PGR FlyCapture

SINGLE-LENS DIGITAL VIDEO CAMERA SYSTEM

User Manual and API Reference

Version 1.4

Software Warranty

Point Grey Research warrants to the Original Purchaser, for a period of one (1) year from date of purchase that:

1. The diskette on which the Software is furnished and the accompanying documentation are not defective;

2. The Software is properly recorded upon the diskettes enclosed;

3. The documentation is substantially complete and contains all the information Point Grey Research deems necessary to use the Software;

4. The Software functions substantially as described in the documentation.

Point Grey Research, Inc.'s entire liability and the Original Purchaser's exclusive remedy shall be the replacement of any diskette or documentation not meeting these warranties. On such an occasion, a copy of the paid receipt accompanied with the faulty diskette or documentation must be returned to Point Grey Research, Inc. or an authorized dealer.

Point Grey Research, Inc. expressly disclaims and excludes all other warranties, express, implied and statutory, including, but without limitation, warranty of merchantability and fitness for a particular application or purpose. In no event shall Point Grey Research, Inc. be liable to the Original Purchaser or any third party for direct, indirect, incidental, consequential, special or accidental damages, including without limitation damages for business interruption, loss of profits, revenue, data or bodily injury or death.

Hardware Warranty

Point Grey Research Inc. (PGR) warrants to the Original Purchaser that the Camera Module provided with this package is guaranteed to be free from material and manufacturing defects for a period of one (1) year. Should a unit fail during this period, PGR will, at its option, repair or replace the damaged unit. Repaired or replaced units will be covered for the remainder of the original equipment warranty period. This warranty does not apply to units which, after being examined by PGR, have been found to have failed due to customer abuse, mishandling, alteration, improper installation or negligence.

Point Grey Research FlyCapture Software License Agreement

The FlyCapture Software Development Kit (the "Software") is owned and copyrighted by Point Grey Research, Inc. All rights are reserved. The Original Purchaser is granted a license to use the Software subject to the following restrictions and limitations.

1. The license is to the Original Purchaser only, and is nontransferable unless you have received written permission of Point Grey Research, Inc.

2. The Original Purchaser may use the Software only with Point Grey Research, Inc. cameras owned by the Original Purchaser, including but not limited to, Firefly™ or Dragonfly™ Camera Modules.

3. The Original Purchaser may make back-up copies of the Software for his or her own use only, subject to the use limitations of this license.

4. Subject to s.5 below, the Original Purchaser may not engage in, nor permit third parties to engage in, any of the following:

A. Providing or disclosing the Software to third parties.

B. Making alterations or copies of any kind of the Software (except as specifically permitted in s.3 above).

C. Attempting to un-assemble, de-compile or reverse engineer the Software in any way.

D. Granting sublicenses, leases or other rights in the Software to others.

5. Original Purchasers who are Original Equipment Manufacturers may make Derivative Products with the Software. Derivative Products are new software products developed, in whole or in part, using the Software and other Point Grey Research, Inc. products. Point Grey Research, Inc. hereby grants a license to Original Equipment Manufacturers to incorporate and distribute the libraries found in the Software with the Derivative Products. The components of any Derivative Product that contain the Software libraries may only be used with Point Grey Research, Inc. products, or images derived from such products.

5.1 By the distribution of the Software libraries with Derivative Products, Original Purchasers agree to:

A. not permit further redistribution of the Software libraries by end-user customers;

B. include a valid copyright notice on any Derivative Product; and

C. indemnify, hold harmless, and defend Point Grey Research, Inc. from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of any Derivative Product.

Point Grey Research, Inc. reserves the right to terminate this license if there are any violations of its terms or if there is a default committed by the Original Purchaser. Upon termination, for any reason, all copies of the Software must be immediately returned to Point Grey Research, Inc. and the Original Purchaser shall be liable to Point Grey Research, Inc. for any and all damages suffered as a result of the violation or default.

Table of Contents

Table of Figures

Chapter 1: Introduction

This chapter presents the features of the PGR FlyCapture SDK and introduces its capabilities.

Introduction

Included with all PGR IEEE-1394 single-lens digital camera is PGR FlyCapture, which includes a device driver and a software development kit (SDK) designed specifically for PGR 1394 single-lens cameras. The driver and SDK work with these cameras and provides a unified Application Programming Interface (API).

Included with the SDK are a number of C/C++ sample programs that you can compile and run yourself and adapt for your own applications, as well as sample programs that you can run "out of the box". One of the latter is FlyCap, an application that allows you to grab images from and control the settings of your PGR camera.

The package you have purchased comes with:

- a PGR 1394 single-lens camera (see *Compatible PGR IEEE-1394 Cameras* below);
- a 1394 6-pin cable that provides both the 1394 connection to your computer and a power source for your camera;
- a 1394 PCI card that interfaces with the camera and your computer;
- a tripod mounting bracket for attaching the camera to a tripod;
- a PGR FlyCapture CD that contains the PGR FlyCapture SDK, the documentation for the system, and the driver for the camera; and
- either three microlenses (4mm, 6mm, and 8mm focal lengths) or a C/CS-mount lens holder.

Intended Audience

The intended audience for this manual is one who has both experience with 1394 Digital Cameras and has a working knowledge of C/C++ programming (for using the SDK). If you require additional information about the IEEE 1394 standard, please consult http://1394ta.org.

Compatible PGR IEEE-1394 Cameras

PGR FlyCapture is designed to be compatible with the following PGR cameras:

Dragonfly™

The Dragonfly™ is an OEM-style IEEE-1394 board level camera providing maximum control and flexibility for digital imaging applications. It is now available in a new format, which includes an enclosure designed to protect the camera from mechanical hazards and reduce the effects of static discharge. The enclosure comes with a built-in CS mount lens holder, and is available with an IR filter option, suitable for the color cameras.

Scorpion

The Scorpion IEEE-1394 digital camera was designed for demanding imaging applications where both high resolution and high frame rate are necessary. With 1280x960 resolution at 30 frames per second, Scorpion uses the maximum available bus bandwidth without compromising the integrity of the data.

Firefly™

The Firefly™ is a fully digital IEEE-1394 board level camera that provides 24-bit true color images from its 1/3" progressive scan CCD at 30fps.

Firefly2

The Firefly2 is an OEM board-level camera using a 1/4" progressive scan CCD. At a resolution of 640x480, the Firefly2 broadcasts color images up to 30 fps without compression. And it's compact footprint 40x40mm makes the Firefly2 easy to integrate.

Caring for your Camera

Your PGR IEEE 1394 digital camera module is a precisely manufactured device and should be handled with care. Here are some tips on how to care for the device.

- CCD image sensors are easily damaged by static discharge. Before handling be sure to take the following protective measures:
  - Either handle bare handed or use non-chargeable gloves, clothes or material. Also use conductive shoes.
  - Install a conductive mat on the floor or working table to prevent the generation of static electricity.
  - Consult the following knowledge base article for more information: http://www.ptgrey.com/support/kb/details.asp?id=42
- When handling the camera unit, avoid touching the lenses. Fingerprints will affect the quality of the image produced by the device.
- To clean the lenses, use a standard camera lens cleaning kit or a clean dry cotton cloth. Do not apply excessive force.
- To clean the imaging surface of your CCD, follow the steps outlined in http://www.ptgrey.com/support/kb/details.asp?id=66.
- Our cameras are designed for an office environment or laboratory use. Extended exposure to bright sunlight, rain, dusty environments, etc. may cause problems with the electronics and the optics of the system.
- Avoid excessive shaking, dropping or any kind of mishandling of the device.

PGR FlyCapture System Requirements and Recommendations

Hardware Requirements

- Compatible PGR IEEE-1394 Camera
- 1394 PCI card and 1 free PCI slot
- 6-pin 1394 cable
- Intel Pentium or compatible processor
- 128 MB of RAM

Software Requirements

- MS Visual C++ version 6.0 (to run the example code and use the SDK)
- Windows 2000 (Service Pack 4 recommended) or Windows XP (Service Pack 1 recommended)

Hardware Recommendations

- Intel Pentium 4 1.7GHz or compatible processor
- 256 MB of RAM

Chapter 2: Installation

This chapter discusses the setup of the PGR FlyCapture driver and accompanying software.

Installing the PGR FlyCapture System

***NOTE**: You must install the driver and SDK before connecting the camera to your computer, otherwise your computer will not use the correct driver. If this happens, you must manually install the driver. Please consult our on-line knowledge base at http://www.ptgrey.com/support/kb/ for instructions on doing this.*

***NOTE**: Before installing PGR FlyCapture, consult the Release Notes available on your CD or downloadable from our website. The Release Notes sometimes contain important information or warnings regarding installation.*

To install your PGR FlyCapture system and associated camera and hardware you will need to complete the following steps in this order:

1. Install the PCI card into an available PCI slot.
2. Remove any old PGR FlyCapture software.
3. Install the new PGR FlyCapture driver and SDK.
4. Connect the camera to your computer.
5. Test the installation.

Installing the 1394 PCI Card

Installing the 1394 PCI OHCI (Open Host Controller Interface) card is very similar to installing other PCI devices. Upon successful installation of the card, Windows Device Manager will have a "IEEE 1394 Bus host controller" listed (see below). To access Windows Device Manager, right-click *My Computer*, select *Properties*, navigate to the *Hardware* tab and click *Device Manager*.

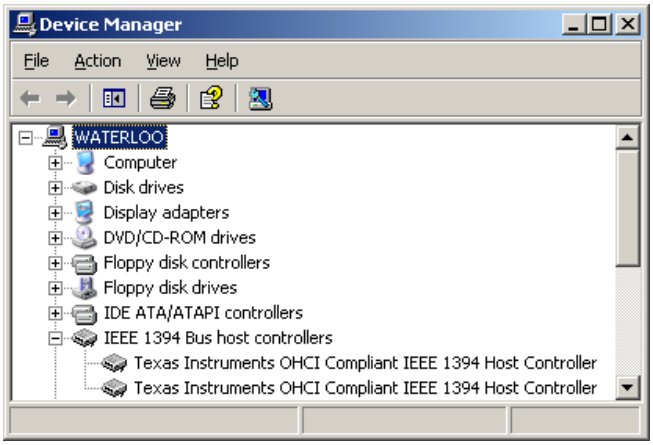


Figure 1: Windows Device Manager listing successfully installed 1394 PCI card

Removing Old PGR FlyCapture Software

If you have previously installed PGR FlyCapture and want to install a newer version, you will need to complete the following steps:

1. Uninstall all cameras from the system by locating the camera in the Windows Device Manager, right-clicking the device, and selecting *Uninstall*.

2. Unplug the camera.

3. Go to the Start Menu and open the Control Panel. Use *Add/Remove Programs* to remove the PGR FlyCapture software from your computer.

Installing the PGR FlyCapture SDK

To install the PGR FlyCapture SDK, you will need to complete the following steps:

1. If the InstallShield Wizard does not automatically run after placing the installation CD in the drive, browse to your CD-ROM directory and run the *setup.exe* file.

2. The InstallShield Wizard splash screen will be launched.

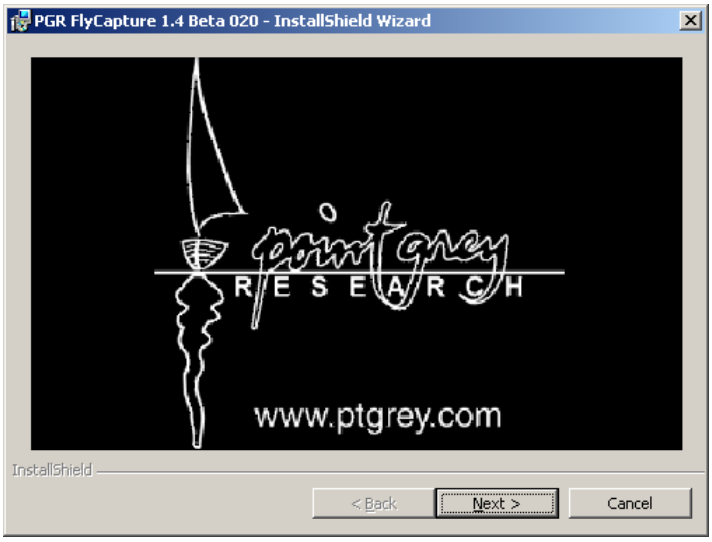
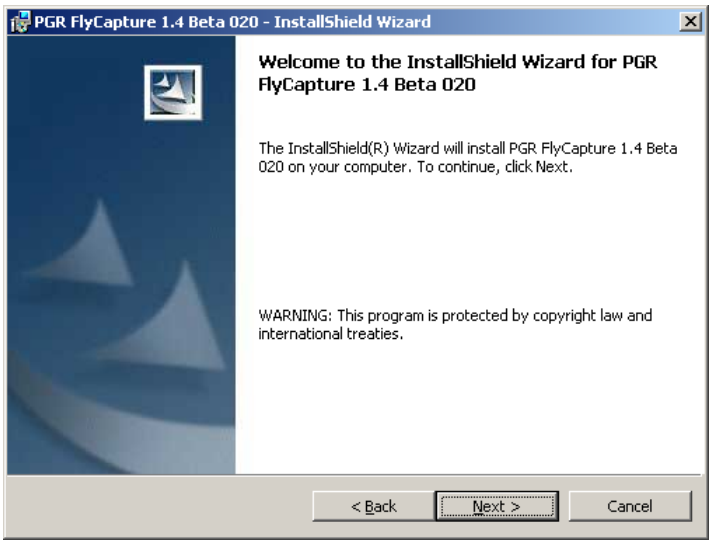
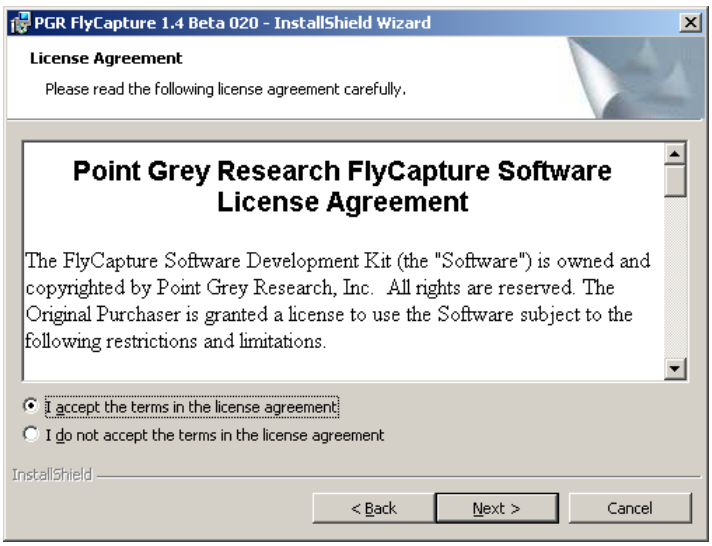


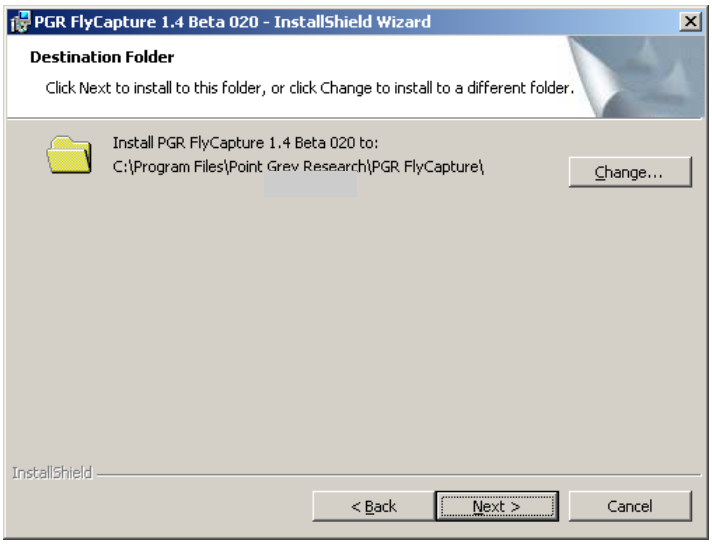
Figure 2: InstallShield Wizard installation screen

3. Follow the InstallShield instructions to click through the splash screen and the software license agreement.





4. The destination to install the SDK will default to a Point Grey Research Inc. directory; you can either click *Next* or click on *Change* to change the destination directory.



5. The install will proceed and you will come to a *Software Installation* dialog box, letting you know that the software has not been signed by Microsoft. Click *Continue Anyway* to proceed with installation. This will not harm your system.

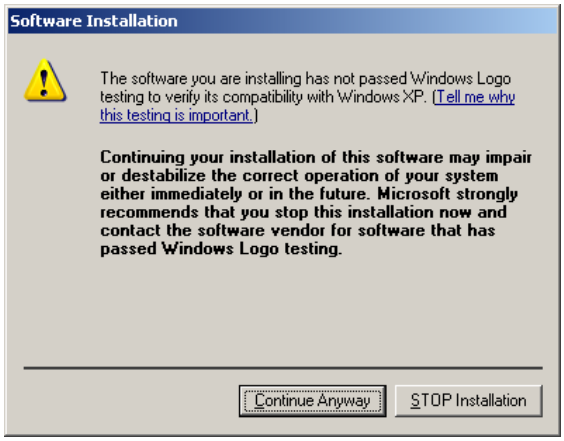


Figure 3: Software Installation screen verifying compatibility with Windows XP

6. Installation will complete and you will receive a "PGRCam driver installation succeeded!" message.

7. Click OK, and you will then be given the option of reading the ReadMe file, which is a set of Release Notes about this version of software.

> ***NOTE:*** *PGR highly recommends reading the Release Notes. They include important information about current bug fixes and new SDK functionality.*

8. After reading the Release Notes, click *Finish* to finalize the SDK installation.

9. Reboot your computer.

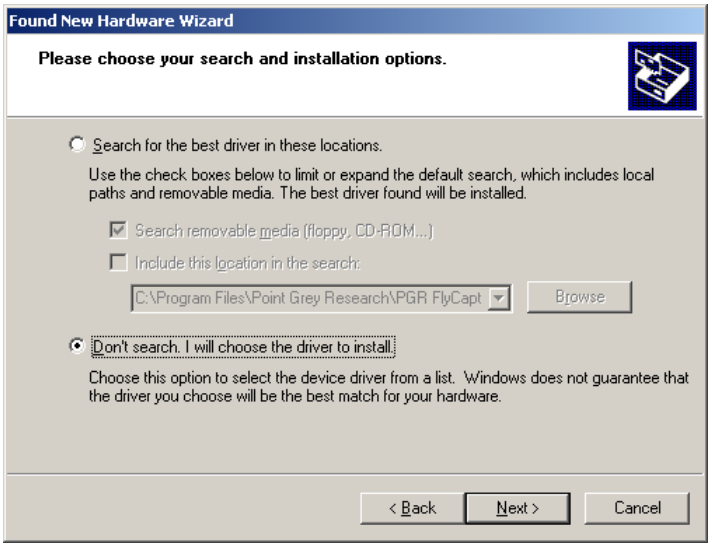
Connect the Camera to Your Computer

1. Connect the 1394 cable to the 1394 PCI card and the camera.

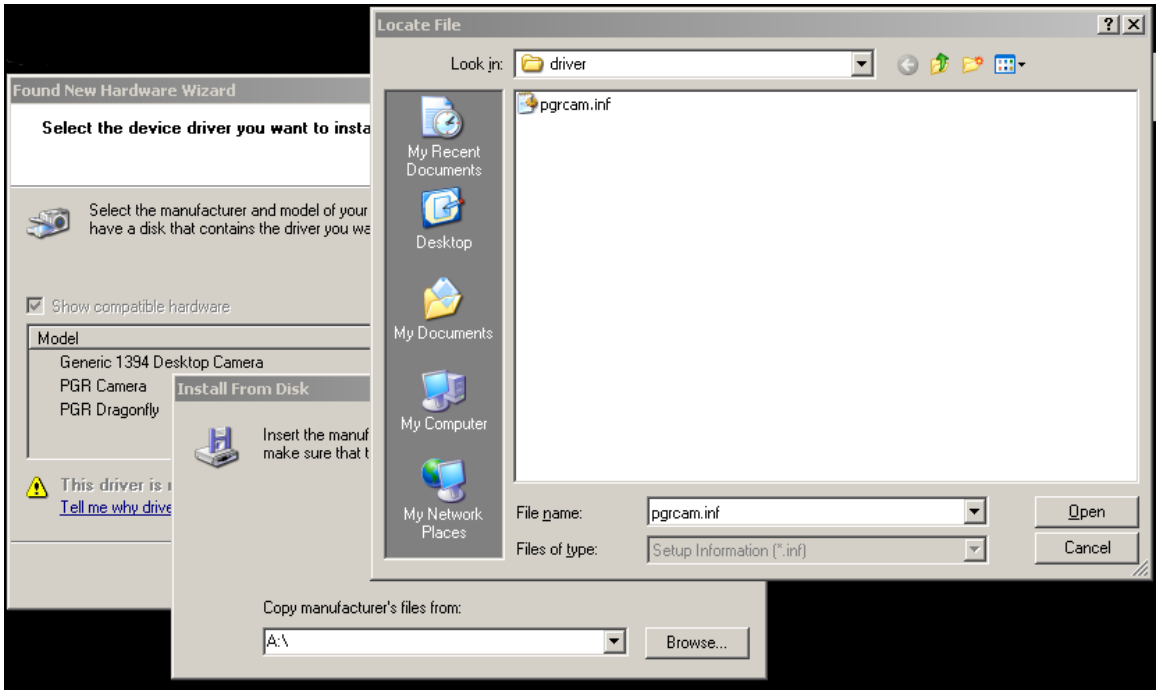
2. The *Found New Hardware Wizard* will appear to help you install the camera driver. Select *Install from a list or specific location (Advanced)* and click *Next*.

Figure 4: Found New Hardware Wizard - advanced (manual) driver installation

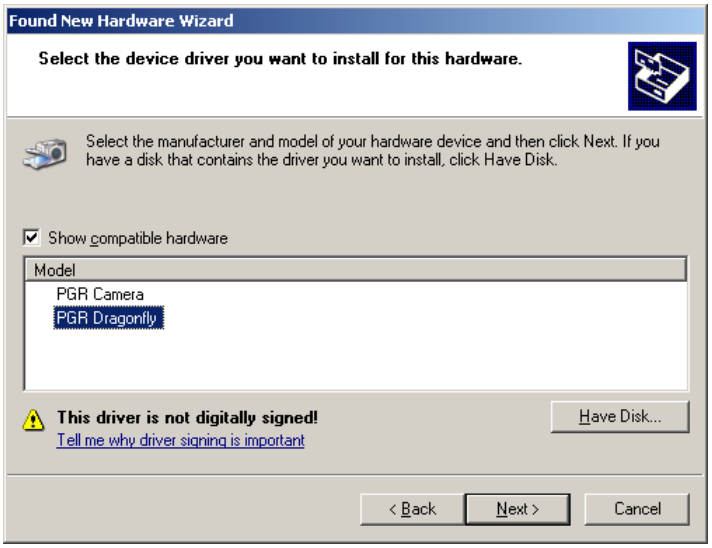
3. Select *Don't search. I will choose the driver to install* and click *Next*.



4. Click the *Have Disk* button and *Browse* to your *C:\Program Files\Point Grey Research\PGR FlyCapture\drivers* folder, then select *pgrcam.inf*.



5. Click *Open*, then click *OK*, then select your camera model from the list and click *Next*.



6. When you are prompted to continue installation, click *Continue Anyway*.

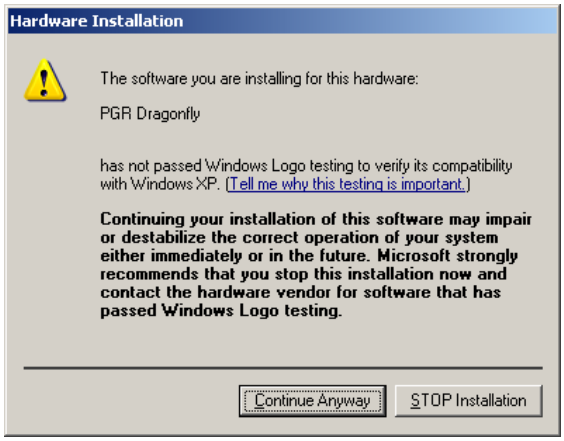


Figure 5: Hardware Installation screen verifying compatibility with Windows XP

7. Click *Finish* to complete the driver installation. Your Device Manager window should look similar to the one below.

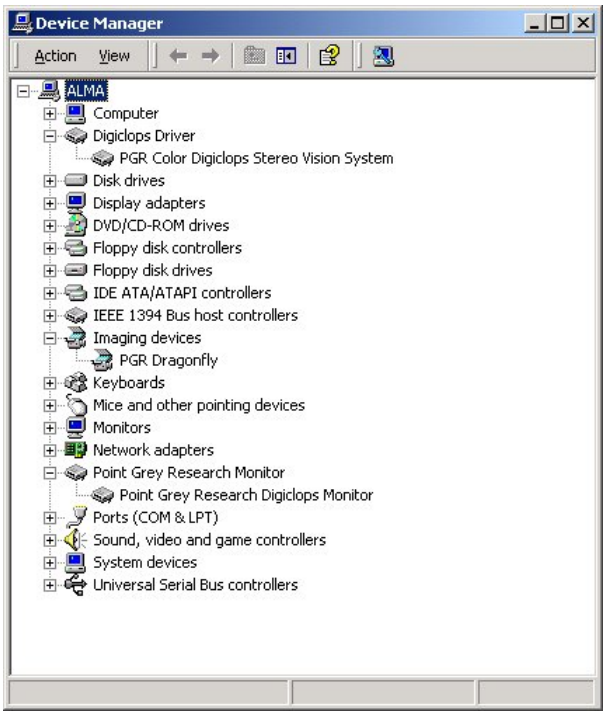


Figure 6: Windows Device Manager showing successful camera driver installation

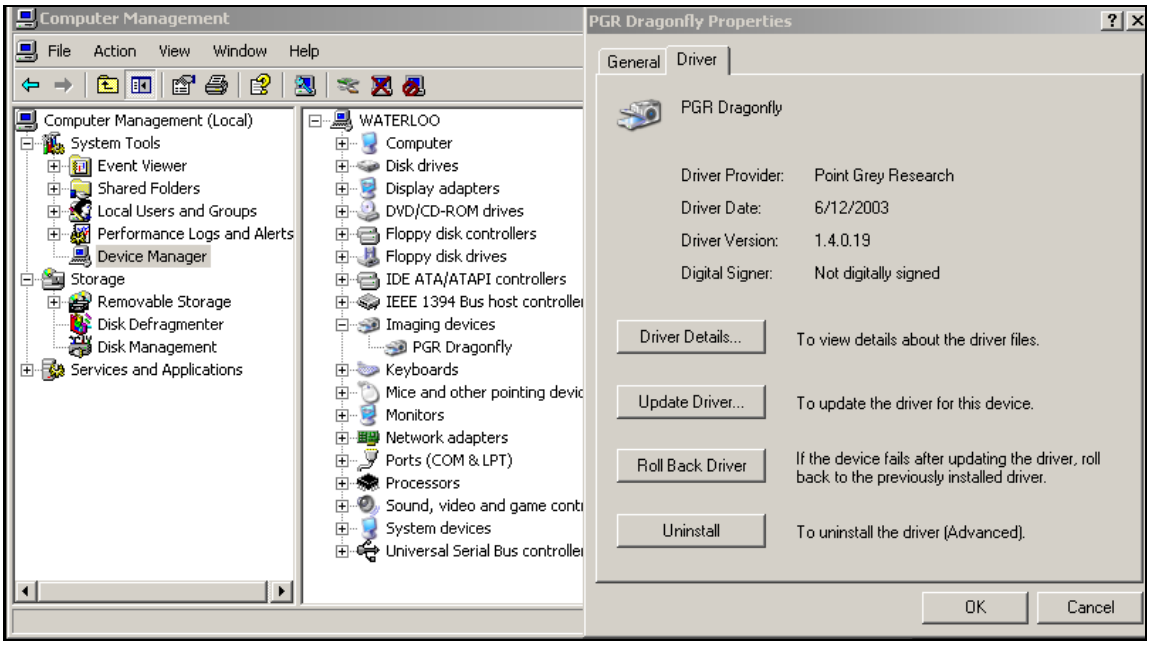
Testing the Installation

To test that the installation was successful, first ensure that the proper driver was installed for the PGR FlyCapture system.

1. Open the *Windows Device Manager* in Windows by right-clicking *My Computer* and selecting *Properties* → *Hardware* → *Device Manager*.

2. Right-click the *Imaging Device* you want to check and select *Properties*.

3. Select the *Driver* tab and click the *Driver Details* button. The version displayed should match the SDK version you installed.



Once you have verified the correct driver was installed, you can try running the camera and acquiring images by running the FlyCap sample program. For more information on this program proceed to the *The FlyCap Application* section of this manual.

Chapter 3: The FlyCapture Software

This chapter guides you through the usage of the software applications included with the PGR FlyCapture SDK.

The FlyCap Application

The FlyCap application is a generic streaming image viewer that can be used to test whether or not the SDK installation succeeded. It is a simple application that will display many of the capabilities of your compatible PGR IEEE-1394 camera. It allows you to view a live video stream from the camera and save individual images, adjust the various properties and settings of the camera, and get and set camera registers.

> ***NOTE**: The full FlyCap C/C++ source code is included with the PGR FlyCapture SDK. To access the FlyCap workspace from the Start menu, select Program Files → Point Grey Research Inc. → PGR FlyCapture → Examples → FlyCap project.*

Grabbing Images

1. From the Start menu, select *Program Files → Point Grey Research Inc. → PGR FlyCapture → flycap.exe*. This will launch the application and the *Select Camera* dialog box will appear.

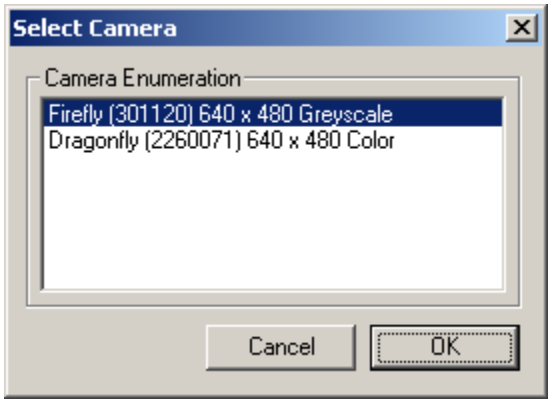


Figure 7: The FlyCap application "Select Camera" dialog box

2. Select the camera you want to acquire images from and click *OK*. The main FlyCap window will appear.

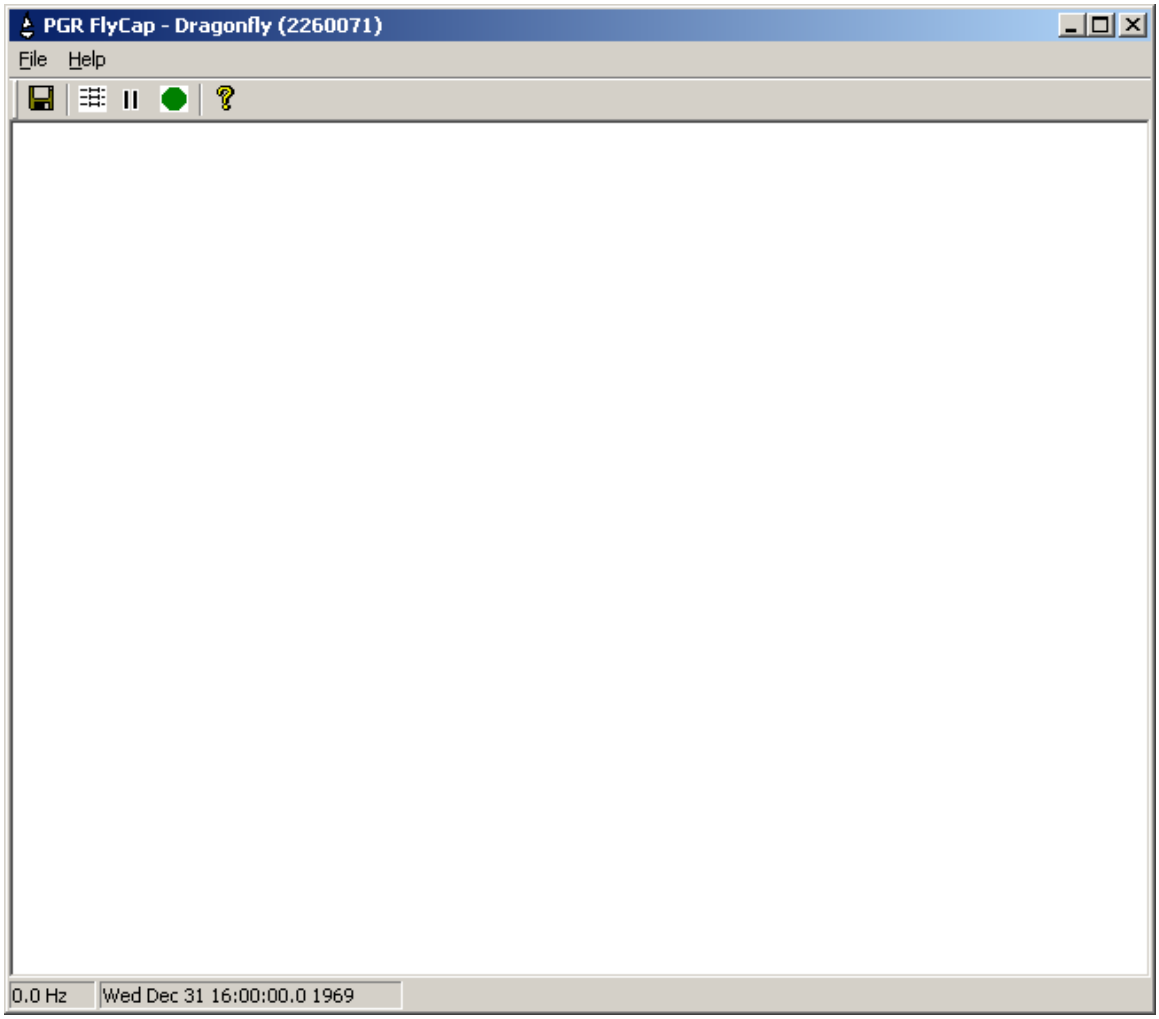


Figure 8: The main FlyCap graphical user interface (GUI)

3. To begin start the camera and begin grabbing images click the green Play button on the toolbar. A video stream will be displayed in the FlyCap window, and frame rate and timing information will be displayed in the lower status bar.

Figure 9: Example of FlyCap grabbing images

4. Click the Pause button to pause image acquisition, or click the green Play button again to actually stop the camera from streaming images to the computer.

Saving Images

FlyCap allows you to save single images in a Portable Pixelmap (.ppm) format. PPM is a very simple color image file format that may store pixel values up to 24 bits in size. Paint Shop Pro by Jasc Software (http://www.jasc.com) is a good image editor that can open and display .ppm images.

To save a single frame, click the “Save” icon on the toolbar. You should see the image paused in the window; this is the image that will be saved.

Camera Control Dialog Functions

The *Camera Control* dialog allows you to alter most camera properties, check camera information, and (for advanced users) directly access camera hardware registers to get and set specific register values.

To access the Camera Control dialog box, either:

- Click *File* → *Camera Control*; or
- Press the F11 key on your keyboard; or
- Click the camera icon on the toolbar.

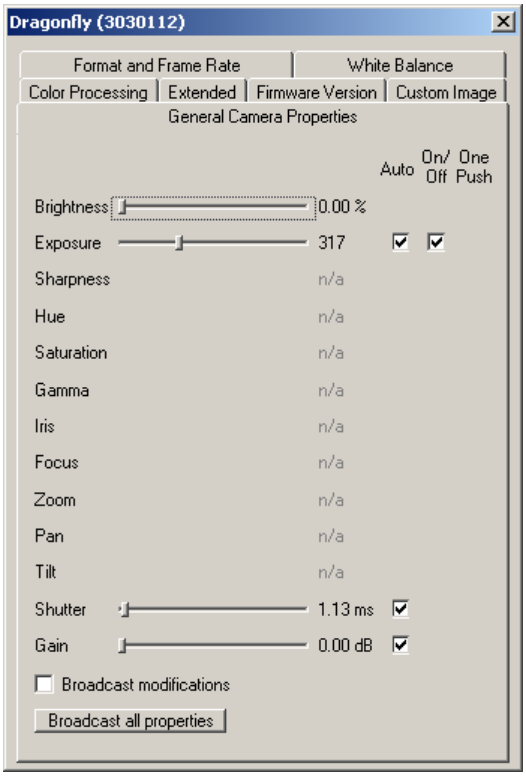


Figure 10: The FlyCap Camera Control dialog box

> ***NOTE:*** *Some camera controls and formats may be greyed out. If a camera control (such as Zoom above) is greyed out, this means that the function is not supported by the camera.*

General Camera Properties

General camera properties include Brightness, Exposure, Shutter and Gain. Some general properties will have an automatic mode in which the camera will automatically make property

adjustments to get the best image possible given the environment and lighting conditions. Following are some brief definitions of some of the camera properties:

Brightness: This is the level of black in an image. A high brightness will result in a low amount of black in the image.

Color Saturation: This is how far a color is from a gray image of the same intensity. For example, red is highly saturated, whereas a pale pink is not.

Exposure: This is the average intensity of the image. It will use other available (non-manually adjustable) controls to adjust the image. Specifically, when shutter and gain are both in auto mode, manually adjusting the exposure slider is actually adjusting the auto-exposure, which tries to make the average intensity of the image 1/4 of the auto-exposure value e.g. exposure is 400, the camera will try to adjust shutter and gain so that the average image intensity is 100. When the auto checkbox is checked for exposure, auto auto-exposure is enabled, which tries to manipulate shutter and gain such that 5% of the image is saturated (pixel value of 255).

Gain (dB): The amount of amplification that is applied to a pixel. An increase in gain can result in an increase in noise.

Gamma: Gamma correction works by adjusting for non-linearity's in the phosphor excitation. It is used to correctly display intensities on a monitor or on film. This is necessary because the pixels are not displayed on a monitor in a linear manner.

Sharpness: This works by filtering the image to reduce blurred edges in an image.

Shutter (ms): This is the speed at which the camera shutter opens and closes.

The *Broadcast Modifications* checkbox allows you to broadcast the current camera's settings to other cameras of the same type that are on the same 1394 bus. In other words, checking this and making a change to your current Dragonfly's gain settings will cause other Dragonfly's on the same FireWire bus to have the same gain settings. Clicking the *Broadcast Properties* button causes this behaviour one time only.

Format and Frame Rate

From the *Format and Frame Rate* tab you can change the resolution (horizontal and vertical pixel dimensions), image format (e.g. Y8, RGB, YUV422, etc.) and frame rate (number of frames transmitted per second) of the camera. Different cameras may have different formats and frame rates implemented. These modes conform to IEEE-1394 Digital Camera specifications. Please consult the 1394 camera specification webpage for more information: http://1394ta.org .

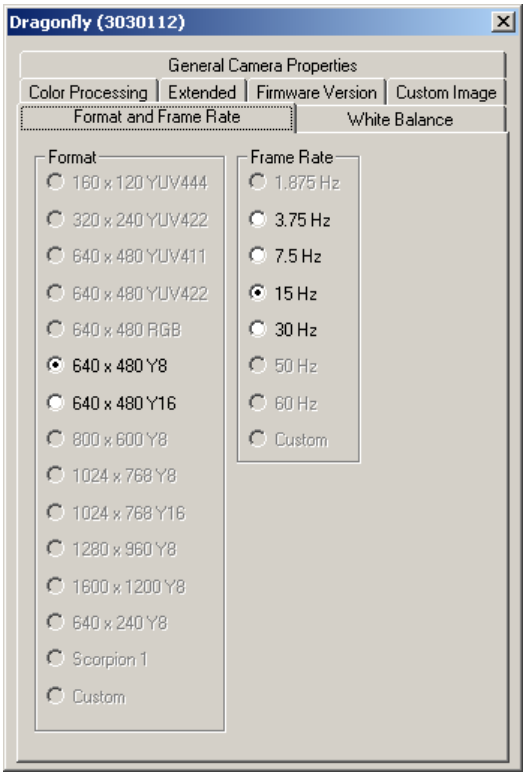


Figure 11: FlyCap Camera Control dialog: Format and Frame Rate tab

Custom Image

Some cameras allow the user to define a custom-sized image starting at a custom location. Image size must be evenly divisible by the "Unit Size". For example, a size of 16 (width) x 12 (height) in Mode 0 is correct – 4 divides evenly into 16, and 2 divides evenly into 12.

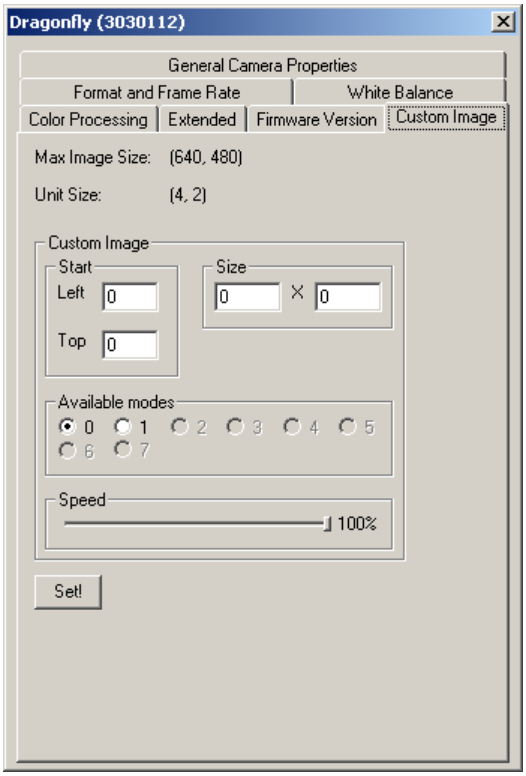


Figure 12: FlyCap Camera Control dialog: Custom Image tab

Mode 0 is a "sub-window" or "region of interest" mode. A portion of the full resolution image is displayed to the user. *Mode 1* is a "sub-sampled" or "down-sampled" mode where the entire image is displayed, but at a lower resolution.

> ***NOTE:*** *Not all PGR single-lens cameras have custom image functionality. Some cameras that do have custom image capabilities may not have all available modes implemented.*

Firmware Version

The *Firmware Version* tab provides information about the version of firmware that is loaded onto the camera, the camera serial number and the camera model name. Firmware is programming that is inserted into programmable read-only memory, thus becoming a permanent part of a computing device. Firmware is created and tested like software and can be loaded onto the camera.

Extended

The *Extended* tab provides an easy mechanism for you to put the camera into *External Trigger* mode and get and set specific camera registers.

The external trigger is a feature implemented on some PGR IEEE-1394 single lens cameras that allows the user to attach a wire to the camera so that an event can trigger the grabbing of images. This event will trigger the light collection in the camera and the image will then be grabbed. It is important to note that while the external event has not occurred, the camera will remain idle and not grab images.

For more detailed information on camera registers or configuring the external trigger, download the Technical Reference for your camera from our website.

When the *Broadcast* checkbox is checked, any register writes are broadcast to all cameras on the same bus.

White Balance

This option is only available for color cameras. White Balance allows you to control the relative levels of red and blue in an image to achieve proper color balance. Moving both red and blue values toward zero should make the image appear more green. Green is kept as a constant and the red and blue colors are adjusted relative to the green pixel. Hardware white balance is actually performed prior to the signal being digitized as it comes off of the sensor, which results in higher quality images. Selecting the *On/Off* checkbox turns on or off white balance control – this functionality only works with cameras that have recent versions of firmware.

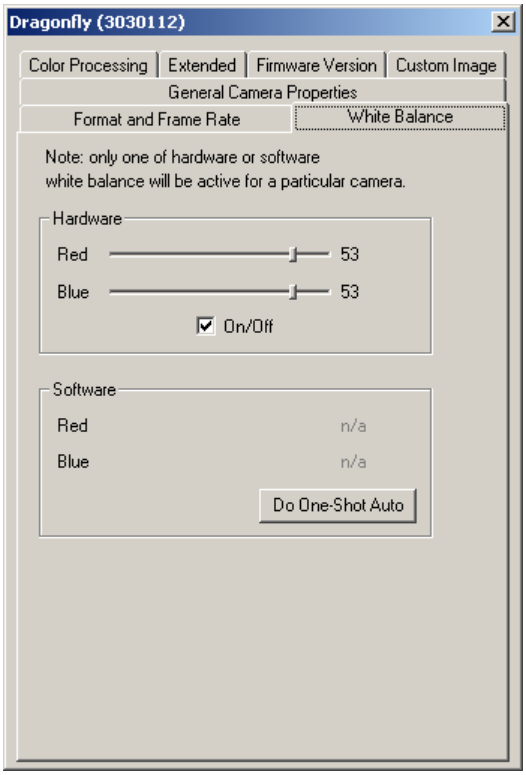


Figure 13: FlyCap Camera Control dialog: White Balance tab

Color Processing

This option is only available for color cameras. For most PGR color cameras, the conversion of the Bayer Tiled images produced by the image sensor into color takes place on the PC, and not on the camera itself. If there is no visible difference in the image quality when selecting different color processing methods, your camera does not do color processing on the PC.

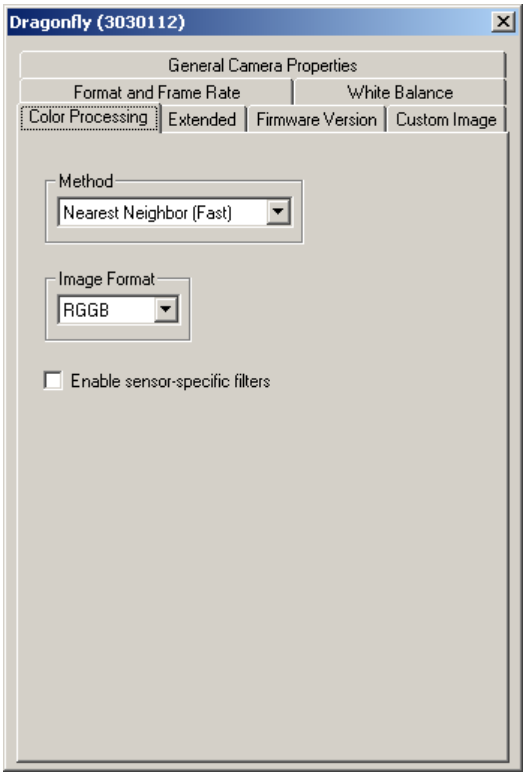


Figure 14: FlyCap Camera Control dialog: Color Processing tab

Which color conversion process is more beneficial will depend on the application of the color conversion. On-board conversion will take more bandwidth on the bus but less processor resources, whereas PC conversion does not take as much bandwidth on the bus but will be more demanding of the processor resources.

The differences between the various color processing algorithms are as follows:

- **Nearest Neighbor Faster** - This is the fastest of all of the provided color processing algorithms. It is generally considered to provide the poorest results.
- **Nearest Neighbor Fast** - Based on a very similar algorithm to the *Nearest Neighbor Faster* this algorithm is slower but performs slightly better.
- **Edge Sensing** - This algorithm is the second slowest algorithm. The algorithm weights surrounding pixels based on localized edge orientations.
- **Rigorous** - This algorithm is the slowest of all of the algorithms and, without a doubt, produces the best color quality. Rigorous image processing takes seconds, not microseconds, and is not meant for real-time processing like the other algorithms.

The *Image Format* is the orientation of the Bayer Tiling on your image sensor. This will default to the correct Bayer Tiling. The *Enable sensor-specific filters* checkbox only applies to some versions of the Scorpion camera.

Chapter 4: Examples

This chapter provides examples that demonstrate the use of the PGR FlyCapture Application Program Interface (API).

Overview

The PGR FlyCapture SDK provides a number of sample programs and source code that is meant to help the advanced programmer get started using the FlyCapture API. Examples range from simple console programs that demonstrate the basic functionality of the API, such as PGRFlyCaptureTest, to more complex examples such as the MFC application FlyCap.

> ***NOTE**: The FlyCapture header files, specifically pgrflycapture.h and pgrflycaptureplus.h, are fully commented and are the best source for specific API documentation and information.*

PGRFlyCaptureTest Example Program

The PGRFlyCaptureTest sample program is a simple program designed to capture a series of images and record the amount of time taken to grab these images. If specified, the last image is then saved in the current directory.

To use the application, select *FlyCaptureTest Example Project* from *Start → Program Files → Point Grey Research Inc. → PGR FlyCapture*. Microsoft Visual C++ will load and you will need to compile and run this example. To compile, press F7. When compiling is complete, press F5 to run the example.

Following is an abbreviated version of this program (no error handling) that initializes and starts camera 0 on the bus, grabs and saves an image and exits. This is followed by brief explanations of some of the programs key components.

```
1  //=============================================================================
2  // PGRFlyCaptureTest.cpp
3  //   - Grabs a few images and saves the last to disk.
4  //=============================================================================
5
6  #include "pgrflycapture.h"
7
8  // The maximum number of cameras on the bus
9  #define _MAX_CAMS       32
10
11 // The number of images to grab.
12 #define _IMAGES_TO_GRAB 5
13
14 // The index of the camera to grab from.
15 #define _CAMERA_INDEX   0
16
17 int main( int /* argc */, char* /* argv[] */ )
18 {
19    FlyCaptureError   error;
20    FlyCaptureContext context;
21
22    // Enumerate the cameras on the bus
23    FlyCaptureInfo arInfo[ _MAX_CAMS ];
24    unsigned int  uiSize = _MAX_CAMS;
25
26    error = flycaptureBusEnumerateCameras( arInfo, &uiSize );
27
28    for( unsigned int uiBusIndex = 0; uiBusIndex < uiSize; uiBusIndex++ )
29    {
30       printf( "Bus index %u: %s (%u)\n",
31               uiBusIndex,
32               arInfo[ uiBusIndex ].pszModelString,
33               arInfo[ uiBusIndex ].SerialNumber );
34    }
35
36    // create the flycapture context.
37    error = flycaptureCreateContext( &context );
38
39    // Initialize the camera.
40    printf( "Initializing camera %u.\n", _CAMERA_INDEX );
41    error = flycaptureInitialize( context, _CAMERA_INDEX );
42
43    // Start grabbing images in 8-bit greyscale (or stippled, if this is a
44    // colour camera) 640x480 mode with a frame rate  of 15 fps.
45    error = flycaptureStart(
46         context, FLYCAPTURE_VIDEOMODE_640x480Y8, FLYCAPTURE_FRAMERATE_15 );
47
48    // Time the grabbing of 30 images.
49    FlyCaptureImage image;
50
51    // Initialize the image structure to sane values
52    image.iCols = 0;
53    image.iRows = 0;
54
55    printf( "Grabbing images" );
56    for ( int iImage = 0; iImage < _IMAGES_TO_GRAB; iImage++ )
57    {
58       error = flycaptureGrabImage2( context, &image );
59    }
60
61    // Convert the last image to BGR24 for flycaptureWritePPM().
62    unsigned char* pimageBGR24 =
63       new unsigned char[ image.iCols * image.iRows * 3 ];
64
65    error = flycaptureConvertImage(
66       context, &image, FLYCAPTURE_OUTPUT_BGR, pimageBGR24 );
67
68    error = flycaptureWritePPM(
69       pimageBGR24, image.iRows, image.iCols, "image.ppm" );
70
71    // Destroy the context.
72    error = flycaptureDestroyContext( context );
73
74    delete [] pimageBGR24;
75
76    return 0;
77 }
```

Figure 15: PGRFlyCaptureTest source code

PGRFlyCaptureTest Example Program: Description

Line #	Description
19	Creates a FlyCaptureError variable. In order to reliably debug your application, define a variable of this type to capture meaningful errors returned by API functions.
26	Enumerates, or lists, all PGR cameras sitting on the bus and their bus index, starting at zero. This function does not enumerate non-PGR cameras.
37	Creates a FlyCaptureContext. A context acts as a handle to the camera, and is required to initialize and start the camera.
41	Initializes the camera located at bus index zero and associates it with the camera context. Multiple cameras connected to the FireWire bus are enumerated at bus indices (FireWire nodes) that start at 0.
45	Starts an initialized camera at a defined resolution (640x480 pixels), image format (Y8 8 bits per pixel) and frame rate (15 frames captured per second). This function also allocates four buffers in main memory that are used to hold the images that are streamed in from the camera. Once a camera has been started, it immediately begins capturing and streaming images via DMA to these memory buffers. Once these buffers are full, they will be overwritten with consecutive images unless they are locked by the user.
49	Creates a FlyCaptureImage variable. The FlyCaptureImage structure contains the image data, as well as video mode, whether the image is stippled (color) and timestamp information. See the section *Timestamping Mechanisms* in the *Advanced Features* chapter for more information on the types of timestamps available.
58	When a call to flycaptureGrabImage2 is made, a pointer to the image buffer (&image) with the newest (latest) complete image is returned. The call to grabImage2 does not involve copying, so it is quite fast. The user will never be given an image that is older than one that has already been seen. Once the pointer to the buffer is returned to the user, this buffer remains locked until grabImage2 is called again. If no buffer contains an image newer than the last returned, then the flycaptureGrabImage2 call will block until a new image is available.
65	Converts the last image grabbed to a 24-bit per pixel image that can be displayed by Microsoft Windows (which uses the BGR format). If the camera is not a color camera, Y8 and Y16 images are converted to BGR24 greyscale.
68	Writes the image out to a PPM format file.
72	Destroys the camera context. In order to prevent memory leaks from occurring, this function must be called when the user is finished with the FlyCaptureContext.

Chapter 5: The FlyCapture API

This chapter gives a detailed description of the PGR FlyCapture Application Programming Interface functionality.

Camera Property Functions

flycaptureGetCameraAbsProperty

Allows the user to get the current absolute value for a given parameter from the camera if it is supported.

Declaration

```
FlyCaptureError
flycaptureGetCameraAbsProperty(
                                FlyCaptureContext   context,
                                FlyCaptureProperty  cameraProperty,
                                float*              pfValue )
```

Parameters

context	The FlyCapture context to query.
cameraProperty	A FlyCaptureProperty indicating which property to query.
pfValue	A pointer to a float that will contain the result.

Return Value
A FlyCaptureERror indicating the sucess or failure of the operation.

flycaptureGetCameraAbsPropertyRange

Allows the user to determine the presence and range of the absolute value registers for the camera

Declaration

```
FlyCaptureError
flycaptureGetCameraAbsPropertyRange(
                                    FlyCaptureContext  context,
                                    FlyCaptureProperty cameraProperty,
                                    bool*              pbPresent,
                                    float*             pfMin,
                                    float*             pfMax,
                                    const char**       ppszUnits,
                                    const char**       ppszUnitAbbr)
```

Parameters

context	The Flycapture context to query.
cameraProperty	A FlyCaptureProperty indicating which property to query.
pbPresent	Whether or not this register has absolute value support.
pfMin	The minimum value that this register can handle.
pfMax	The maximum value that this register can handle.

ppszUnits	A string indicating the units of the register.
ppszUnitAbbr	An abbreviation of the units

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureGetCameraProperty

Allows the user to query the current value of the given property.

Declaration

```
FlyCaptureError
flycaptureGetCameraProperty(
                            FlyCaptureContext   context,
                            FlyCaptureProperty  cameraProperty,
                            long*               plValueA,
                            long*               plValueB,
                            bool*               pbAuto   )
```

Parameters

context	The FlyCapture context to extract the properties from.
cameraProperty	A FlyCaptureProperty indicating the property to query.
plValueA	A pointer to storage space for the “A”, or first value associated with this property.
plValueB	A pointer to storage space for the “B”, or second value associated with this property.
pbAuto	A pointer to a bool that will store the current Auto value of the property.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Pass NULL for any pointer argument to ignore that argument.

flycaptureGetCameraPropertyEx

Replaces flycaptureGetCameraProperty() and provides better access to camera features.

Declaration

```
FlyCaptureError
flycaptureGetCameraPropertyEx(
                              FlyCaptureContext   context,
                              FlyCaptureProperty  cameraProperty,
                              bool*               pbOnePush,
                              bool*               pbOnOff,
                              bool*               pbAuto,
                              int*                piValueA,
                              int*                piValueB )
```

Parameters

context	The FlyCapture context to extract the properties from.
cameraProperty	A FlyCaptureProperty indicating the property to query.
pbOnePush	The value of the one push bit.
pbOnOff	The value of the On/Off bit.
pbAuto	The value of the Auto bit.
piValueA	The current value of this property.
piValueB	The current secondary value of this property. (only used for the two whitebalance values.)

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Pass NULL for any pointer argument to ignore that argument.

flycaptureGetCameraPropertyRange

Allows the user to examine the default, minimum, maximum, and auto characteristics for the given property.

Declaration

```
FlyCaptureError
flycaptureGetCameraPropertyRange(
                                  FlyCaptureContext  context,
                                  FlyCaptureProperty cameraProperty,
                                  bool*              pbPresent,
                                  long*              plMin,
                                  long*              plMax,
                                  long*              plDefault,
                                  bool*              pbAuto,
                                  bool*              pbManual   )
```

Parameters

context	The FlyCapture context to extract the properties from.
cameraProperty	A FlyCaptureProperty indicating the property to examine.
pbPresent	A pointer to a bool that will contain whether or not camera property is present.
plMin	A pointer to a long that will contain the minimum property value.
plMax	A pointer to a long that will contain the maximum property value.
plDefault	A pointer to a long that will contain the default property value.
pbAuto	A pointer to a bool that will contain whether or not the Auto setting is available for this property.

pbManual	A pointer to a bool that will contain whether or not this property may be manually adjusted.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Pass NULL for any pointer argument to ignore that argument.

flycaptureGetCameraPropertyRangeEx

Replaces flycaptureGetCameraPropertyRange() and provides better access to camera features.

Declaration

```
FlyCaptureError
flycaptureGetCameraPropertyRangeEx(
                                    FlyCaptureContext    context,
                                    FlyCaptureProperty   cameraProperty,
                                    bool*                pbPresent,
                                    bool*                pbOnePush,
                                    bool*                pbReadOut,
                                    bool*                pbOnOff,
                                    bool*                pbAuto,
                                    bool*                pbManual,
                                    int*                 piMin,
                                    int*                 piMax )
```

Parameters

context	The FlyCapture context to extract the properties from.
cameraProperty	A FlyCaptureProperty indicating the property to examine.
pbPresent	Indicates the presence of this property on the camera.
pbOnePush	Indicates the availablilty of the one push feature.
pbReadOut	Indicates the ability to read out the value of this property.
pbOnOff	Indicates the ability to turn this property on and off.
pbAuto	Indicates the availability of auto mode for this property.
pbManual	Indicates the ability to manually control this property.
piMin	The minimum value of the property is returned in this argument.
piMax	The maximum value of the property is returned in this argument.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Pass NULL for any pointer argument to ignore that argument.

flycaptureGetCameraRegister

This function allows the user to get any of camera's registers.

Declaration

```
FlyCaptureError
flycaptureGetCameraRegister(
                            FlyCaptureContext context,
                            unsigned long     ulRegister,
                            unsigned long*    pulValue )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
ulRegister	The 32 bit register location to query.
pulValue	The 32 bit value currently stored in the register.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureGetCameraTrigger

This function allows the user to query the presence and state of the camera's asynchronous trigger register.

Declaration

```
FlyCaptureError
flycaptureGetCameraTrigger(
                           FlyCaptureContext context,
                           unsigned int*     puiPresence,
                           unsigned int*     puiOnOff,
                           unsigned int*     puiPolarity,
                           unsigned int*     puiTriggerMode )
```

Parameters

context	The context associated with the camera to be queried.
puiPresence	Whether or not the camera supports the feature (1) or not (0).
puiOnOff	Whether the feature is engaged (1) or disengaged(0).
puiPolarity	Whether the trigger has a low (0) or high (1) active input.
puiTriggerMode	The mode (0-15) that the trigger is currently in.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureSetCameraAbsProperty

Allows the user to set the absolute value of the given parameter if the mode is supported.

Declaration

```
FlyCaptureError
flycaptureSetCameraAbsProperty(
                                FlyCaptureContext  context,
                                FlyCaptureProperty cameraProperty,
                                float              fValue )
```

Parameters

context	The FlyCapture context to query.
cameraProperty	A FlyCaptureProperty indicating which property to query.
fValue	A float containing the new value of the parameter.

Return Value
A FlyCaptureError indicating the success or failure of the operation.

flycaptureSetCameraAbsPropertyBroadcast

Allows the user to set the abosolute value of the given parameter to all cameras on the current bus.

Declaration

```
FlyCaptureError
flycaptureSetCameraAbsPropertyBroadcast(
                                        FlyCaptureContext  context,
                                        FlyCaptureProperty cameraProper
ty,
                                        float              fValue )
```

Parameters

context	The FlyCapture context to query.
cameraProperty	A FlyCaptureProperty indicating which property to query.
fValue	A float containing the new value of the parameter.

Return Value
A FlyCaptureError indicating the success or failure of the operation.

flycaptureSetCameraProperty

Allows the user to set the given property.

Declaration

```
FlyCaptureError
flycaptureSetCameraProperty(
                            FlyCaptureContext  context,
                            FlyCaptureProperty cameraProperty,
                            long               lValueA,
                            long               lValueB,
                            bool               bAuto )
```

Parameters

context	The FlyCaptureContext to set the properties in.
cameraProperty	A FlyCaptureProperty indicating the property to set.
lValueA	A long containing the “A”, or first new value of the property.
lValueB	A long containing the “B”, or second new value of the property.
bAuto	A boolean containing the new 'auto' state of the property.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Calling this function with either of FLYCAPTURE_SOFTWARE_WHITEBALANCE as the cameraProperty parameter and 'true' for the bAuto parameter will invoke a single shot auto white balance method. The assumption is that flycaptureGrabImage() has been called previously with a white object centered in the field of view. This will only work if the camera is a color camera and in RGB mode. The Red and Blue whitebalance parameters only affect cameras that do offboard color calculation such as the Dragonfly.

flycaptureSetCameraPropertyBroadcast

Allows the user to set the given property for all cameras on the bus.

Declaration

```
FlyCaptureError
flycaptureSetCameraPropertyBroadcast(
                                     FlyCaptureContext  context,
                                     FlyCaptureProperty cameraProperty,
                                     long               lValueA,
                                     long               lValueB,
                                     bool               bAuto )
```

Parameters

context	The FlyCaptureContext to set the properties in.
cameraProperty	A FlyCaptureProperty indicating the property to set.
lValueA	A long containing the “A”, or first new value of the property.
lValueB	A long containing the “B”, or second new value of the property.
bAuto	A boolean containing the new 'auto' state of the property.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function will set the given property for all the cameras on the 1394 bus. If you are using multiple busses (ie, more than one 1394 card) you must call this function for each bus, on a context representing a camera on that bus.

flycaptureSetCameraPropertyBroadcastEx

Replaces flycaptureSetCameraPropertyBroadcast() and provides better access to camera features.

Declaration

```
FlyCaptureError
flycaptureSetCameraPropertyBroadcastEx(
                                        FlyCaptureContext    context,
                                        FlyCaptureProperty   cameraPrope
rty,
                                        bool                 bOnePush,
                                        bool                 bOnOff,
                                        bool                 bAuto,
                                        int                  iValueA,
                                        int                  iValueB )
```

Parameters

context	The FlyCaptureContext to set the properties in.
cameraProperty	A FlyCaptureProperty indicating the property to set.
bOnePush	Set the one push bit.
bOnOff	Set the on/off bit.
bAuto	Set the auto bit.
iValueA	The value to set.
iValueB	The seconary value to set. (only used for the two whitebalance values.)

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function will set the given property for all the cameras on the 1394 bus. If you are using multiple busses (ie, more than one 1394 card) you must call this function for each bus, on a context representing a camera on that bus.

flycaptureSetCameraPropertyEx

Replaces flycaptureSetCameraPropertyEx() and provides better access to camera features.

Declaration

```
FlyCaptureError
flycaptureSetCameraPropertyEx(
                               FlyCaptureContext    context,
                               FlyCaptureProperty   cameraProperty,
                               bool                 bOnePush,
                               bool                 bOnOff,
                               bool                 bAuto,
                               int                  iValueA,
                               int                  iValueB )
```

Parameters

context	The FlyCaptureContext to set the properties in.
cameraProperty	A FlyCaptureProperty indicating the property to set.
bOnePush	Set the one push bit.
bOnOff	Set the on/off bit.
bAuto	Set the auto bit.
iValueA	The value to set.
iValueB	The seconary value to set. (only used for the two whitebalance values.)

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureSetCameraRegister

This function allows the user to get any of the camera's registers.

Declaration

```
FlyCaptureError
flycaptureSetCameraRegister(
                            FlyCaptureContext context,
                            unsigned long     ulRegister,
                            unsigned long     ulValue )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
ulRegister	The 32 bit register location to set.
ulValue	The 32 bit value to store in the register.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureSetCameraRegisterBroadcast

This function allows the user to get any register for all cameras on the bus.

Declaration

```
FlyCaptureError
flycaptureSetCameraRegisterBroadcast(
                                     FlyCaptureContext context,
                                     unsigned long     ulRegister,
                                     unsigned long     ulValue )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
ulRegister	The 32 bit register location to set.
ulValue	The 32 bit value to store in the register.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureSetCameraTrigger

This function allows the user to set the state of the camera's asynchronous trigger register.

Declaration

```
FlyCaptureError
flycaptureSetCameraTrigger(
                            FlyCaptureContext context,
                            unsigned int      uiOnOff,
                            unsigned int      uiPolarity,
                            unsigned int      uiTriggerMode )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
uiOnOff	Whether to engage (1) or disengage(0) the feature.
uiPolarity	Whether the trigger should have a low (0) or high (1) active input.
uiTriggerMode	The mode (0-15) that the trigger should be in.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
If you have set a grab timeout using flycaptureSetGrabTimeout(), this timeout will be used in asynchronous trigger mode as well: flycaptureGrabImage*() will return with the image when you either trigger the camera, or the timeout value expires.

flycaptureSetCameraTriggerBroadcast

This function allows the user to set the state of the asynchronous trigger register for all cameras on the bus.

Declaration

```
FlyCaptureError
flycaptureSetCameraTriggerBroadcast(
                                     FlyCaptureContext context,
                                     unsigned char     ucOnOff,
                                     unsigned char     ucPolarity,
                                     unsigned char     ucTriggerMode )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
ucOnOff	An unsigned char that indicates whether to engage (1) or disengage(0) the feature.
ucPolarity	An unsigned char that indicates whether the trigger should have a low (0) or high (1) active input.
ucTriggerMode	An unsigned char that indicates the mode (0-15) that the trigger should be in.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureSetGrabTimeout

This function allows the user to set the timeout value for flycaptureGrabImage*(). This is not normally necessary.

Declaration

```
FlyCaptureError
flycaptureSetGrabTimeout(
                        FlyCaptureContext   context,
                        unsigned int        uiTimeout )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
uiTimeout	The timeout value, in milliseconds. A value of zero indicates an infinite wait.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
The default grab timeout value is “infinite.”

Construction/Destruction

flycaptureCreateContext

This function creates a FlyCaptureContext and allocates all of the memory that it requires. The purpose of the FlyCaptureContext is to act as a handle to one of the cameras attached to the system. This call must be made before any other calls involving the context will work.

Declaration

```
FlyCaptureError
flycaptureCreateContext(
                       FlyCaptureContext* pContext )
```

Parameters

pContext	A pointer to the FlyCaptureContext to be created.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureDestroyContext

Destroys the given FlyCaptureContext. In order to prevent memory leaks from occurring, this function must be called when the user is finished with the FlyCaptureContext.

Declaration

```
FlyCaptureError
flycaptureDestroyContext(
                          FlyCaptureContext context )
```

Parameters

context	The FlyCaptureContext to be destroyed.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureGetCameraInformation

Fills a FlyCaptureInfo structure with the relevant information from the camera system.

Declaration

```
FlyCaptureError
flycaptureGetCameraInformation(
                                FlyCaptureContext context,
                                FlyCaptureInfo*   pFlyCaptureInfo )
```

Parameters

context	The FlyCaptureContext to access the FlyCapture camera.
pFlyCaptureInfo	A pointer to a FlyCaptureInfo structure.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureInitialize

This function initializes one of the cameras on the bus and associates it with the provided FlyCaptureContext. This call must be made after a flycaptureCreateContext() command and prior to a flycaptureStart() command in order for images to be grabbed. Users can also use the flycaptureInitializeFromSerialNumber() command to initialize a FlyCapture with a specific serial number.

Declaration

```
FlyCaptureError
```

```
flycaptureInitialize(
                    FlyCaptureContext context,
                    unsigned long     ulDevice )
```

Parameters

context	The FlyCaptureContext to be associated with the camera being initialized.
ulDevice	The 1394 bus index of the FlyCapture camera to be initialized.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureInitializeFromSerialNumber(), flycaptureCreateContext(), flycaptureStart()

flycaptureInitializeFromSerialNumber

Similar to the flycaptureInitialize() command, this function initializes one of the cameras on the bus and associates it with the given FlyCaptureContext. This function differs from its counter part in that it takes a serial number rather than a bus index.

Declaration

```
FlyCaptureError
flycaptureInitializeFromSerialNumber(
                                    FlyCaptureContext            conte
xt,
                                    FlyCaptureCameraSerialNumber seria
lNumber )
```

Parameters

context	The FlyCaptureContext to be associated with the camera being initialized.
serialNumber	The serial number of the FlyCapture camera system to be initialized.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureInitialize(), flycaptureCreateContext(), flycaptureStart()

Control Functions

flycaptureCheckVideoMode

This funciton allows the user to check if a given mode is supported by the camera.

Declaration

```
FlyCaptureError
flycaptureCheckVideoMode(
```

```
                                FlyCaptureContext   context,
                                FlyCaptureVideoMode videoMode,
                                FlyCaptureFrameRate frameRate,
                                bool*               pbSupported )
```

Parameters

context	An initialized FlyCaptureContext.
videoMode	The video mode to check.
frameRate	The frame rate to check.
pbSupported	A pointer to a bool that will store whether or not the mode is supported.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureGetColorProcessingMethod

This function allows users to check the current color processing method.

Declaration

```
FlyCaptureError
flycaptureGetColorProcessingMethod(
                                    FlyCaptureContext          context,
                                    FlyCaptureColorMethod*     pMethod )
```

Parameters

context	The FlyCapture context to access.
pMethod	A pointer to a FlyCaptureColorMethod that will store the current color processing method.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function is only applicable when using the SDK and driver with cameras that do not do on board color processing, such as the dragonfly.

See Also
flycaptureSetColorProcessingMethod()

flycaptureGetColorTileFormat

This function allows users to check the current color tile destippling format.

Declaration

```
FlyCaptureError
flycaptureGetColorTileFormat(
                                FlyCaptureContext          context,
                                FlyCaptureStippledFormat*  pformat )
```

Parameters

context	The FlyCapture context to access.
pformat	A pointer to a FlyCaptureStippledFormat that will store the current color tile format.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
The color tile format indicates the format of the stippled image the camera returns. This function is only applicable to cameras that do not do onboard color processing.

flycaptureGetCurrentVideoMode

This funciton allows the user to request the current video mode.

Declaration

```
FlyCaptureError
flycaptureGetCurrentVideoMode(
                              FlyCaptureContext    context,
                              FlyCaptureVideoMode* pVideoMode,
                              FlyCaptureFrameRate* pFrameRate)
```

Parameters

context	An initialized FlyCaptureContext.
pVideoMode	A pointer to a video mode to be filled in.
pFrameRate	A pointer to a frame rate to be filled in.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureQueryCustomImage

This function queries the options available for the advanced Custom Image functionality.

Declaration

```
FlyCaptureError
flycaptureQueryCustomImage(
                           FlyCaptureContext context,
                           unsigned int      uiMode,
                           bool*             pbAvailable,
                           unsigned int*     puiMaxImagePixelsWidth,
                           unsigned int*     puiMaxImagePixelsHeight,
                           unsigned int*     puiPixelUnitHorz,
                           unsigned int*     puiPixelUnitVert )
```

Parameters

context	The FlyCaptureContext to start grabbing.

uiMode	The mode to query (0-7).
pbAvailable	Indicates the availability of this mode.
puiMaxImagePixelsWidth	Maximum horizonal pixels.
puiMaxImagePixelsHeight	Maximum vertical pixels.
puiPixelUnitHorz	Indicates the horizonal “step size” of the custom image.
puiPixelUnitVert	Indicates the vertical “step size” of the custom image.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureStartCustomImage()

flycaptureSetColorProcessingMethod

This function allows users to select the method used for color processing.

Declaration

```
FlyCaptureError
flycaptureSetColorProcessingMethod(
                                    FlyCaptureContext        context,
                                    FlyCaptureColorMethod    method )
```

Parameters

context	The FlyCapture context to access.
method	A variable of type FlyCaptureColorMethod indicating the color processing method to be used.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Selecting FLYCAPTURE_DISABLE turns off all color processing. FLYCAPTURE_NEAREST_NEIGHBOR uses a fast nearest neighbor approach to de-mosaic the images and DIGICLOPS_EDGE_SENSING uses a more expensive method to produce higher quality color images.

Remarks
This function is only applicable when using the SDK and driver with cameras that do not do on board color processing, such as the dragonfly.

See Also
flycaptureGetColorProcessingMethod()

flycaptureSetColorTileFormat

This function sets the color tile destippling format.

Declaration

```
FlyCaptureError
flycaptureSetColorTileFormat(
                              FlyCaptureContext           context,
                              FlyCaptureStippledFormat    format )
```

Parameters

context	The FlyCapture context to access.
format	The FlyCaptureStippledFormat to set.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
The color tile format indicates the format of the stippled image the camera returns. This function is only applicable to cameras that do not do onboard color processing.

flycaptureStart

This function starts the image grabbing process. It should be called after flycaptureCreateContext() and flycaptureInitialize().

Declaration

```
FlyCaptureError
flycaptureStart(
                FlyCaptureContext      context,
                FlyCaptureVideoMode    videoMode,
                FlyCaptureFrameRate    frameRate   )
```

Parameters

context	The FlyCaptureContext to start grabbing.
videoMode	The video mode to start the camera in.
frameRate	The frame rate to start the camera at.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureCreateContext(), flycaptureInitialize(), flycaptureInitializeFromSerialNumber(), flycaptureStop()

flycaptureStartCustomImage

This function starts the image grabbing process with advanced "custom image" (DCAM Format 7) functionality, which allows the user to select a custom image size and/or region of interest.

Declaration

```
FlyCaptureError
```

```
flycaptureStartCustomImage(
                            FlyCaptureContext context,
                            unsigned int      uiMode,
                            unsigned int      uiImagePosLeft,
                            unsigned int      uiImagePosTop,
                            unsigned int      uiWidth,
                            unsigned int      uiHeight,
                            float             fBandwidth  )
```

Parameters

context	The FlyCaptureContext to start grabbing.
uiMode	The camera-specific mode. (0-7).
uiImagePosLeft	The left position of the (sub)image.
uiImagePosTop	Top top posision of the (sub)image.
uiWidth	The width of the (sub)image.
uiHeight	The height of the (sub)image.
fBandwidth	The bandwidth to assign to this camera. 100.0 indicates full bandwidth.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureQueryCustomImage()

flycaptureStop

This function halts all image grabbing for the specified FlyCaptureContext.

Declaration

```
FlyCaptureError
flycaptureStop(
               FlyCaptureContext context )
```

Parameters

context	The FlyCaptureContext to stop.

Return Value
A FlyCaptureError indicating the success or failure of the function.

FlyCapture Bus

flycaptureBusCameraCount

This function returns the number of cameras on the 1394 bus.

Declaration

```
FlyCaptureError
flycaptureBusCameraCount(
                          unsigned int* puiCount )
```

Parameters

puiCount	Returns the number of cameras on the bus.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureBusEnumerateCameras

This function enumerates all of the FlyCapture cameras found on the 1394 bus. It fills an array of FlyCaptureInfo structures with all of the pertinent information from the attached cameras. The index of a given FlyCaptureInfo structure in the array pFlyCaptureInfoArray is the FlyCapture device number.

Declaration

```
FlyCaptureError
flycaptureBusEnumerateCameras(
                               FlyCaptureInfo*   pFlyCaptureInfoArray,
                               unsigned int*     puiSize )
```

Parameters

pFlyCaptureInfoArray	An array of FlyCaptureInfo structures, at least as large as the number of FlyCapture cameras on the bus.
puiSize	The size of the array passed in. The number of cameras detected is passed back in this argument also.

Return Value
A FlyCaptureError indicating the success or failure of the function.

See Also
flycaptureBusCameraCount()

flycaptureBusRegisterNotificationCallback

This function is depricated. Use flycaptureModifyCallback().

Declaration

```
FlyCaptureError
flycaptureBusRegisterNotificationCallback(
   FlyCaptureBusNotificationCallback* pCallbackFunction,
   bool                               bRegister )
```

Return Value

Value	Meaning
FLYCAPTURE_NOT_IMPLEMENTED	Function not implemented.

flycaptureModifyCallback

This function registers or deregisters a bus callback function. When the state of the bus changes, the FlyCaptureCallback function will be called with an integer message parameter. Please see the FlyCap example for more information on how to use callback functionality.

Declaration

```
FlyCaptureError
flycaptureModifyCallback(
                          FlyCaptureContext   context,
                          FlyCaptureCallback* pfnCallback,
                          void*               pparam,
                          bool                bAdd )
```

Parameters

context	The FlyCapture context to access.
pfnCallback	A pointer to an externally defined callback function.
pparam	A user-specified parameter to be passed back to the callback function. Can be NULL.
bAdd	true if the callback is to be added to the list of callbacks, false if the callback is to be removed.

Return Value
A FlyCaptureError indicating the success or failure of the function.

General

flycaptureErrorToString

This function returns a description of the provided FlyCaptureError.

Declaration

```
const char*
flycaptureErrorToString(
                         FlyCaptureError error )
```

Parameters

error	The FlyCapture error to be parsed.

Return Value
A null-terminated character string that describes the FlyCapture error.

flycaptureGetLastError

Deprecated. This function will be removed in a future release.

Declaration

```
FlyCaptureError
flycaptureGetLastError(
                        FlyCaptureContext context )
```

Parameters

context	The FlyCaptureContext to examine.

Return Value
FLYCAPTURE_NOT_IMPLEMENTED.

flycaptureGetLibraryVersion

This function returns the version of the library in the format 100*(major version)+(minor version).

Declaration

```
int
flycaptureGetLibraryVersion()
```

Return Value
An integer indicating the current version of the library.

Image Related Functions

flycaptureConvertImage

Convert an arbitrary image format to another format.

Declaration

```
FlyCaptureError
flycaptureConvertImage(
                        FlyCaptureContext        context,
                        const FlyCaptureImage*   pimage,
                        FlyCaptureOutputFormat   outputFormat,
                        unsigned char*           pDestBuffer )
```

Parameters

context	The FlyCapture context to access.
pimage	A pointer to the image to convert.
outputFormat	The desired output format.
pDestBuffer	Pointer to an allocated buffer to hold the resultant image.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks

This function replaces flycaptureConvertToBGR24(), flycaptureStippledToBGR24(), and flycaptureStippledToBGRU32().

flycaptureConvertToBGR24

Deprecated. Please use flycaptureConvertImage(). Convert an arbitrary image format to windows-displayable 24-bit BGR.

Declaration

```
FlyCaptureError
flycaptureConvertToBGR24(
                        const FlyCaptureImage* pimage,
                        unsigned char*         pDestBuffer )
```

Parameters

pimage	A pointer to the image to convert.
pDestBuffer	Pointer to an allocated buffer to hold the resultant image. This buffer must be at least iImagePixels * 3 bytes long.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
Note that passing an RGB image into this function will work, but it is much more efficient to just call flycaptureInplaceRGB24toBGR24(). Y8 and Y16 images are converted to BGR24 greyscale. Note that in the case of Y16 this is a lossy conversion.

flycaptureGrabImage

This function grabs the newest image from the FlyCapture camera system and passes the image buffer and information to the user.

Declaration

```
FlyCaptureError
flycaptureGrabImage(
                    FlyCaptureContext     context,
                    unsigned char**       ppImageBuffer,
                    int*                  piRows,
                    int*                  piCols,
                    int*                  piRowInc,
                    FlyCaptureVideoMode*  pVideoMode  )
```

Parameters

context	The FlyCapture context to lock the image in.
ppImageBuffer	Pointer to the returned image buffer pointer.
piRows	Pointer to the returned rows.
piCols	Pointer to the returned columns.

piRowInc	Pointer to the returned row increment (number of bytes per row.)
pVideoMode	Pointer to the returned video mode.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function will block until a new image is available. You can optionally set the timeout value for the wait using the flycaptureSetGrabTimeout() (by default the wait time is infinite.) Setting the timeout value should normally not be necessary.

See Also
flycaptureStart(), flycaptureGrabImage2(), flycaptureSetGrabTimeout()

flycaptureGrabImage2

This function is identical to flycaptureGrabImage() except that it returns a FlyCaptureImage.

Declaration

```
FlyCaptureError
flycaptureGrabImage2(
                    FlyCaptureContext    context,
                    FlyCaptureImage*     pimage   )
```

Parameters

context	The FlyCapture context to lock the image in.
pimage	A pointer to a FlyCaptureImage structure that will contain the image information.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
See remarks for flycaptureGrabImage().

See Also
flycaptureStart(), flycaptureGrabImage(), flycaptureSetGrabTimeout()

flycaptureInplaceRGB24toBGR24

Changes the input image buffer from 24-bit RGB to windows-displayable 24-bit BGR.

Declaration

```
FlyCaptureError
flycaptureInplaceRGB24toBGR24(
                              unsigned char* pImageBuffer,
                              int            iImagePixels )
```

Parameters

pImageBuffer	Pointer to the image contents.
iImagePixels	Size of the image, in pixels.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureInplaceWhiteBalance

This function performs an inplace software based white balance on the provided image.

Declaration

```
FlyCaptureError
flycaptureInplaceWhiteBalance(
                              FlyCaptureContext context,
                              unsigned char*    pData,
                              int               iRows,
                              int               iCols )
```

Parameters

context	The FlyCapture context.
pData	The BGR24 image data.
iRows	Image rows.
iCols	Image columns.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
The image must be in BGR24 format. I.e., the output from one of the above functions. This function has no effect on cameras that are detected to have hardware whitebalance.

flycaptureStippledToBGR24

Deprecated. Please use flycaptureConvertImage(). Convert a stippled image of arbitrary format to a 24-bit BGR image.

Declaration

```
FlyCaptureError
flycaptureStippledToBGR24(
                          FlyCaptureContext        context,
                          const FlyCaptureImage*   pimage,
                          unsigned char*           pDestBuffer )
```

Parameters

context	The FlyCapture context.
pimage	A pointer to the image to convert.

pDestBuffer	Pointer to an allocated buffer to hold the resultant image. This buffer must be at least iRows * iCols * 3 bytes in size.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function is only relevant to cameras that do not do onboard colour processing such as older versions of the Dragonfly. This function uses the color method set in flycaptureSetColorProcessingMethod() and the stippled format used in flycaptureSetColorTileFormat().

flycaptureStippledToBGRU32

Deprecated. Please use flycaptureConvertImage(). Convert a stippled image of arbitrary format to a 32-bit BGRU image.

Declaration

```
FlyCaptureError
flycaptureStippledToBGRU32(
                          FlyCaptureContext        context,
                          const FlyCaptureImage*   pimage,
                          unsigned char*           pDestBuffer )
```

Parameters

context	The FlyCapture context.
pimage	A pointer to the image to convert.
pDestBuffer	Pointer to an allocated buffer to hold the resultant image. This buffer must be at least iRows * iCols * 4 bytes in size.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function is only relevant to cameras that do not do onboard colour processing such as older versions of the Dragonfly. This function uses the color method set in flycaptureSetColorProcessingMethod() and the stippled format used in flycaptureSetColorTileFormat().

flycaptureWritePPM

Writes the specified image buffer to disk in .PPM format.

Declaration

```
FlyCaptureError
flycaptureWritePPM(
                   const unsigned char*   pImageBuffer,
                   int                    iRows,
                   int                    iCols,
                   const char*            pszFileName )
```

Parameters

pImageBuffer	Pointer to the image buffer to write from.
iRows	Image rows.
iCols	Image columns.
pszFileName	The name of the file to write to.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
The image must be in BGR24 format, ie, the output from any of the conversion or colour processing routines.

Type Definitions

FlyCaptureBusNotificationCallback

Declaration

```
typedef void FlyCaptureBusNotificationCallback( int iBusNotificationMsg
 )
```

Remarks
Depricated function prototupe for old bus callback mechanism. Do not use. This will be removed in future versions of this SDK.

FlyCaptureCallback

Declaration

```
typedef void __cdecl
FlyCaptureCallback( void* pparam, int iMessage, unsigned long ulParam )
```

Remarks
Function prototype for the bus callback mechanism. pparam contains the parameter passed in when registering the callback. iMessage is one of the above FLYCAPTURE_MESSAGE_* #defines and ulParam is a message-defined parameter.

FlyCaptureCameraModel

An enumeration used to describe the different camera models that can be accessed through this SDK.

Declaration

```
enum FlyCaptureCameraModel
{
   FLYCAPTURE_FIREFLY,
```

```
   FLYCAPTURE_DRAGONFLY,
   FLYCAPTURE_AIM,
   FLYCAPTURE_SCORPION,
   FLYCAPTURE_TYPHOON,
   FLYCAPTURE_SONY,
   FLYCAPTURE_FLEA,
   FLYCAPTURE_UNKNOWN,

}
```

Elements

FLYCAPTURE_AIM	
FLYCAPTURE_DRAGONFLY	
FLYCAPTURE_FIREFLY	
FLYCAPTURE_FLEA	
FLYCAPTURE_SCORPION	
FLYCAPTURE_SONY	
FLYCAPTURE_TYPHOON	
FLYCAPTURE_UNKNOWN	

FlyCaptureCameraSerialNumber

The type used to store the serial number uniquely identifying a FlyCapture camera.

Declaration

```
typedef unsigned long FlyCaptureCameraSerialNumber
```

FlyCaptureCameraType

An enumeration used to describe the different camera color configurations.

Declaration

```
enum FlyCaptureCameraType
{
   FLYCAPTURE_BLACK_AND_WHITE,
   FLYCAPTURE_COLOR

}
```

Elements

FLYCAPTURE_BLACK_AND_WHITE	black and white system.
FLYCAPTURE_COLOR	color system.

FlyCaptureColorMethod

An enumeration used to describe the different color processing method.

Declaration

```
enum FlyCaptureColorMethod
{
```

```
   FLYCAPTURE_DISABLE,
   FLYCAPTURE_EDGE_SENSING,
   FLYCAPTURE_NEAREST_NEIGHBOR,
   FLYCAPTURE_NEAREST_NEIGHBOR_FAST,
   FLYCAPTURE_RIGOROUS,

}
```

Elements

FLYCAPTURE_DISABLE	Disable color processing.
FLYCAPTURE_EDGE_SENSING	Edge sensing de-mosaicing. This is the most accurate method that can still keep up with the camera's frame rate.
FLYCAPTURE_NEAREST_NEIGHBOR	Nearest neighbor de-mosaicing. This algorithm is significantly faster than edge sensing, at the cost of accuracy.
FLYCAPTURE_NEAREST_NEIGHBOR_FAST	Faster, less accurate nearest neighbor de-mosaicing.
FLYCAPTURE_RIGOROUS	Rigorous de-mosaicing. This provides the best quality colour reproduction. This method is so processor intensive that it might not keep up with the camera's frame rate. Best used for offline processing where accurate colour reproduction is required.

Remarks

This is only relevant for cameras that do not do onboard color processing, such as the Dragonfly. The FLYCAPTURE_RIGOROUS method is very slow and will not keep up with high frame rates.

FlyCaptureContext

Context pointer for the PGRFlyCapture library.

Declaration

```
typedef void* FlyCaptureContext
```

FlyCaptureError

The error codes returned by the functions in this library.

Declaration

```
enum FlyCaptureError
{
   FLYCAPTURE_OK,
   FLYCAPTURE_FAILED,
   FLYCAPTURE_INVALID_ARGUMENT,
   FLYCAPTURE_INVALID_CONTEXT,
   FLYCAPTURE_NOT_IMPLEMENTED,
   FLYCAPTURE_ALREADY_INITIALIZED,
```

```
    FLYCAPTURE_ALREADY_STARTED,
    FLYCAPTURE_CALLBACK_NOT_REGISTERED,
    FLYCAPTURE_CALLBACK_ALREADY_REGISTERED,
    FLYCAPTURE_CAMERACONTROL_PROBLEM,
    FLYCAPTURE_COULD_NOT_OPEN_FILE,
    FLYCAPTURE_COULD_NOT_OPEN_DEVICE_HANDLE,
    FLYCAPTURE_MEMORY_ALLOC_ERROR,
    FLYCAPTURE_NO_IMAGE,
    FLYCAPTURE_NOT_INITIALIZED,
    FLYCAPTURE_NOT_STARTED,
    FLYCAPTURE_MAX_BANDWIDTH_EXCEEDED,
    FLYCAPTURE_NON_PGR_CAMERA,
    FLYCAPTURE_INVALID_MODE,
    FLYCAPTURE_ERROR_UNKNOWN,
    FLYCAPTURE_INVALID_CUSTOM_SIZE,
    FLYCAPTURE_TIMEOUT,
    FLYCAPTURE_TOO_MANY_LOCKED_BUFFERS,
    FLYCAPTURE_VERSION_MISMATCH,
    FLYCAPTURE_DEVICE_BUSY,

}
```

Elements

FLYCAPTURE_ALREADY_INITIALIZED	Device already initialized.
FLYCAPTURE_ALREADY_STARTED	Grabbing has already been started.
FLYCAPTURE_CALLBACK_ALREADY_REGISTERED	Callback is already registered
FLYCAPTURE_CALLBACK_NOT_REGISTERED	Callback is not registered
FLYCAPTURE_CAMERACONTROL_PROBLEM	Problem controlling camera.
FLYCAPTURE_COULD_NOT_OPEN_DEVICE_HANDLE	Failed to open a device handle.
FLYCAPTURE_COULD_NOT_OPEN_FILE	Failed to open file.
FLYCAPTURE_DEVICE_BUSY	The camera responded that it is currently busy.
FLYCAPTURE_ERROR_UNKNOWN	Unknown error.
FLYCAPTURE_FAILED	General failure.
FLYCAPTURE_INVALID_ARGUMENT	Invalid argument passed.
FLYCAPTURE_INVALID_CONTEXT	Invalid context passed.
FLYCAPTURE_INVALID_CUSTOM_SIZE	Invalid custom size.
FLYCAPTURE_INVALID_MODE	Invalid video mode or framerate passed or retrieved.
FLYCAPTURE_MAX_BANDWIDTH_EXCEEDED	Request would exceed maximum bandwidth.
FLYCAPTURE_MEMORY_ALLOC_ERROR	Memory allocation error
FLYCAPTURE_NON_PGR_CAMERA	Attached camera is not a PGR camera.

FLYCAPTURE_NOT_IMPLEMENTED	Function not implemented.
FLYCAPTURE_NOT_INITIALIZED	Device not initialized.
FLYCAPTURE_NOT_STARTED	flycaptureStart() not called.
FLYCAPTURE_NO_IMAGE	flycaptureGrabImage() not called.
FLYCAPTURE_OK	Function completed successfully.
FLYCAPTURE_TIMEOUT	Operation timed out.
FLYCAPTURE_TOO_MANY_LOCKED_BUFFERS	Too many image buffers are locked by the user.
FLYCAPTURE_VERSION_MISMATCH	There is a version mismatch between one of the interacting modules: pgrflycapture.dll, pgrflycaptureregui.dll, and the camera driver.

FlyCaptureFrameRate

Enum describing different framerates.

Declaration

```
enum FlyCaptureFrameRate
{
   FLYCAPTURE_FRAMERATE_1_875,
   FLYCAPTURE_FRAMERATE_3_75,
   FLYCAPTURE_FRAMERATE_7_5,
   FLYCAPTURE_FRAMERATE_15,
   FLYCAPTURE_FRAMERATE_30,
   FLYCAPTURE_FRAMERATE_50,
   FLYCAPTURE_FRAMERATE_60,
   FLYCAPTURE_NUM_FRAMERATES,
   FLYCAPTURE_FRAMERATE_CUSTOM,
   FLYCAPTURE_FRAMERATE_ANY,

}
```

Elements

FLYCAPTURE_FRAMERATE_15	15 fps.
FLYCAPTURE_FRAMERATE_1_875	1.875 fps. (Frames per second)
FLYCAPTURE_FRAMERATE_30	30 fps.
FLYCAPTURE_FRAMERATE_3_75	3.75 fps.
FLYCAPTURE_FRAMERATE_50	50 fps.
FLYCAPTURE_FRAMERATE_60	60 fps.
FLYCAPTURE_FRAMERATE_7_5	7.5 fps.
FLYCAPTURE_FRAMERATE_ANY	Hook for “any usable frame rate.”
FLYCAPTURE_FRAMERATE_CUSTOM	Custom frame rate. Used with custom image size

	functionality.
FLYCAPTURE_NUM_FRAMERATES	Number of possible camera frame rates.

FlyCaptureImage

This structure represents the necessary information about an image returned from the API.

Declaration

```
struct FlyCaptureImage
{
   int iRows;
   int iCols;
   int iRowInc;
   FlyCaptureVideoMode  videoMode;
   FlyCaptureTimestamp  timeStamp;
   unsigned char*       pData;
   bool           bStippled;
   unsigned long  ulReserved[ 7 ];

}
```

Elements

bStippled	If the returned image is Y8 or Y16, this flag indicates whether it is a greyscale or stippled (bayer tiled) image. In modes other than Y8 or Y16, this flag has no meaning.
iCols	Columns, in pixels, of the image.
iRowInc	Row increment. The number of bytes per row.
iRows	Rows, in pixels, of the image.
pData	Pointer to the actual image data.
timeStamp	Timestamp of this image.
ulReserved	Reserved for future use.
videoMode	Video mode that this image was captured with.

Remarks

The size of the image buffer is iRowInc * iRows, and depends on the current video mode.

FlyCaptureImagePlus

A wrapper for FlyCaptureImage that provides advanced functionality.

Declaration

```
struct FlyCaptureImagePlus
{
```

```
   FlyCaptureImage   image;

   unsigned int      uiSeqNum;

   unsigned int      uiBufferIndex;

   unsigned long     ulReserved[ 8 ];

}
```

Elements

image	The FlyCaptureImage that this FlyCaptureImagePlus sturcture is wrapping. Please see documentation in pgrflycapture.h.
uiBufferIndex	The internal buffer index that the image buffer contained in the FlyCaptureImage corresponds to. For functions that lock the image, this number must be passed back to the "unlock" function. If flycaptureInitializePlus() was called, this number corresponds to the position of the buffer in the buffer array passed in.
uiSeqNum	The sequence number of the image. This number is generated in the driver and sequential images should have a difference of one. If the difference is greater than one, it indicates the number of missed images since the last lock image call.
ulReserved	Reserved for future use.

FlyCaptureInfo

A record used in querying the FlyCapture Camera properties.

Declaration

```
struct FlyCaptureInfo
{
   FlyCaptureCameraSerialNumber  SerialNumber;
   FlyCaptureCameraType          CameraType;
   FlyCaptureCameraModel         CameraModel;
   char                          pszModelString[ 512 ];

}
```

Elements

CameraModel	Camera model.
CameraType	type of CCD (color or b&w).
SerialNumber	camera serial number.
pszModelString	Null-terminated camera model string for attached camera.

FlyCaptureOutputFormat

An enumeration used to indicate the desired output format form image conversion.

Declaration

```
enum FlyCaptureOutputFormat
{
   FLYCAPTURE_OUTPUT_BGR,
   FLYCAPTURE_OUTPUT_BGRU,

}
```

Elements

FLYCAPTURE_OUTPUT_BGR	A 24 bit per pixel BGR image.
FLYCAPTURE_OUTPUT_BGRU	A 32 bit per pixel BGRU image.

FlyCaptureProperty

An enumeration of the different camera properties that can be set via the API.

Declaration

```
enum FlyCaptureProperty
{
   FLYCAPTURE_BRIGHTNESS,
   FLYCAPTURE_AUTO_EXPOSURE,
   FLYCAPTURE_SHARPNESS,
   FLYCAPTURE_WHITE_BALANCE,
   FLYCAPTURE_HUE,
   FLYCAPTURE_SATURATION,
   FLYCAPTURE_GAMMA,
   FLYCAPTURE_IRIS,
   FLYCAPTURE_FOCUS,
   FLYCAPTURE_ZOOM,
   FLYCAPTURE_PAN,
   FLYCAPTURE_TILT,
   FLYCAPTURE_SHUTTER,
   FLYCAPTURE_GAIN,
   FLYCAPTURE_SOFTWARE_WHITEBALANCE,

}
```

Elements

FLYCAPTURE_AUTO_EXPOSURE	The auto exposure property of the camera.
FLYCAPTURE_BRIGHTNESS	The brightness property of the camera.
FLYCAPTURE_FOCUS	The focus property of the camera.
FLYCAPTURE_GAIN	The gain property of the camera.
FLYCAPTURE_GAMMA	The gamma property of the camera.
FLYCAPTURE_HUE	The hue property of the camera.

FLYCAPTURE_IRIS	The iris property of the camera.
FLYCAPTURE_PAN	The pan property of the camera.
FLYCAPTURE_SATURATION	The saturation property of the camera.
FLYCAPTURE_SHARPNESS	The sharpness property of the camera.
FLYCAPTURE_SHUTTER	The shutter property of the camera.
FLYCAPTURE_SOFTWARE_WHITEBALANCE	Software white balance property. Use this to manipulate the values for software whitebalance. This is only applicable to cameras that do not do onboard colour processing. On these cameras, hardware white balance is disabled.
FLYCAPTURE_TILT	The tilt property of the camera.
FLYCAPTURE_WHITE_BALANCE	The hardware white balance property of the camera.
FLYCAPTURE_ZOOM	The zoom property of the camera.

Remarks
A lot of these properties are included only for completeness and future expandability, and will have no effect on a PGR single lens Camera.

FlyCaptureStippledFormat

An enumeration used to indicate the format of the stippled images passed into a destippling function.

Declaration

```
enum FlyCaptureStippledFormat
{
   FLYCAPTURE_STIPPLEDFORMAT_BGGR,
   FLYCAPTURE_STIPPLEDFORMAT_GBRG,
   FLYCAPTURE_STIPPLEDFORMAT_GRBG,
   FLYCAPTURE_STIPPLEDFORMAT_RGGB,
   FLYCAPTURE_STIPPLEDFORMAT_DEFAULT

}
```

Elements

FLYCAPTURE_STIPPLEDFORMAT_BGGR	indicates a BGGR image.
FLYCAPTURE_STIPPLEDFORMAT_DEFAULT	indicates the default stipple format for the Dragonfly or Firefly.
FLYCAPTURE_STIPPLEDFORMAT_GBRG	indicates a GBRG image.
FLYCAPTURE_STIPPLEDFORMAT_GRBG	indicates a GRBG image.
FLYCAPTURE_STIPPLEDFORMAT_RGGB	indicates a RGGB image.

Remarks

This is only relevant for cameras that do not do onboard color processing, such as the Dragonfly. The four letters of the enum value correspond to the "top left" 2x2 section of the stippled image. For example, the first line of a BGGR image image will be BGBGBG..., and the second line will be GRGRGR....

FlyCaptureTimestamp

This structure defines the format by which time is represented in the PGRFlycapture SDK. The ulSeconds and ulMicroSeconds values represent the absolute system time when the image was captured. The ulCycleSeconds and ulCycleCount are higher-precision values that have been propagated up from the 1394 bus. The ulCycleSeconds value will wrap around after 128 seconds. The ulCycleCount represents the 1/8000 second component. Use these two values when synchronizing grabs between two computers that may not have precisely synchronized system timers.

Declaration

```
struct FlyCaptureTimestamp
{
   unsigned long ulSeconds;
   unsigned long ulMicroSeconds;
   unsigned long ulCycleSeconds;
   unsigned long ulCycleCount;

}
```

Elements

ulCycleCount	The cycle time count. 0-7999.
ulCycleSeconds	The cycle time seconds. 0-127.
ulMicroSeconds	The microseconds component.
ulSeconds	The number of seconds since the epoch.

FlyCaptureVideoMode

Enum describing different video modes.

Declaration

```
enum FlyCaptureVideoMode
{
   FLYCAPTURE_VIDEOMODE_160x120YUV444,
   FLYCAPTURE_VIDEOMODE_320x240YUV422,
   FLYCAPTURE_VIDEOMODE_640x480YUV411,
   FLYCAPTURE_VIDEOMODE_640x480YUV422,
   FLYCAPTURE_VIDEOMODE_640x480RGB,
   FLYCAPTURE_VIDEOMODE_640x480Y8,
   FLYCAPTURE_VIDEOMODE_640x480Y16,
   FLYCAPTURE_VIDEOMODE_800x600Y8,
   FLYCAPTURE_VIDEOMODE_1024x768Y8,
   FLYCAPTURE_VIDEOMODE_1024x768Y16,
```

```
    FLYCAPTURE_VIDEOMODE_1280x960Y8,
    FLYCAPTURE_VIDEOMODE_1600x1200Y8,
    FLYCAPTURE_VIDEOMODE_640x240Y8,
    FLYCAPTURE_VIDEOMODE_SCORPION1,
    FLYCAPTURE_NUM_VIDEOMODES,
    FLYCAPTURE_VIDEOMODE_CUSTOM,
    FLYCAPTURE_VIDEOMODE_ANY,

}
```

Elements

FLYCAPTURE_NUM_VIDEOMODES	Number of possible video modes.
FLYCAPTURE_VIDEOMODE_1024x768Y16	800x600 16-bit greyscale.
FLYCAPTURE_VIDEOMODE_1024x768Y8	1024x768 8-bit greyscale.
FLYCAPTURE_VIDEOMODE_1280x960Y8	1280x960 8-bit greyscale or bayer titled color image.
FLYCAPTURE_VIDEOMODE_1600x1200Y8	1600x1200 8-bit greyscale or bayer titled color image.
FLYCAPTURE_VIDEOMODE_160x120YUV444	160x120 YUV444.
FLYCAPTURE_VIDEOMODE_320x240YUV422	320x240 YUV422.
FLYCAPTURE_VIDEOMODE_640x240Y8	640x240 8-bit greyscale image. Only supported on Dragonfly Cameras.
FLYCAPTURE_VIDEOMODE_640x480RGB	640x480 24-bit RGB.
FLYCAPTURE_VIDEOMODE_640x480Y16	640x480 16-bit greyscale or bayer tiled color image.
FLYCAPTURE_VIDEOMODE_640x480Y8	640x480 8-bit greyscale or bayer tiled color image.
FLYCAPTURE_VIDEOMODE_640x480YUV411	640x480 YUV411.
FLYCAPTURE_VIDEOMODE_640x480YUV422	640x480 YUV422.
FLYCAPTURE_VIDEOMODE_800x600Y8	800x600 8-bit greyscale or bayer tiled color image.
FLYCAPTURE_VIDEOMODE_ANY	Hook for “any usable video mode.”
FLYCAPTURE_VIDEOMODE_CUSTOM	Custom video mode. Used with custom image size functionality.
FLYCAPTURE_VIDEOMODE_SCORPION1	(temporary) Scorpion mode.

flycaptureGetImageFilters

Retrieves the currently active filters. The returned number is a bitmap corresponding to the FLYCAPTURE_IMAGE_FILTER_* values.

Declaration

```
FlyCaptureError
flycaptureGetImageFilters(
                          FlyCaptureContext   context,
                          unsigned int*       puiFilters )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
puiFilters	The filter bitmap is returned in this value.

Return Value

A FlyCaptureError indicating the success or failure of the function.

flycaptureInitializePlus

Identical behaviour to flycaptureInitialize(), except that the user has the option of specifying the number of buffers to use, and optionally allocate those buffers outside the library.

Declaration

```
FlyCaptureError
flycaptureInitializePlus(
                         FlyCaptureContext   context,
                         unsigned int        uiBusIndex,
                         unsigned int        uiNumBuffers,
                         unsigned char**     arpBuffers  )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
uiBusIndex	The zero-based device index of the camera to be initialized.
uiNumBuffers	The number of buffers to expect or allocate. This value must not be greater than 128.
arpBuffers	An array of pointers to buffers. If this argument is NULL the library will allocate and free the buffers internally, otherwise the caller is responsible for allocation and deallocation. No boundary checking is done on these images, if you are supplying your own buffers, they must be large enough to hold the largest image you are expecting.

Return Value

A FlyCaptureError indicating the success or failure of the function.

Remarks

If you wish to use the camera serial number to initialize, or you don't care about the number of buffers being allocated, use either of the other initialize methods in pgrflycapture.h

flycaptureLockLatest

Lock the "latest" image that has not been seen. If there is an unseen image waiting, this function will return immediately with that image, otherwise it will block until the next image has been received. The difference in the sequence numbers of images returned by consecutive calls to this function indicates the number of missed images between calls.

Declaration

```
FlyCaptureError
flycaptureLockLatest(
                    FlyCaptureContext    context,
                    FlyCapturePlus*      pimage   )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
pimage	The returned FlyCaptureImagePlus.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
flycaptureUnlock() must be called using the buffer index returned in pimage when processing on this image has been completed. This function behaves identically to flycaptureGrabImage(), except it doesn't implictly unlock the previously seen image first. The camera must have been started using flycaptureStart() in order for this function to succeed.

flycaptureLockNext

Lock the "next" image that has not been seen. Provided that the previous image processing time is not greater than the time taken for the camera to transmit images to the available unlocked buffers, this function can be called repeatedly to guarantee that each image will be seen. If the camera has not finished transmitting the next image, this function will block. Users can verify image sequentiality by comparing sequence numbers of sequential images.

Declaration

```
FlyCaptureError
flycaptureLockNext(
                   FlyCaptureContext      context,
                   FlyCapturePlus*        pimage   )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
pimage	The returned FlyCaptureImagePlus.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks

flycaptureUnlock() must be called using the buffer index returned in pimage when processing on this image has been completed. The camera must have been started using flycaptureStartLockNext() for this function to succeed.

flycaptureReadRegisterBlock

Provides block-read (asynchronous) access to the entire register space of the camera.

Declaration

```
FlyCaptureError
flycaptureReadRegisterBlock(
                            FlyCaptureContext   context,
                            unsigned short      usAddrHigh,
                            unsigned long       ulAddrLow,
                            unsigned long*      pulBuffer,
                            unsigned long       ulLength )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
usAddrHigh	The top 16 bits of the 48-bit absolute address to read.
ulAddrLow	The bottom 32 bits of the 48-bit absolute addresss to read.
pulBuffer	The buffer that will receive the data. Must be of size ulLength.
ulLength	The length, in quadlets, of the block to read.

Return Value

A FlyCaptureError indicating the success or failure of the function.

flycaptureSetImageFilters

Sets the active filters. The returned number is a bitmap corresponding to the FLYCAPTURE_IMAGE_FILTER_* values.

Declaration

```
FlyCaptureError
flycaptureSetImageFilters(
                          FlyCaptureContext   context,
                          unsigned int        uiFilters )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
uiFilters	The filters to set. Use FLYCAPTURE_IMAGE_FILTER_NONE to disable image filtering.

Return Value

A FlyCaptureError indicating the success or failure of the function.

flycaptureStartLockNext

Starts the camera streaming and initializes the library for "grab next" functionality. This function needs to used instead of flycaptureStart() for the following "lock next" functions.

Declaration

```
FlyCaptureError
flycaptureStartLockNext(
                        FlyCaptureContext    context,
                        FlyCaptureVideoMode  videoMode,
                        FlyCaptureFrameRate  frameRate   )
```

Parameters

context	The context associated with the camera to be started.
videoMode	The video mode to start the camera in.
frameRate	The frame rate to start the camera at.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
For "lock latest" functionality, use flycaptureStart() and the flycaptureLockLatest().

flycaptureSyncForLockNext

Takes in an array of contexts attached to multiple cameras that are synchronized in hardware and assures that the next time lockNext() is called for all contexts, the images locked will correspond to one another. Note that this function only needs to be called once after the contexts have been started. The contexts should be started in the same order that they are listed in arContexts before this function is called.

Declaration

```
FlyCaptureError
flycaptureSyncForLockNext(
                          FlyCaptureContext* arContexts,
                          unsigned int       uiContexts  )
```

Parameters

arContexts	An array of contexts attached to the cameras to synchronize.
uiContexts	The number of contexts in arContext.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Remarks
This function operates by skipping the appropriate number of images in contexts that were started "after" the reference context (position 0 in the array). If this function fails it does not necessarily mean the cameras are out of sync. This is still experimental.

flycaptureUnlock

Returns a buffer into the pool to be filled by the camera driver. This must be called for each image locked using the above lock functions after processing on that image has been completed.

Declaration

```
FlyCaptureError
flycaptureUnlock(
                FlyCaptureContext  context,
                unsigned int       uiBufferIndex  )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.
uiBufferIndex	The buffer to unlock.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureUnlockAll

Unlocks all locked images. This is equivalant to maintaining a list of locked buffers and calling flycaptureUnlock() for each.

Declaration

```
FlyCaptureError
flycaptureUnlockAll(
                   FlyCaptureContext  context )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.

Return Value
A FlyCaptureError indicating the success or failure of the function.

flycaptureWriteRegisterBlock

Provides block-write (asynchronous) access to the entire register space of the camera.

Declaration

```
FlyCaptureError
flycaptureWriteRegisterBlock(
                            FlyCaptureContext      context,
                            unsigned short         usAddrHigh,
                            unsigned long          ulAddrLow,
                            const unsigned long*   pulBuffer,
                            unsigned long          ulLength )
```

Parameters

context	The FlyCaptureContext associated with the camera to be queried.

usAddrHigh	The top 16 bits of the 48-bit absolute address to write.
ulAddrLow	The bottom 32 bits of the 48-bit absolute addresss to write.
pulBuffer	The buffer that contains the data to be written.
ulLength	The length, in quadlets, of the block to write.

Return Value
A FlyCaptureError indicating the success or failure of the function.

Macros

FLYCAPTURE_BUS_INVALID

Declaration

```
#define  FLYCAPTURE_BUS_INVALID  0
```

Remarks
Depricated message value for old bus callback mechanism. Do not use. This will be removed in future versions of this SDK.

FLYCAPTURE_BUS_VALID

Declaration

```
#define  FLYCAPTURE_BUS_VALID    1
```

Remarks
Depricated message value for old bus callback mechanism. Do not use. This will be removed in future versions of this SDK.

FLYCAPTURE_MESSAGE_BUS_RESET

Declaration

```
#define FLYCAPTURE_MESSAGE_BUS_RESET      0x02
```

Remarks
A message returned from the bus callback mechanism indicating a bus reset.

FLYCAPTURE_MESSAGE_DEVICE_ARRIVAL

Declaration

```
#define FLYCAPTURE_MESSAGE_DEVICE_ARRIVAL 0x03
```

Remarks

A message returned from the bus callback mechanism indicating a device has arrived on the bus. ulParam contains the serial number of the device.

FLYCAPTURE_MESSAGE_DEVICE_REMOVAL

Declaration

```
#define FLYCAPTURE_MESSAGE_DEVICE_REMOVAL 0x04
```

Remarks
A message returned from the bus callback mechanism indicating a device has been removed from the bus. ulParam contains the serial number of the device.

PGRFLYCAPTURE_API

Declaration

```
#define PGRFLYCAPTURE_API __declspec( dllexport )
```

PGRFLYCAPTURE_VERSION

The version of the library.

Declaration

```
#define PGRFLYCAPTURE_VERSION 10401023
```

Chapter 6: Advanced Features

This chapter details the more advanced features of the PGR FlyCapture SDK.

PGRFlyCapturePlus Functionality

Extended functionality has recently been exposed in the new PGRFlyCapturePlus.h header file. PGRFlyCapturePlus functions provide advanced users with greater control over the way images are handled once they are streamed via DMA from the camera to the PC, as well as some additional functionality such as register block reads and writes.

The most significant additions are the flycaptureLockLatest(), flycaptureLockNext(), and flycaptureSyncForLockNext() functions, and the ability of the user to define more than the standard four memory buffers automatically allocated by the API. Put simply, flycaptureLockLatest() returns to the user a pointer to the most recent image received by the PC; in this respect it is identical to flycaptureGrabImage2().flycaptureLockNext(), however, returns a pointer to the next image that has not yet been seen by the user. This is very useful when used with a large number of custom buffers, as it allows the user to lock an image, do some processing, and grab the very next image in the sequence without taking the risk that this next image has been overwritten by consecutive images being streamed in from the camera.

The Custom Buffers Example in the SDK is an excellent example of how to use the PGRFlyCapturePlus functionality.

Multiple Cameras on One Bus

The PGR FlyCapture system supports having multiple cameras on one bus. With the Firefly™ and Firefly2 cameras, the grabbing of images will not be synchronized. However, with the Dragonfly™ and Scorpion, the grabbing of images will automatically be synchronized at the hardware level using timing information provided by the 1394 bus.

There is no master camera to which the other cameras are synced. The way synchronization is achieved is that each camera is constantly receiving the 1394 bus cycle timer information, and a register on the camera contains this cycle time information (query CYCLE_TIME register FF100200h using the *Camera Control* > *Extended* tab of FlyCap). The camera firmware is designed to grab at selected intervals of the 1394 cycle time, so each camera on the bus has the same cycle time info and can grab at the same cycle time interval.

NOTES:

- Automatic synchronization of multiple cameras is only supported in recent camera firmware versions.
- Cameras on different 1394 FireWire buses are not automatically synchronized. This requires a PGR Sync Unit.

Timestamping Mechanisms

The user has access to three different timestamps that can provide information on when, or approximately when, a specific image was grabbed. Two are accessed directly via the PGR FlyCapture API, and one from the image itself.

1. The first one is generated from the PC's clock; captured images are timestamped once the last packet arrives at the PC. This gives a rough time estimate and is generally not recommended for precision purposes. This timestamp can be accessed through the FlyCaptureTimeStamp.ulSeconds and FlyCaptureTimeStamp.ulMicroSeconds members of the FlyCaptureImage structure. For more information on these structures, please consult the API Reference Manual.
2. Another one is based on the cycle timer (which increments at 8kHz) and again is stamped when the image arrives at the PC. These cycle times are not epoch-based and can only provide relative, not absolute, timing. The cycle seconds range from 0-127, and the cycle count ranges from 0-7999 i.e. the cycle timer wraps aroung after 128 seconds. This timestamp can be accessed through the FlyCaptureTimeStamp.ulCycleSeconds and FlyCaptureTimeStamp.ulCycleCount members of the FlyCaptureImage structure. For more information on these structures, please consult the API documentation.
3. The recommended timestamp mechanism for PGR cameras is the image timestamp. It takes the cycle timer at the time that the shutter was closed. This is the most accurate of the timestamps. In order to access this, you need to turn on the Frame_Timestamp register 12F8h, and then use a function to read the first 4 bytes of the image (the timestamp is located in the first 4 pixels of the image).

Relation Between Color Processing and Bits Per Pixel

When selecting different stipple formats (with the flycaptureSetColorTileFormat() function), this does not affect the programming of the CCD sensor's Bayer tile mapping. The Bayer tiling on the chip is fixed - when setting different stipple formats, this simply changes the interpretation of the image pixels for color processing.

When color processing is disabled via the flycaptureSetColorProcessingMethod() function, an 8-bit per pixel image is delivered (unless the camera is in 16-bit mode, or Y16, then it's 16 bits/pixel), so for a 640x480 image, 640 * 480 * 1 byte of memory is used. Also, when calling any of the flycaptureStippledTo*() functions, the 8 bit image is upsampled to 24 or 32 bits. For example, the flycaptureStippledtoBGR() function yields 24-bit BGR (in Windows format, where R and B are reversed), while flycaptureStippledToBGRU() yields 32-bit BGRU pixels (where U stands for Unused - although it usually stands for Alpha, but the byte doesn't contain any meaningful information).

The call to flycaptureGrabImage() always returns the raw 8-bit Bayer stippled image for Dragonfly's (unless the camera is in Y16 mode), then as a secondary processing step, a call to flycaptureStippledToBGR*() retrieves the color buffer.

Chapter 7: Installation Troubleshooting

This chapter will help to isolate problems encountered while installing your PGR FlyCapture system and assist in solving them.

***NOTE**: This chapter does not cover problems encountered while using the FlyCapture API to develop your own application, or problems specific to certain cameras. It is intended to troubleshoot software and driver installation issues only.*

For help with non-installation related problems, please try the following:

1. *Consult our on-line knowledge base at http://www.ptgrey.com/support/kb for answers to some of the most common support questions. It is constantly updated, expanded, and refined to ensure that you have access to current information.*

2. *Download the latest product versions from our downloads site at http://www.ptgrey.com/support/downloads. Recent versions may include bug fixes that solve problems you are encountering.*

3. *Contact our Technical Support group by completing our on-line support request form at http://www.ptgrey.com/support/contact.*

Driver Installation Problems

When you plug the camera in, one of two expected outcomes will occur: the Device Manager window will refresh itself, showing details of your camera e.g. *Imaging Devices > PGR Dragonfly*; or the *Found New Hardware Wizard* will start.

If this does not occur, ensure you are running one of the supported operating systems. If the Device Manager does not list your camera as a "PGR Dragonfly", "PGR Firefly", or "PGR Camera" under Imaging Devices, your camera may be using the wrong set of drivers. If the camera is plugged in and listed in Device Manager, but is not listed correctly, right-click the camera, select *Properties*, and select the *Driver* tab. If you have already installed the PGR software and drivers, select *Update Driver*, which starts the *Upgrade Device Wizard*. Repeat the driver installation steps outlined in the previous section *Connect the Camera to Your Computer.*

If the camera is still incorrectly listed, consult our on-line knowledge base for additional troubleshooting suggestions.

Appendix A: Camera Assembly

This appendix details some of the assembly instructions for parts of your PGR IEEE-1394 single-lens camera.

Assembling an Extended Head Camera

If you have purchased an extended head camera – which is useful when a smaller profile camera head is needed – it will come in three pieces as shown in the figure below.

> ***NOTE**: The ends of recent models of the flexible ribbon cable are blue, not black.*

To assemble the extended camera:

1. Make sure the camera is unplugged and no power is going to the camera, and take appropriate steps to limit any electrostatic discharge.

2. Lift the brown tab on the camera body.

3. Insert the extending cable, with the black (blue) side of the cable facing the brown tab on the camera body. Close the brown tab.

4. Lift the brown tab on the camera head.

5. Insert the other end of the extending cable, with the black side of the cable facing the brown tab, into the camera head. Close the brown tab.

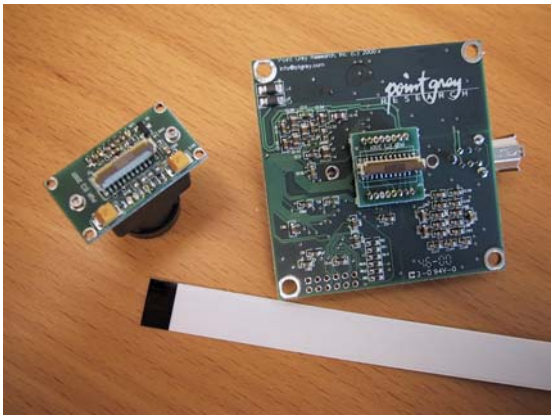


Figure 16: Unassembled extended head camera

Figure 17: Assembled extended head camera

Mounting the Tripod Bracket

You will need a 3/32” hex key in addition to the tripod bracket, the spacers, and the mounting screws.

1. Place the bracket face down (with the edge that will attach to the tripod facing up).
2. Place the spacers on the four corners of the bracket
3. Place the camera face down on the spacers.
4. Insert and fasten the screws to the bracket.
5. See the figures below to ensure that the bracket has been mounted correctly.

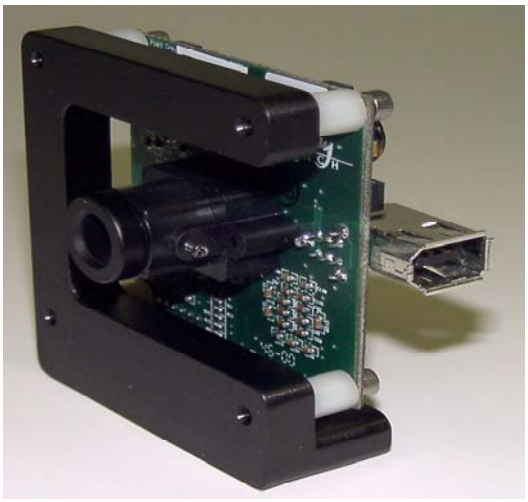


Figure 18: Firefly camera with mounting bracket - Front

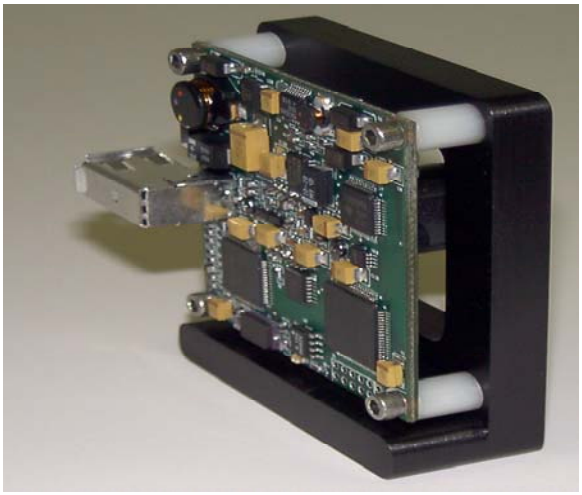


Figure 19: Firefly camera with mounting bracket - Back

Appendix B: Camera Properties

This appendix details some of the physical properties and general functionality of your PGR IEEE-1394 single-lens camera.

***NOTE**: Detailed technical information regarding the Dragonfly™ camera can be found in the Dragonfly Technical Reference, included with the PGR FlyCapture SDK documentation.*

Firefly2, Dragonfly and Scorpion Specifications

Although the Dragonfly, Firefly2 and Scorpion are all single-lens IEEE-1394 FireWire cameras, there are a variety of differences between the three.

Feature	**Firefly2**	**Dragonfly**	**Scorpion SYMAGERY**
Image Sensor Type	CCD	CCD	CMOS
Electronic Shutter	Frame	Frame	Rolling
Monochrome Sensor	N/A	640x480 - ICX084AL 640x480 - ICX424AL 1024x768 - ICX204AL	VCA 1281M
Color Sensor	640x480 – ICX098BQ	640x480 - ICX084AK 640x480 - ICX424AQ 1024x768 - ICX204AK	VCA 1281C
Optical Format	1/4"	1/3"	2/3"
Resolution	640x480	640x480 1024x768	1280x1024
Supported Frame Rates	FPS: 30, 15, 7.5, 3.75	1024x768 FPS: 15, 7.5, 3.75 640x480 FPS: 30, 15, 7.5, 3.75	1280x960 FPS: 15, 7.5, 3.75 1280x960 FPS: 30 in DCAM Format 7 640x480 FPS: 120 in DCAM Format 7
Sub-Sampling	Yes 320x240 FPS: 160x120 FPS:	Yes 640x240 FPS: 50	No
ROI	No	Yes (Limited)	Yes (Available with speed up. Increase proportional to reduction in # of pixels)
Supported Lenses	Microlens C/CS lens	Microlens C/CS lens	C/CS lens
Extended Sensor Form Factor	No	Yes	No
Color Processing Performed On-Board the Camera	Yes	No	No
Color Processing Performed On the PC	Possible	Yes	Yes
Multiple Color Processing Algorithms Available	No (for on-board the camera)	Yes	Yes
On Board Image Processing	Yes (Sharpening, Gamma	No	TBD

	Correction, etc. See chipset specification)		
General Purpose IO Pins for External Triggering / Strobe	Limited (see chipset specification)	Yes (Available with 4 PINS)	Yes (Available with 10 PINS)
Automatic Synchronization Between Multiple Cameras on the Same Firewire Bus	No	Yes	Possible (dependent on frame rate and resolution - see related article)
S/N Ratio	>40dB	60dB	40dB
ADC	8-bit	8/10-bit	8-bit
On Board Memory			16MB
Chipset	TSB15LV01	FPGA & Microcontroller	FPGA & Microcontroller
Upgradeable Firmware via FireWire	No	Yes	Yes
1394 DCAM Version Compliancy	v1.04	v1.30	v1.30

Physical Description of the Firefly2 Camera Module

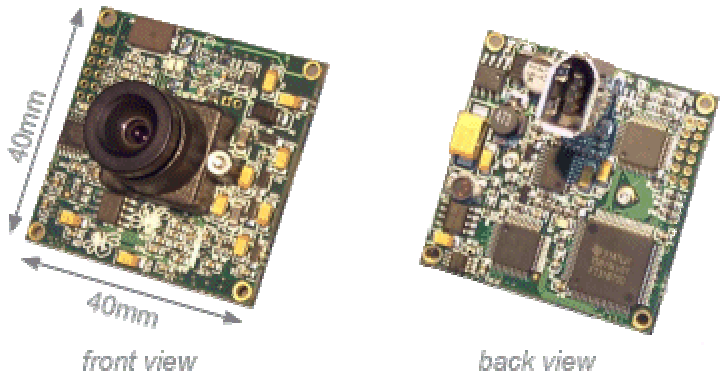


Figure 20: The Firefly2 camera module

- The Firefly™ camera comes with three different lenses: 4mm, 6mm (already on the camera) and 8mm.
- The set screw on the microlens holder is used to hold the lens in focus. To remove the lens or adjust the focus, one must loosen the set screw first, and then adjust the lens.
- The 1394 connector connects to a 1394 (firewire) cable and provides the camera with both power and a connection to your computer. No additional power source is required to operate the camera.

Scorpion Camera Drawings and Dimensions

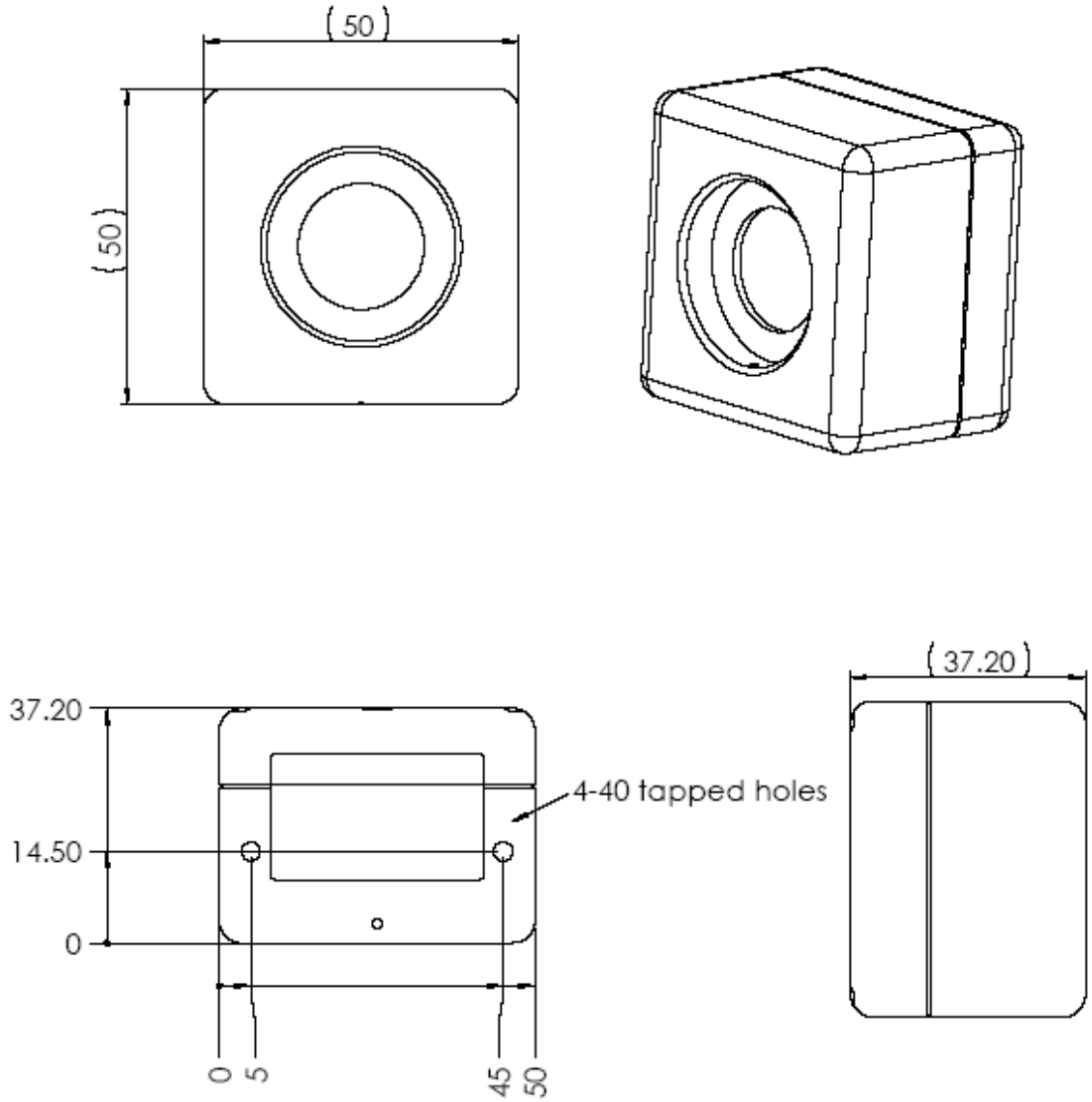


Figure 21: Scorpion camera dimensions

Scorpion General Purpose Input/Output Pins Diagram

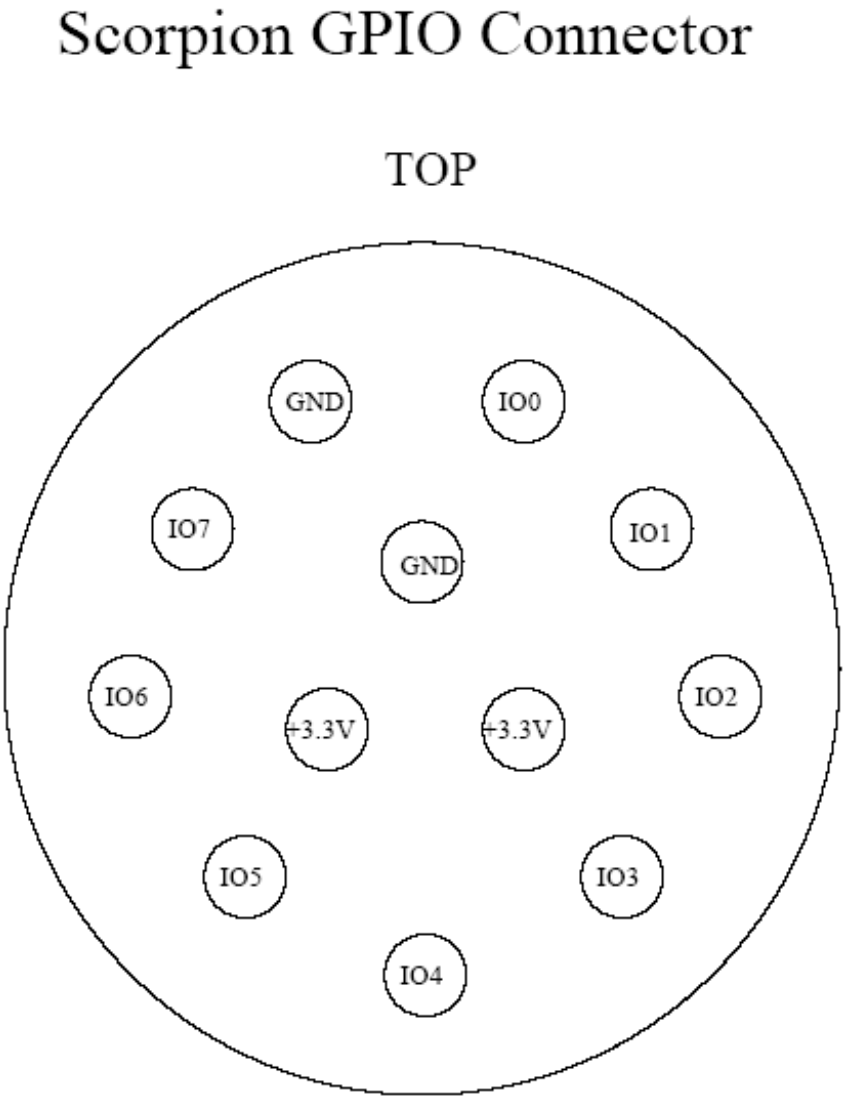


Figure 22: Scorpion GPIO connector schematic

Contacting Point Grey Research Inc.

For any questions, concerns or comments please contact us via the following methods:

Email:

For all general questions about Point Grey Research or our products contact info@ptgrey.com.

For all specific questions about our products or for quotes or product pricing contact sales@ptgrey.com.

For technical support (existing customers only) contact us using our on-line web form at http://www.ptgrey.com/support/contact/

Telephone:

(604) 730-9937

Fax:

(604) 732-8231

Mail:

Point Grey Research, Inc.

305-1847 W. Broadway

Vancouver, BC

V6J 1Y6 CANADA

Or, visit our webpage http://www.ptgrey.com for detailed product information and support.

Index

G

H

I

M

O

P

S

T

U

V

W