress FX3 USB 3.0 chipset are now supported. If you wish to include the Cypress device support, add the --enable-cyusb to the configure command line. This is experimental and is not used by 500 and 600 series cameras.

For example:

./configure --enable-cyusb

Otherwise, proceed with the instructions below to add the require supporting libraries:

This change requires that the Cypress libcyusb library, udev development files, and the libusb-1.0 libraries be installed prior to building the QSI API release.

You must also install the development package for libudev. On Fedora before release 18, the package name is "libudev-devel". On Fedora 18 and beyond, the package name is systemd-devel. On Ubuntu, the package name is "libudev-dev".

This package installs the libudev.h header file required by libcyusb.

For example:

yum install libudev-devel

The Cypress libcyusb release can be found at:
http://www.cypress.com/?docID=42387&dlm=1

Download and install fx3_sdk_v1.3.1_linux.tar.gz (or the latest version) from the link above. The Cypress download site will require that you create a login for their website to allow the download. Registration is free of charge. Unzip the downloaded file:

tar -xvf fx3_sdk_V1.3.1_linux.tar.gz

Run make in the top directory to build the library:

make

Run make in the src directory:

cd src
make
cd ..

Run the install.sh script in the top directory to install the library. You must have root permissions in install the library.

sudo ./install.sh

Copy the cyusb.h header file to the /usr/local/include directory:

cp cyusb.h /usr/local/include

Update the shared library cache:

sudo ldconfig -n /usr/local/lib

The installation of the Cypress FX3 library is now complete.

Installing the QSI Linux API Release

Extract the release files from the qsiapi tar archive.

```
tar –xvf qsiapi.7.6.x.tar

(where x in 7.6.x.tar.gz is the current build number)
```

Configure the release for the ftdi driver of your choice:

Use one of the following configuration options:

--with-ftd=ftd2xx	for the ftd2xx driver
--with-ftd=ftdi	for the ftdi version 0.20 driver (Deprecated)
--with-ftd=ftdi1	for the ftdi1 driver

Run the GNU build tools:

In the extracted qsiapi directory:

if you chose the libftd2xx drivers:

```
./configure –-with-ftd=ftd2xx
make all
make install
```

If you chose the 0.20 libftdi drivers: (not recommended, for legacy support)

```
./configure –-with-ftd=ftdi
make all
make install
```

If you chose libftdi1 drivers:

```
./configure –-with-ftd=ftdi1
make all
make install
```

With all three cases shown above, you may add the –enable-cyusb option if you wish to support QSI cameras that utilize the Cypress USB 3.0 chipset.

Confirm the installation of the header and library files.

The include files `qsiapi.h` and `QSIError.h` are installed in `/usr/local/include`. The library files `libqsiapi.so`, `libqsiapi.a`, etc. are installed in `/usr/local/lib`.

Run ldconfig for the newly installed libraries:

```
cd /usr/local/lib
ldconfig /usr/local/lib
```

Confirm that all libraries are loaded and the system is properly configured.

```
./src/qsiapitest
```

qsiapitest first displays the version of the api. If there is a QSI camera connected to the system, it will execute a series of command to exercise the camera. See the qsiapitest.cpp source code for further details.

Troubleshooting:

If you have problems with the installation steps above, first use lsusb to confirm that the USB file system is mounted properly and the QSI camera is connected to the system.

Some problems may be related to the ftdi_sio driver loading. You must unload this driver (and usbserial) if it is attached to your device ("rmmod ftdi_sio" and "rmmod usbserial"as root user).

Insure that the user running the api has read/write access to the camera usb raw device. Check the permissions in /dev/bus/usb/XXX/YYY, where XXX is a hub number and YYY is the device number obtained from lsusb.

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 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 NumXand NumY which define the binned size of the frame. Pixel locations are zero based; ranging from 0 to X - 1, or 0 to Y - 1.

Binning pixels 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 pixels per bin for each axis. If CanAsymmetricBin is False, BinX must equal BinY.

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

The total number of pixels in an 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 an 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 (third party) filter wheel.

Programming notes

Programs using the API should include the header files `qsiapi.h` and `QSIError.h` which are normally installed in `/usr/local/include`. Add `-I/usr/local/include` to your compiler options if the `/usr/local/include` directory is not in you default include search path.

Programs using the API must link to the `libqsiapi` library which is normally installed in `/usr/local/lib`. Use the `-lqsiapi` option with the linker to properly build an executable that uses the api. Add the `-L/usr/local/lib` option to direct the linker to the library directory.

QSICamera object errors are reported either by throwing an exception or by testing the method return value. Throwing exceptions on errors can be enabled by setting put_EnableStructured Exceptions(true). If this property is set to false, methods will not throw an exception on error. In either case, camera methods will return an int error code.

For example:

```
double eGain;
int result;
QSICamera cam;

cam.put_UseStructuredErrorExceptions(true);
try
{
   result = cam.get_ElectronsPerADU(eGain;
}
catch (std::runtime_error& err) {/*handle error*/ }

—or—

cam.put_UseStructuredErrorException(false):
result = cam.get_ElectronsPerADU(eGain);
if (result != 0) { /*handle error*/}
```

Chapter

2

API Reference

QSICamera properties and methods

Properties

AntiBlooming
BinX
BinY
CameraGain
CameraState
CameraXSize
CameraYSize
CanAbortExposure
CanAsymmetricBin
CanGetCoolerPower
CanPulseGuide
CanSetCCDTemperature
CanSetGain
CanStopExposure
CCDTemperature
Connected
CoolerOn
CoolerPower
Description
DriverInfo
ElectronsPerADU
EnableShutterStatusOutput
FanMode
FilterCount
FilterOffset
FlushCycles
FullWellCapacity
HasFilterWheel
HasFIlterWheelTrim
HasShutter
HeatSinkTemperature
HostTimedExposure
ImageArray
ImageArraySize
ImageReady
IsMainCamera
IsPulseGuiding
LastError
LastExposureDuration
LastExposureStartTime
LEDEnabled
ManualShutterMode
ManualShutterOpen
MaxADU
MaxBinX
MaxBinY
ModelNumber
Name
Named Filter Wheels

Names
NumX
NumY
PixelMask
PixelSizeX
PixelSizeY
Position
PowerOfTwoBinning
PreExposureFlush
ReadoutSpeed
SelectCamera
SelectedFilterWheel
SerialNumber
SetCCDTemperature
ShutterMode
ShutterPriority
SoundEnabled
StartX
StartY
UseStructuredExceptions

Note: Properties associated with named filter wheels are documented in the section "Named Filter Wheels", not under the individual property names.

Methods

AbortExposure
PulseGuide
StartExposure
StopExposure
External Trigger Input

AntiBlooming

Property

int QSICamera.put_AntiBlooming (enum AntiBloom)

int QSICamera.get_AntiBloom(enum AntiBloom *)

Syntax

```
int result = cam.put_AntiBlooming( x );
```

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 .QSIConfig file and will be maintained between camera connections.

Examples:

QSICamera cam;

```
//
// example with UseStructedExceptions == false
//
int result = cam.put_AntiBlooming(QSICamera::AntiBloomNormal);
if (result != 0)
{
        //handle error//
}
```

Remarks

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

Symbolic Constants

The symbolic values for AntiBloom are:

Constant	Value
AntiBloomNormal	0
AntiBloomHigh	1

BinX

Property

int QSICamera.put_BinX (short)

int QSICamera.get_BinX(short*)

Syntax

```
int result = cam.put_BinX( x );
int result = cam.get_BinX(&x );
```

Exceptions

If exceptions are enabled, throws an exception for illegal binning values

Examples:

QSICamera cam;

```
short binX;
//
// example with UseStructedExceptions == false
//
int result = cam.get_BinX(&binX);
if (result != 0)
{
        //handle error//
}
//
// Example with UseStructedExceptions == true
//
try
{
  cam.put_BinX(2);
}
catch (std::runtime_error& err)
{
  cout << err.what();
}
```

Remarks

Gets or sets the binning factor for the X axis in parameter 1. 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 issuance of the StartExposure method.

BinY

Property

int QSICamera.put_BinY (short)

int QSICamera.get_BinY (short*)

Syntax

```
int result = cam.put_BinY (y);
int result = cam.get_BinY(&y);
```

Exceptions

If exceptions are enabled, throws an exception for illegal binning values

Examples

```
QSICamera cam;
short binY, binX;
int results;
results = cam.get_BinY(&binY);
results = cam.put_BinY(binX);
```

Remarks

Gets or sets the binning factor for the Y axis in parameter 1. 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 issuance of the StartExposure method.

CameraGain

Property

int QSICamera.put_CameraGain (enum CameraGain)

int QSICamera.get_CameraGain(enum CameraGain *)

Syntax

```
int result = cam.put_CameraGain(QSICamera::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 .QSIConfig file and will be maintained between camera connections.

Examples:

QSICamera cam;

```
//
// example with UseStructedExceptions == false
//
int result = cam.put_CameraGain(QSICamera::CameraGainLow);
if (result != 0)
{
   //handle 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 AntiBloom are:

Constant	Value
CameraGainHigh	0
CameraGainLow	1
CameraGainAuto	2

CameraState

Property

int QSICamera.get_CameraState (QSICamera::CameraState*)

Syntax

results = cam.get_CameraState(&state);

Exceptions

If exceptions are enabled, throws an exception if the camera status is unavailable.

Examples

```
int results;
QSICamera::CameraState state;

results = cam.get_CameraState(&state);
if (results == 0 && state == CameraIdle)
{
   return;
}
```

Remarks

Returns one of the following status information values in parameter 1:

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

int QSICamera.get_CameraXSize(long)

Syntax

result = cam.get_CameraXSize(&size);

Exceptions

If exceptions are enabled, thows exception if the value is not known.

Example

```
long size;
int results;
results = cam.get_CameraXSize(&size);
```

Remarks

Returns the width of the CCD sensor in un-binned pixels in parameter 1.

CameraYSize

Property

int QSICamera.get_CameraYSize (long*)

Syntax

result = cam.get_CameraYSize(& size);

Exceptions

If exceptions are enabled, throws exception if the value is not known.

Examples:

```
long size;
int results;
results = cam.get_CameraYSize(&size);
```

Remarks

Returns the height of the CCD sensor in un-binned pixels in parameter 1.

CanAbortExposure

Property

int QSICamera.get_CanAbortExposure (bool*)

Syntax

result = QSICamera.get_CanAbortExposure(& canAbort);

Exceptions

None.

Examples:

```
QSICamera cam;
bool canAbort;
cam.get_CanAbortExposure(&canAbort);
if (canAbort)
   cam.AbortExposure ();
```

Remarks

Returns true in parameter 1 if the camera can abort exposures; false if not.

CanAsymmetricBin

Property

int QSICamera.get_CanAsymmetricBin (bool*)

Syntax

int result =cam.get_CanAsymmetricBin(&canAsymBin);

Exceptions

If exceptions are enabled, throws an exception if the value is not known.

Examples:

```
try
{
   bool canAsymBin;
   cam.get_CanAsymmetricBin(&canAsymBin);
   if (canAsymBin)
   {
       cam.BinX = 1;
       cam.BinY = 2;
   }
}
catch (std::runtime_error& err)
{
}
```

Remarks

If parameter 1 is 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

int QSICamera.get_CanGetCoolerPower (bool*)

Syntax

result =cam.get_CanGetCoolerPower(&canGetPower);

Exceptions

None

Examples:

```
bool canGetPower;
int results;
results = cam.get_CanGetCoolerPower(&canGetPower);
if (canGetPower)
{
   int power;
   cam.get_coolerPower(&power);
}
```

Remarks

If parameter 1 is true, the camera's cooler set point can be adjusted and the camera can return the cooling power level. If false, the camera either uses open-loop cooling or does not have the ability to adjust temperature from software.

CanPulseGuide

Property

int QSICamera.get_CanPulseGuide (bool*)

Syntax

results = cam.get_CanPulseGuide(&canGuide);

Exceptions

None

Examples:

```
bool canGuide;
int results;
results = cam.get_CanPulseGuide(&canGuide);
if (canGuide)
{
   results = cam.PulseGuide (guideWest, 10);
}
```

Remarks

Returns true in parameter 1 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

int QSICamera.get_CanSetCCDTemperature (bool*)

Syntax

results = cam.get_CanSetCCDTemperature(&canSetTemp);

Exceptions

None

Examples:

```
bool canSetTemp;
int results;
results = cam.get_CanSetCCDTemperature(&canSetTemp);
if (canSetTemp)
{
   cam.put_SetCCDTemperature (-10.7);
}
```

Remarks

If parameter 1 is 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.

CanSetGain

Property

int QSICamera.get_CanSetGain (bool*)

Syntax

int result =cam.get_CanSetGain(&canSetGain);

Exceptions

If exceptions are enabled, throws an exception if the value is not known.

Examples:

```
try
{
   bool canSetGain;
   cam.get_CanSetGain(&canSetGain);
   if (canAsymBin)
   {
       cam.put_CameraGain(CameraGainHigh);
   }
}
catch (std::runtime_error& err)
{
}
```

Remarks

CanStopExposure

Property

int QSICamera.get_CanStopExposure (bool*)

Syntax

results = cam.get_CanStopExposure(&canStop);

Exceptions

If exceptions are enabled, throws an exception if not supported.

Examples:

```
bool canStop;
cam.get_CanStopExposure(&canStop);
if (canStop)
{
   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 in parameter 1 if StopExposure is available, false if not.

CCDTemperature

Property

int QSICamera.get_CCDTemperature (double)

Syntax

results = cam.get_CCDTemperature(&temp);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
double temp;
int results;
results = cam.get_CCDTemperature( &temp );
```

Remarks

Returns the current CCD temperature in degrees Celsius in parameter 1. Only valid if CanSetCCDTemperature is true.

Connected

Property

int QSICamera.put_Connected (bool*)

int QSICamera.get_Connected(bool)

Syntax

result = cam.put_Connected(true);

result = cam.get_Connected (&connected);

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool connected;
int result;
result = cam.get_Connected(&connected);
if (result == 0 && connected == false)
   cam.put_Connected(true);
```

Remarks

Controls the link between the driver and the camera. Set parameter 1 to true to enable the link. Set false to disable the link (this does not switch off the cooler). You can also read the property using the get_ method 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

int QSICamera.put_CoolerOn (bool)

int QSICamera.get_CoolerOn(bool*)

Syntax

result = cam.put_CoolerOn(true);

result = cam.get_CoolerOn(*coolerOn);

Exceptions

If exceptions are enabled, throws an exception if not supported.

Examples:

```
bool coolerOn;
int result;
result = cam.get_CoolerOn(&coolerON);
if (result == 0 && coolerOn == false)
   cam.put_CoolerOn (true);
```

Remarks

Turns on and off the camera cooler, and returns the current on/off state.

CoolerPower

Property

int QSICamera.get_CoolerPower (double*)

Syntax

results = cam.get_CoolerPower(&power);

Exceptions

If exceptions are enabled, throws an exception if not supported by the camera.

Examples:

```
bool canGetPower;
int results;
results = cam.get_CanGetCoolerPower(&canGetPower);
if (canGetPower)
{
   double power;
   cam.get_CoolerPower(&power);
}
```

Remarks

Returns the present cooler power level, in percent in parameter 1. Returns zero if CoolerOn is false.

Description

Property

int QSICamera.Description (std::string&)

Syntax

int results = cam.get_Description(desc);

Exceptions

If exceptions are enabled, throws an exception if description unavailable.

Examples:

```
try
{
   std::string desc;
   cam.get_Description(desc);
}
catch (std::runtime_error& err)
{
   // No Description available from this device
}
```

Remarks

Returns a description of the camera model, such as manufacturer and model number in parameter 1.

DriverInfo

Property

int QSICamera.get_DriverInfo (std::string&)

Syntax

results = cam.get_DriverInfo(info);

Exceptions

If exceptions are enabled, throws an exception if description unavailable.

Examples:

```
try
{
   string info;
   cam.DriverInfo(info);
}
catch (std::runtime_error& err)
{

}
```

Remarks

Returns revision information of the loaded driver in parameter 1.

ElectronsPerADU

Property

int QSICamera.get_ElectronsPerADU (double*)

Syntax

int reault = cam.get_ElectronsPerADU(&eADU);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
try
{
   double eADU;
   cam.get_ElectroncsPerADU(&eADU);
}
catch (std::runtime_error& err)
{
   // ElectronsPerADU not available from this device
}
```

Remarks

Returns the gain of the camera in photoelectrons per A/D unit in parameter 1.

EnableShutterStatusOutput

Property

int QSICamera.put_EnableShutterStatusOutput (bool*)

int QSICamera.get_EnableShutterStatusOutput (bool)

Syntax

results = cam.put_EnableShutterStatusOutput (true);

results = cam.get_EnableShutterStatusOutput (&enabled);

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
int results;
results = cam.get_EnableShutterStatusOutput(&enabled);
if (results == 0 && enabled == false)
   cam.put_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 "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 at power off.

FanMode

Property

int QSICamera.put_FanMode (enum FanMode)

int QSICamera.get_FanMode (enum & FanMode)

Syntax

results = cam.put_FanMode(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 .QSIConfig file and will be maintained between camera connections.

Examples:

```
try
{
   cam.put_FanMode(fanQuiet);
}
catch (std::runtime_error& err)
{

}
```

Remarks

Controls the speed of the camera's cooling fans.

Symbolic Constants

The symbolic values for FanMode 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

int QSICamera.get_FullWellCapacity (double*)

Syntax

int results = cam.get_FullWellCapacity(&fwc);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
try
{
   double fwc;
   cam.get_FullWellCapacity(&fwc);
}
catch (std::runtime_error& err)
{
   // FullWellCapacity not available from this device
}
```

Remarks

Returns the full well capacity of the camera in electrons, at the current camera settings in parameter 1.

HasFilterWheel

Property

int QSICamera.get_HasFilterWheel (bool*)

Syntax

int result = cam.get_HasFilterWheel(hasFilters);

Exceptions

If exceptions are enabled, throws an exception if no camera is connected.

Examples:

```
int results;
bool hasFilters;
results = cam.get_HasFilterWheel(&hasFilters);
if (results == 0 && hasFilters)
{
    cam.put_Position(2);
}
```

Remarks

If parameter 1 is true, the camera has an internal filter wheel. If false, the camera does not have an internal filter wheel.

HasShutter

Property

int QSICamera.get_HasShutter (bool*)

Syntax

int result = cam.get_HasShutter(&hasShutter);

Exceptions

If exceptions are enabled, throws an exception if no camera is connected.

Examples:

```
int results;
bool hasShutter;
results = cam.get_HasShutter(&hasShutter);
if (hasShutter)
{

}
```

Remarks

If parameter 1 is true, the camera has a 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

int QSICamera.get_HeatSinkTemperature (double*)

Syntax

results = cam.get_HeatSinkTemperature(temp);

Exceptions

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

Examples:

```
double temp;
results = cam.get_HeatSinkTemperature(&temp);
```

Remarks

Returns the current heat sink temperature (The ambient air temperature as it enters the fans on the heatsink) in degrees Celsius in parameter 1. Only valid if CanSetCCDTemperature is true. Will return 0 for cameras without heat sink sensors.

HostTimedExposure

Property

int QSICamera.put_HostTimedExposure(bool)

Syntax

```
int result = cam.put_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:

```
//
// example with UseStructedExceptions == false
//

int result = cam.put_HostTimedExposure(true);
if (result != 0)
{
   //handle error//
}
```

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

int QSICamera.get_ImageArray (unsigned short*)

--or--

int QSICamera.get_ImageArray(double*)

Syntax

results = cam.get_ImageArray((unsigned short*)image);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
result = cam.StartExposure(.05, true);

bool imageReady = false;
result = cam.get_ImageReady(&imageReady);
while(!imageReady)
{
      usleep(5000);
      result = cam.get_ImageReady(&imageReady);
}
int x,y,z;
result = cam.get_ImageArraySize(x, y, z);
unsigned short* image = new unsigned short[x * y];
result = cam.get_ImageArray((void*)image);
```

Remarks

Fills an array of unsigned shorts or doubles (depending on the arguments type) of size NumX * NumY containing the pixel values from the last exposure. The application must inspect the get_ImageArraySize 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.

The caller must test the property ImageReady and received a true result prior to using this method, to insure that the transfer of the image from the camera is complete. Once the image is complete, the caller should first call get_ImageArraySize to determine the required size of the image array.

ImageArraySize

Property

int QSICamera.get_ImageArraySize (int& xsize, int& ysize, int& pixelsize)

Syntax

results = cam.get_ImageArraySize(xsize, ysize, pixelsize);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
result = cam.StartExposure(.05, true);

bool imageReady = false;
result = cam.get_ImageReady(&imageReady);
while(!imageReady)
{
      usleep(5000);
      result = cam.get_ImageReady(&imageReady);
}
int x,y,z;
result = cam.get_ImageArraySize(x, y, z);
unsigned short* image = new unsigned short[x * y];
result = cam.get_ImageArray((void*)image);
```

Remarks

Returns the dimensions of the pending image in parameters 1, 2, and 3. Use this call to provide size information when creating an array for transfer of the image data. The xsize and ysize parameters specify the dimensions of the pending image in pixels units. The pixel size parameter is the size of each pixel value in bytes. 500 series cameras always return a value of 2 for the pixel size.

ImageReady

Property

int QSICamera.get_ImageReady (bool*)

Syntax

results = cam.get_ImageReady(&imageReady);

Exceptions

If exceptions are enabled, throws an exception if hardware or communications link error has occurred.

Examples:

```
result = cam.StartExposure(.05, true);

bool imageReady = false;
result = cam.get_ImageReady(&imageReady);
while(!imageReady)
{
      usleep(5000);
      result = cam.get_ImageReady(&imageReady);
}
int x,y,z;
result = cam.get_ImageArraySize(x, y, z);
unsigned short* image = new unsigned short[x * y];
result = cam.get_ImageArray((void*)image);
```

Remarks

If parameter 1 is 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 or error.

IsMainCamera

Property

int QSICamera.put_IsMainCamera (bool)

int QSICamera.get_IsMainCamera(bool*)

Syntax

result = cam.put_IsMainCamera(true);

result = cam.get_IsMainCamera(&isMain);

Exceptions

If exceptions are enabled, throws an exception if the camera is connected when attempting to set the property.

Examples:

```
try
{
   bool isMain;
   cam.get_IsMainCamera(isMain);
   if (!isMain)
   {
         cam.put_IsMainCamera(true);
   }
}
catch (std::runtime_error& err) {}
```

Remarks

If parameter 1 is 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 (i.e. when setting this property).

IsPulseGuiding

Property

int QSICamera.get_IsPulseGuiding (bool*)

Syntax

results = cam.get_IsPulseGuiding(&guiding);

Exceptions

If exceptions are enabled, throws an exception if hardware or communications link error has occurred.

Examples:

```
bool guiding;
results = cam.get_IsPulseGuiding(&guiding)
if (results == 0 && guiding)
{

}
```

Remarks

If parameter 1 is true, pulse guiding is in progress.

LastError

Property

int QSICamera.get_LastError (std::string&)

Syntax

results = cam.get_LastError(error);

Exceptions

If no error has occurred, the string "0x00000000: No Error" is returned.

Examples:

```
std::string error;
cam.get_LastError(error);
```

Remarks

Reports the last error condition reported by the camera hardware or communications link. The string contains an error code and an error message.

LastExposureDuration

Property

int QSICamera.get_LastExposureDuration (double*)

Syntax

result = cam.get_LastExposureDuration(time);

Exceptions

If exceptions are enabled, throws an exception if not supported or no exposure has been taken.

Examples:

```
double time;
int result;
try
{
   result = cam.get_LastExposureDuration(&time);
}
catch( Exception err){}
```

Remarks

Reports the actual exposure duration in seconds (i.e. shutter open time) in parameter 1. This may differ from the exposure time requested is StopExposure() was called (and is available on the camera).

LastExposureStartTime

Property

int QSICamera.get_LastExposureStartTime (std::string&)

Syntax

result = cam.get_LastExposureStartTime(info);

Exceptions

If exceptions are enabled, throws an exception if not supported or no exposure has been taken.

Examples:

```
std:string time;
try
{
   cam.get_LastExposureStartTime(time);
}
catch(std::runtime_error& err){}
```

Remarks

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

LEDEnabled

Property

int QSICamera.put_LEDEnabled (bool)

int QSICamera.get_LEDEnabled (bool &)

Syntax

result = cam.get_LEDEnabled(enabled);

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
results = cam.get_LEDEnabled(enabled);
if (enabled == false)
   cam.put_LEDEnabled( true );
```

Remarks

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

ManualShutterMode

Property

int CCDCamera.put_ManualShutterMode (bool)

int CCDCamera.get_ManualShutterMode (bool *)

Syntax

result = CCDCamera.put_ManualShutterMode(true);

Exceptions

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

Examples:

```
bool enabled;
results = cam.get_ManualShutterMode(&enabled);
if (enabled == false)
   cam.put_ManualShutterMode( true );
```

Remarks

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

ManualShutterOpen

Property

int CCDCamera.put_ManualShutterOpen (bool)

Syntax

result = CCDCamera.put_ManualShutterOpen(true);

Exceptions

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

Examples:

```
results = cam.put_ManualShutterOpen(true); //Open the shutter
results = cam.put_ManualShutterOpen(false); //Close the shutter
```

Remarks

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

MaxADU

Property

int QSICamera.get_MaxADU (long*)

Syntax

result = cam.get_MaxADU(&adu);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
try
{
   long adu;
   cam.get_MaxADU(&adu);
}
catch (std::runtime_error& err) {}
```

Remarks

Reports the maximum ADU value the camera can produce in parameter 1.

MaxBinX

Property

int QSICamera.get_MaxBinX (short*)

Syntax

result = cam.get_MaxBinX(&maxBinX);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
short maxBinX;
result = cam.get_MaxBinX(&maxBinX);
```

Remarks

If AsymmetricBinning = false, returns the maximum allowed binning factor in parameter 1. If AsymmetricBinning = true, returns the maximum allowed binning factor for the X axis in parameter 1.

MaxBinY

Property

int QSICamera.get_MaxBinY (short*)

Syntax

result = cam.get_MaxBinY(&maxBinY);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
short maxBinY;
result = cam.get_MaxBinY(&maxBinY);
```

Remarks

If AsymmetricBinning = false, parameter 1 equals MaxBinX. If AsymmetricBinning = true, returns the maximum allowed binning factor for the Y axis in parameter 1.

MaxExposureTime

Property

int QSICamera.get_MaxExposureTime (double *)

Syntax

result = cam.get_MaxExposureTime(&maxExp)

Exceptions

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

Examples:

```
double maxExp;
result = cam.get_MaxExposureTime(&maxExp);
```

Remarks

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

MinExposureTime

Property

int QSICamera.get MinExposureTime (double *)

Syntax

result = cam.get_MinExposureTime(&minExp);

Exceptions

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

Examples::

```
double minExp;
result = cam.get_MinExposureTime(&minExp;
```

Remarks

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

ModelNumber

Property

int QSICamera.get_ModelNumber (std::string&)

Syntax

result = cam.get_ModelNumber(model);

Exceptions

If exceptions are enabled, throws an exception if description unavailable

Examples:

```
try
{
   std::string model;
   cam.get_ModelNumber(model);
}
catch (std::runtime_error& err){}
```

Remarks

Returns the model number of the camera in parameter 1.

Name

Property

int QSICamera.get_Name (std::string&)

Syntax

result = cam.get_Name(name);

Exceptions

If exceptions are enabled, throws an exception if description unavailable

Examples:

```
try
{
   std::string name;
   cam.get_Name(name);
}
catch (std::runtime_error& err) {}
```

Remarks

Returns the model name of the camera in parameter 1.

Named Filter Wheels

Properties

```
int QSICamera.get_HasFilterWheelTrim(bool* pVal);
int QSICamera.get_FilterPositionTrim( std::vector<short> * pVal);
int QSICamera.put_FilterPositionTrim( std::vector<short>);
int QSICamera.get_FilterWheelNames( std::vector<std::string> * pVal);
int QSICamera.get_SelectedFilterWheel(std::string * pVal);
int QSICamera.put_SelectedFilterWheel(std::string newVal);
int QSICamera.NewFilterWheel(std::string Name);
int QSICamera.DeleteFilterWheel(std::string Name);
```

Syntax

```
result =  cam.get_HasFilterWheelTrim(&bHasTrim);
result =  cam.get_FilterPositionTrim( &vTrimVal);
result =  cam.put_FilterPositionTrim( vTrimVal );
result =  cam.get_FilterWheelNames( &vNames);
result =  cam.get_SelectedFilterWheel(&strName);
result =  cam. put_SelectedFilterWheel("CYMKWheel");
result =  cam.NewFilterWheel("LRGBWHEEL");
result =  cam.DeleteFilterWheel("CYMKWheel");
```

Exceptions

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

Examples:

```
cam.NewFilterWheel("WheelOne");
cam.NewFilterWheel("WheelTwo");
cam.NewFilterWheel("WheelThree");

std::string WheelName;
cam.get_SelectedFilterWheel(& WheelName);
cam.put_SelectedFilterWheel("WheelOne");
std::vector<std::string> Wheels;
cam.get_FilterWheelNames( & Wheels);
bool HasTrim = false;
cam.get_HasFilterWheelTrim(& HasTrim);
if (HasTrim)
{
      std::vector<short> Trims;
      cam.get_FilterPositionTrim( & Trims);
      Trims[0] = 100;
      cam.put_FilterPositionTrim( Trims );

}
// Allocate arrays for filter names and offsets
names = new std::string[filters];
offsets = new long[filters];
cam.get_Names(names);
cam.get_FocusOffset(offsets);
// change the name and offset of filter 1
names[1] = "UV";
offsets[1] = 100L;
// Save the new names and offsets
cam.put_Names(names);
cam.put_FocusOffset(offsets);
cam.DeleteFilterWheel("WheelOne");
cam.DeleteFilterWheel("WheelTwo");
cam.DeleteFilterWheel("WheelThree");
```

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.

NumX

Property

int QSICamera.put_NumX (long)

int QSICamera.get_NumX (long*)

Syntax

result = cam.put_NumX(100);

result = cam.get_NumX(&numX);

Exceptions

None

Examples:

```
long numX;
int result;
result = cam.get_NumX(&numX);
```

Remarks

Sets the sub-frame width. Also returns the current value in parameter 1. If binning is active, value is in binned pixels. No error check is performed when the value is set. Defaults to CameraXSize.

NumY

Property

int QSICamera.put_NumY (long)

int QSICamera.get_NumY (long*)

Syntax

result = cam.put_NumY(100);

result = cam.get_NumY(&numY);

Exceptions

None

Examples:

```
long numY;
int result;
result = cam.get_NumY(&numY);
```

Remarks

Sets the sub-frame height. Also returns the current value in parameter 1. If binning is active, value is in binned pixels. No error check is performed when the value is set. Defaults to CameraYSize.

PixelMask

Methods / Classes

```
int QSICamera .get_PixelMask(std::vector<pixel> *)
int QSICamera .put_PixelMask(std::vector<pixel>)
int QSICamera .get_MaskPixels (bool *)
int QSICamera.put_MaskPixels(bool)
class Pixel
{
public:
        Pixel(int x, int y){ this->x = x; this->y = y;}
        ~Pixel(void) { }
        int x;
        int y;
};
```

Syntax

```
result = cam.get_PixelMask(*vPixels);
result = cam.put_PixelMask(vPixel);
result = cam.MaskPixels(true);
```

Exceptions

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

Example

```
bool enabled;
std::vector<Pixel> map;
// get old values to save
cam.get_MaskPixels(&enabled);
cam.get_PixelMask(&map);
// create new settings
std::vector<Pixel> newmap;
newmap.push_back(Pixel(234,678));
newmap.push_back(Pixel(233,244));
cam.put_PixelMask(newmap);
cam.put_MaskPixels(true);
// Read back the new settings
bool isenabled;
std::vector<Pixel> checkmap;
cam.get_MaskPixels(&isenabled);
cam.get_PixelMask(&checkmap);
std::cout << "Pixel mapping :" << isenabled << "\n";
std::cout << "Pixels: \n";
std::vector<Pixel>::iterator vi;
for (vi = checkmap.begin(); vi != checkmap.end(); vi++)
{
std::cout << "(" << (*vi).x << "," << (*vi).y << ")\n";
}
// now reset to the orginal settings
cam.put_PixelMask(map);
cam.put_MaskPixels(enabled);
```

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.

PixelSizeX

Property

int QSICamera.get_PixelSizeX (double*)

Syntax

result = cam.get_PixelSizeX(&x);

Exceptions

If exceptions are enabled, throws exception if data unavailable.

Examples:

```
try
{
   double x;
   cam.get_PixelSizeX(&x);
}
catch (std::runtime_error& err){}
```

Remarks

Returns the width of the CCD chip pixels in microns in parameter 1, as provided by the camera driver.

PixelSizeY

Property

int QSICamera.get_PixelSizeY (double*)

Syntax

result = cam.get_PixelSizeY(&y);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
try
{
   double y;
   cam.get_PixelSizeY(&y);
}
catch (std::runtime_error& err) {}
```

Remarks

Returns the height of the CCD chip pixels in microns in parameter 1, as provided by the camera driver.

PowerOfTwoBinning

Property

int QSICamera.get_PowerOfTwoBinning (bool*)

Syntax

result = cam.get_PowerOfTwoBinning(&p2bin);

Exceptions

None

Examples:

```
bool p2bin;
result = cam.get_PowerOfTwoBinning(&p2bin);
if (result ==0 && p2bin)
{

}
```

Remarks

If parameter 1 is true, the camera bins in powers of two. If false, the camera bins in increments of one.

PreExposureFlush

Property

int QSICamera.put_PreExposureFlush (enum PreExposureFlush)

int QSICamera.get_PreExposureFlush(enum PreExposureFlush *)

Syntax

```
int result = cam.put_PreExposureFlush(QSICamera::FlushNormal );
```

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 .QSIConfig file and will be maintained between camera connections.

Examples:

QSICamera cam;

```
//
// example with UseStructedExceptions == false
//
int result = cam.put_PreExposureFlush(QSICamera::FlushNormal);
if (result != 0)
{
   //handle 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	one flush cycle
Normal	two flush cycles
Aggressive	four flush cycles
Very Aggressive	eight flush cycles.

Symbolic Constants

The symbolic values for PreExposureFlush are:

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

ReadoutSpeed

Property

int QSICamera.put_ReadoutSpeed (enum)

int QSICamera.get_ReadoutSpeed (enum&)

Syntax

result = cam.put_ReadoutSpeed(QSICamera::FastReadout);

Exceptions

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

Examples:

```
cam.put_ReadoutSpeed( QSICamera::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.

Enum for ReadoutSpeed, as defined in qsiapi.h:

```
enum ReadoutSpeed
{
	HighImageQuality = 0,
	FastReadout = 1
};
```

SelectCamera

Property

int QSICamera.put_SelectCamera (std::string)

int QSICamera.get_ SelectCamera (std::string *)

Syntax

result = cam.put_ SelectCamera (“00502884”);

result = cam.get_ SelectCamera (&camera);

Exceptions

.

Examples:

```
std::string camera;
int result;
result = cam.put_ SelectCamera (“00502884”);
result = cam.get_ SelectCamera (&camera);
```

Remarks

When multiple cameras are connected to the USB bus, this method selects which camera will be opened by the put_Connected(true) method. The QSICamera object must be in the disconnected state to change the camera selection, otherwise an exception will be thrown.

SerialNumber

Property

int QSICamera.get_SerialNumber (std::string&)

Syntax

result = cam.get_SerialNumber(num);

Exceptions

If exceptions are enabled, throws an exception if description unavailable

Examples:

```
try
{
   std::string num;
   cam.get_SerialNumber(num);
}
catch (std::runtime_error& err) {}
```

Remarks

Returns the serial number of the camera in parameter 1.

SetCCDTemperature

Property

int QSICamera.put_SetCCDTemperature (double)

int QSICamera.get_SetCCDTemperature (double*)

Syntax

result = cam.put_SetCCDTemperature (-10.0);

result = cam.get_SetCCDTemperature(&temp);

Exceptions

If exceptions are enabled, throws an exception if command not successful. If exceptions are enabled, throws an exception if CanSetCCDTemperature is false.

Examples:

```
try
{
   double temp;
   cam.get_SetCCDTemperature(&temp
   cam.put_SetCCDTemperature ( 10.0);
}
catch (std::runtime_error& err) {}
```

Remarks

The put_ method sets the camera cooler set-point in degrees Celsius in parameter 1, and the get_ method returns the current set-point in parameter 1.

ShutterMode

No longer supported. See ShutterPriority.

ShutterPriority

Property

int QSICamera.put_ShutterPriority(enum ShutterPriority)

int QSICamera.get_ShutterPriority(enum Shutter Priority *)

Syntax

```
int result =
cam.put_ShutterPriority(QSICamera::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 .QSIConfig file and will be maintained between camera connections.

Examples:

QSICamera cam;

```
//
// example with UseStructedExceptions == false
//
int result =
cam.put_ShutterPriority(QSICamera::ShutterPriorityMechanical);
if (result != 0)
{
   //handle 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 ShutterPriority are:

Constant	Value
ShutterPriorityMechanical	0
ShutterPriorityElectronic	1

SoundEnabled

Property

int QSICamera.put_SoundEnabled (bool)

int QSICamera.get_SoundEnabled (bool &)

Syntax

result = cam.put_SoundEnabled (true);

Exceptions

If exceptions are enabled, throws an exception if unsuccessful.

Examples:

```
bool enabled;
int results = cam.get_SoundEnabled(enabled);
if (enabled == false)
   cam.put_SoundEnabled( true );
```

Remarks

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

StartX

Property

int QSICamera.put_StartX (long)

int QSICamera.get_StartX(long*)

Syntax

result = cam.put_StartX (0);

result = cam.get_StartX(&x);

Exceptions

None

Examples:

```
long x;
int result;
result = cam.get_StartX(&x);
result = cam.put_StartX (0);
```

Remarks

The put_ method sets the sub-frame start position for the X axis (0 based) in parameter 1. The get_ method returns the current value in parameter 1. If binning is active, value is in binned pixels.

StartY

Property

int QSICamera.put_StartY(long)

int QSICamera.get_StartY(long*)

Syntax

result = cam.put_StartY (0);

result = cam.get_StartY(&y);

Exceptions

None

Examples:

```
long y;
int result;
result = cam.get_StartY(&y);
result = cam.put_StartY (0);
```

Remarks

The put_ method sets the sub-frame start position for the Y axis (0 based) in parameter 1. The set_ method returns the current value in parameter 1. If binning is active, value is in binned pixels.

AbortExposure

Method

int QSICamera.AbortExposure ()

Syntax

result = cam.AbortExposure ();

Parameters

None.

Returns

Nothing

Exceptions

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

Examples:

```
try
{
   cam.AbortExposure ();
}
catch (std::runtime_error& err)
{

}
```

Remarks

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

GetPixelMask / SetPixelMask

Method

CCDCamera.get_PixelMask(std::Vector<Pixel>* Pixels)
CCDCamera.put_PixelMask(std::Vector<Pixel> Pixels)

Syntax

CCDCamera.get_PixelMask(&x, &y);
CCDCamera.Sput_PixelMask(x, y);

Exceptions

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

Example:

C++:

```
// Demonstrate Pixel masking
bool enabled;
std::vector<Pixel> map;
// get old values to save
cam.get_MaskPixels(&enabled);
cam.get_PixelMask(&map);
// create new settings
std::vector<Pixel> newmap;
newmap.push_back(Pixel(234,678));
newmap.push_back(Pixel(233,244));
cam.put_PixelMask(newmap);
cam.put_MaskPixels(true);
// Read back the new settings
bool isenabled;
std::vector<Pixel> checkmap;
cam.get_MaskPixels(&isenabled);
cam.get_PixelMask(&checkmap);
std::cout << "Pixel mapping :" << isenabled << "\n";
std::cout << "Pixels: \n";
std::vector<Pixel>::iterator vi;
for (vi = checkmap.begin(); vi != checkmap.end(); vi++)
{
std::cout << "(" << (*vi).x << "," << (*vi).y << ")\n";
}
// now reset to the orginal settings
cam.put_PixelMask(map);
cam.put_MaskPixels(enabled);
```

Remarks

The QSI camera driver provides the capability to individually mask pixels, replacing the image 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 of type Pixel, with addresses starting at location (0,0).

Use the get_MaskPixels and set_MaskPixel 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.

PulseGuide

Method

int QSICamera.PulseGuide (QSICamera;:GuideDirections, long)

Syntax

result = cam.PulseGuide (QSICamera;:guideNorth, 10);

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

If exceptions are enabled, throws an exception if PulseGuide command is unsupported or the command is unsuccessful.

Examples:

```
try
{
   cam.PulseGuide (QSICamera::guideWest, 10);
}
catch (std::runtime_error& err)
{

}
```

Remarks

Use the IsPulseGuiding property to detect when all moves have completed.

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.

StartExposure

Method

int QSICamera.StartExposure (double Duration, bool Light)

Syntax

result = cam.StartExposure (0.50, true);

Parameters

double Duration - Duration of exposure in seconds

bool Light - true for light frame, false for dark frame (ignored if no shutter)

Returns

Nothing.

Exceptions

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

Examples:

```
try
{
   cam.StartExposure(1.00, true);
}
catch (std::runtime_error& err)
{

}
```

Remarks

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

StopExposure

Method

QSICamera.StopExposure ()

Syntax

result = cam.StopExposure ();

Parameters

None.

Returns

Nothing.

Exceptions

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

Examples:

```
try
{
   bool canStop;
   cam.get_CanStopExposure(&canStop);
   if (canStop)
          cam.StopExposure ();
}
catch (std::runtime_error& err){}
```

Remarks

Stops the current exposure, if any. If an exposure is in progress, the readout process is initiated. The user must issue a get_ImageArray to transfer the pending image from the camera.

External Trigger Input

Methods

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

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


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

Syntax

```
result = cam.SetTriggerMode(LongWait, HighToLow);
result = cam.TerminatePendingTrigger();
result = cam.CancelTriggerMode();
```

Exceptions

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

Example:

```
#include "qsiapi.h"
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <cmath>
#include <stdlib.h>

int main(int argc, char** argv)
{
      QSICamera cam;
      try
      {
            cam.put_UseStructuredExceptions(true);
            cam.put_Connected(true);
      }
      catch (std::runtime_error err)
      {
            exit(1);
      }
////////////////////////////////////////////
// Short Wait Trigger Mode
```

```
////////////////////////////////////////////
// Set Short Wait Ext Trigger (4 seconds max), Pos to Neg polarity
////////////////////////////////////////////
      cam.EnableTriggerMode(QSICamera::ShortWait,QSICamera::HighTo
      Low);
      bool imageReady = false;
      try
      {
            cam.StartExposure(0.3, true);
            // Short wait, so this will return
            //    with an image, or will timeout
            cam.get_ImageReady(&imageReady);
            while (!imageReady)
            {
                  sleep(1);
                  cam.get_ImageReady(&imageReady);
            }
            long x;
            cam.get_NumX(&x);
            long y;
            cam.get_NumY(&y);
            long len = x * y;
            unsigned short * pixels = new unsigned short[len];
            cam.get_ImageArray(pixels);
            //Process image
            //Then clean up
            delete [] pixels;
      }
      catch (std::runtime_error err)
      {
            // Timeout comes here
      }
      // turn off external trigger mode
      cam.CancelTriggerMode();
////////////////////////////////////////////////////////
// Long Wait Trigger Mode
////////////////////////////////////////////////////////
      cam.EnableTriggerMode(   QSICamera::LongWait,
                               QSICamera::LowToHigh);
      try
      {
            // Start a long wait exposure
            cam.StartExposure(0.3, true);
            // Sleep for 5 seconds as a demo
            sleep(5);
            // Demo cancelling of the pending trigger
            // This would not normally be done...
            cam.TerminatePendingTrigger();
      }
      catch (std::runtime_error err)
      {
            exit(1);
      }

      // Start a new exposure with long wait trigger.
      //Note that the trigger mode remains
```

```
	//in effect until canceled.
	try
	{
		cam.StartExposure(0.3, true);
		cam.get_ImageReady(&imageReady);
		while (!imageReady)
		{
			sleep(1);
			 cam.get_ImageReady(&imageReady);
		}
		long x;
		cam.get_NumX(&x);
		long y;
		cam.get_NumY(&y);
		long len = x * y;

		unsigned short * pixels = new unsigned short[len];
		cam.get_ImageArray(pixels);
		// Process image here
		// Then clean up
		delete [] pixels;
	}
	catch (std::runtime_error err)
	{
		exit(1);
	}
////////////////////////////////////////////////////////////
// Cancel Trigger Mode
////////////////////////////////////////////////////////////
	cam.CancelTriggerMode();
	cam.put_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.

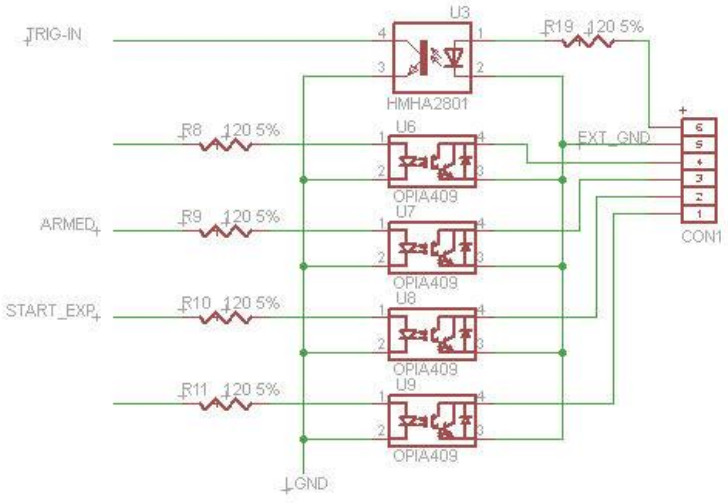
Electrical Interface:

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.

Filter Wheel Properties

FilterCount

Property

int QSICamera.get_FilterCount (int*)

Syntax

int results = cam.get_FilterCount(&fc);

Exceptions

If exceptions are enabled, throws an exception if data unavailable.

Examples:

```
try
{
   double fc;
   cam.get_FilterCount(&fc);
}
catch (std::runtime_error& err)
{
   // Filter count not available from this device
}
```

Remarks

Returns the number of positions in the filter wheel (both empty and occupied) in parameter 1.

Names

Property

int QSICamera.put_Names(std::string[])

int QSICamera.get_Names (std::string[])

Syntax

result = cam.get_Names(names);

result = cam.put_Names (names);

Exceptions

Examples:

```
int result;
int num;

try
{
   cam.get_FilterCount(&num);
   std::string names[num];
   cam.get_Names(names);
   names[1] = "UV";
   cam.put_Names(names);
}
catch (std::runtime_error err) {}
```

Remarks

For each valid slot number (from 0 to N-1), reports the name given to the filter position in parameter 1.

Position

Property

int QSICamera.put_Position (short)

int QSICamera.get_Position(short*)

Syntax

result = cam.put_Position (0);

result = cam.get_Position(&pos);

Exceptions

.

Examples:

```
short pos;
int result;
result = cam.put_Position(2);
result = cam.get_Position(&pos);
```

Remarks

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

FocusOffset

Property

int QSICamera.putFocusOffset(long[])

int QSICamera.get_FocusOffset(long[])

Syntax

result = cam.put_FocusOffset(offsets);

result = cam.get_FocusOffset(offsets);

Exceptions

Examples:

```
int result;
int num;

try
{
   cam.get_FilterCount(&num);
   long  offsets[num];
   cam.get_FocusOffset(offsets);
   offsets[1] = 100L;
   cam.put_FocusOffset(offsets);
}
catch (std::runtime_error err) {}
```

Remarks

For each valid slot number (from 0 to N-1), reports the focus offset for the given filter position in parameter 1.

Appendix A

Notes on usb raw device permissions:

You must use udev to properly set the device permission when the camera is connected via USB.

For example, to give read/write access rights to all users for all QSI USB cameras in Fedora 24:

Create a file in */etc/udev/rules.d* called "*99-qsi.rules*" with the following contents:

KERNEL=="*", SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="eb48", MODE="0666"
KERNEL=="*", SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="eb49", MODE="0666"

After saving the file, execute the following commands:
sudo udevadm control --reload-rules
sudo udevadm trigger

In Ubuntu 6:

In the file /etc/udev/rules.d/40-permisssions.rules

after:
#USB devices (usbfs replacement)
SUBSYSTEM="usb_device", MODE="0664"

Add the following lines:
SUBSYSTEM="usb_device", SYSFS{idVendor}=="0403", SYSFS{idProduct}== "eb48", MODE="0666"
SUBSYSTEM="usb_device", SYSFS{idVendor}=="0403", SYSFS{idProduct}== "eb49", MODE="0666"

Index

Notes