The World's Sixth Sense®

Transitioning from FlyCapture2 to Spinnaker SDK

Technical Application Note TAN2015020

Revised 12/15/2017

Applicable Products

- Spinnaker® SDK
- FlyCapture®2 SDK

Application Note Description

This document provides an overview for customers familiar with FlyCapture2 to transition to the Spinnaker SDK.

Benefits of Spinnaker

The Spinnaker SDK is developed based on GenICam. GenICam provides a unified application programming interface to users of machine vision cameras. An introduction to GenICam can be found on EMVA's website.

Some of the key benefits of using the Spinnaker SDK include:

Forward Compatibility

- Features are loaded dynamically from the camera. When we introduce new camera features, you can take advantage of them by simply recompiling your application.
- New machine vision standards are following GenICam (USB3 Vision, GigE Vision). Spinnaker allows you to write more generic code that can be easily migrated to future standards.
- Follows SFNC (standard feature naming convention).

New Graphical User Interface

- There is a new class of GUI controls that allows you to add individual GUI elements to your applications

Backwards Compatibility

- Source compatibility—Newer versions of Spinnaker will not require developers to change their existing code. You can simply recompile your application with a newer version of Spinnaker.
- Functional compatibility—Newer versions of Spinnaker will not remove any functional requirements.
- Binary compatibility—You can run C++ or C based applications by swapping new Spinnaker binaries (DLL) with the old binaries. (This only applies to Spinnaker binaries which have the same major and minor version.)

Differences between Spinnaker and FlyCapture2 Support

The Spinnaker SDK is recommended for users developing new vision applications. Spinnaker provides users with many powerful features to streamline their development process. Users of USB 2.0 and IEEE1394 cameras, or users looking for certain GPIO features, may still require FlyCapture2. The following table summarizes the differences between the features, cameras and platforms supported by Spinnaker and FlyCapture2.

		Spinnaker	FlyCapture2
Feature Support	Feature Search	Yes	No
	GenICam Compliant	Yes	No
	Dynamic Feature Loading	Yes	No
	Standard Feature Naming Convention	Yes	No
	Serial on GPIO	No	Yes
	PWM via GPIO	No	Yes
Camera Support	Blackfly S	Yes	No
	Oryx	Yes	No
	USB 3.1 Cameras GS3-U3, BFLY-U3, CM3-U3, FL3-U3	Yes	Yes
	GigE Cameras GS3-PGE, BFLY-PGE, FL3-GE	Yes	Yes
	USB 2.0 Cameras	No	Yes
	IEEE 1394 Cameras	No	Yes
	Ladybug (use the Ladybug SDK)	No	No
	Bumblebee (use the Triclops SDK)	No	No
Platform Support	Windows	Yes	Yes
	Linux	Yes	Yes
	ARM	Yes	Yes

Terminology Changes

FlyCapture2	Spinnaker	Notes
Brightness	Black Level	Refers to the output of the camera when not illuminated
Exposure	Exposure and Gain	Refers to the combination of camera's shutter and gain. This is also known as the average intensity of the image.
Shutter	Exposure	Refers to the amount of time that the camera's electronic shutter stays open
Packet Size	GevSCPSPacketSize	Refers to packet size, in bytes, to send on the selected channel for a GVSP transmitter or receiver
Packet Delay	GevSCPD	Refers to the delay to insert between each packet for this stream channel
Trigger Mode 0	TriggerSelector→ FrameStartAcquisitionMode→Continuous	Refers to the mode where camera starts integration of light from external trigger source. Sensor exposure time is controlled by shutter (FlyCap2) or exposure (Spinnaker)
Trigger Mode 1	TriggerSelector→ ExposureActiveAcquisitionMode→Continuous	Same as Trigger Mode 0 above except sensor exposure time is controlled by external trigger source
Trigger Mode *X*	Logic Block (Blackfly S and Oryx only)	Refers to all other trigger modes supported by the camera. Logic Block allows you to define any internal logic, including custom trigger modes. For more information see Using Logic Blocks
Memory Channel	User Set	Refers to storing camera settings onto non-volatile memory
High Dynamic Range (HDR)	Sequencer (Blackfly S and Oryx cameras) HDR (USB3 Vision and GigE Vision cameras)	Refers to the cycling of frames with different settings (such as gain and exposure) in order to capture the darkest and brightest portions of the image For more information see Using the Sequencer Feature

FlyCapture2	Spinnaker	Notes
Frame Buffer	Transfer Control	Refers to the transferring of image data to the host
Video Mode	Image Format Control	Refers to controls that define binning/decimation and image size
Mirror / Flip	Reverse X / Reverse Y	Refers to the flipping (either horizontally or vertically) of the image sent from the camera
One Shot	Single Frame	Refers to the ability to fire a single hardware or software trigger and have the camera acquire one image
Multi Shot	Multi Frame	Refers to the ability to fire a single hardware or software triggers and have the camera acquire a specified number of images
Pulse Width Modulation (PWM)	Counters and Timers (Blackfly S and Oryx only)	Refers to a GPIO pin outputting a specified number of pulses with programmable high an low duration. For more information see Using Counters and Timers
Pixel Format: Mono 12	Pixel Format: Mono12 Packed (IIDC-msb) Mono12 Packed	
Pixel Format: Raw8	Pixel Format: Bayer (format) 8	Format is dependent on the bayer pattern of the camera (for example, GB or GR)
Pixel Format: Raw12	Pixel Format: Bayer (format) 12 Packed (IIDC-msb) Bayer (format) 12 Packed	Format is dependent on the bayer pattern of the camera (for example, GB or GR)
Pixel Format: Raw16	Pixel Format: Bayer (format) 16	Format is dependent on the bayer pattern of the camera (for example, GB or GR)
Pixel Format: YUV411	Pixel Format: YCbCr 411 8	
Pixel Format: YUV422	Pixel Format: YCbCr 422 8	

FlyCapture2	Spinnaker	Notes
Pixel Format: YUV444	Pixel Format: YCbCr 8	
Diagnostics: Image consistency errors	Error logging HAL_IMAGE_CONSISTENCY_ERROR	
Diagnostics: Image conversion errors	Error logging The input pixel format is not supported for conversion to the desired output format	
Diagnostics: Dropped images	Buffer Underrun Count	
Diagnostics: Skipped images	Transmit Queue Overflow Count	
Diagnostics: Number of bus arrivals/removals	Error logging Device Arrival/Removal Event Received	

Examples

Included with both the FlyCapture SDK and the Spinnaker SDK are a number of source code examples to help you get started. These examples are provided for C, C++, C#, and VB.NET languages and are precompiled for your convenience.

The table below describes the available Spinnaker SDK examples. Where appropriate, the FlyCapture2 equivalent example is identified.

Spinnaker Example	Description
Acquisition	Enumerate, start acquisition, and grab images *Similar to FlyCapture2: FlyCapture2Test*
AcquisitionMultipleCamera	How to capture images from multiple cameras simultaneously *Similar to FlyCapture2: MultipleCameraEx (FireWire only)*
ChunkData	How to get chunk data on an image, either from the nodemap or from the image itself
DeviceEvents	Create a handler to access device events
Enumeration*	Enumerate interfaces and cameras
EnumerationEvents	Explore arrival and removal events on interfaces and the system
Exposure*	Configure a custom exposure time *Similar to FlyCapture2: ExtendedShutterEx*
ImageEvents	Image events shows how to acquire images using the image event handler. *Similar to FlyCapture2: ImageEventEx (FireWire only)*
ImageFormatControl*	Configure a custom image size and format *Similar to FlyCapture2: CustomImageEx*
Logging	Create a logging event handler
LookupTable	Configure lookup tables for the customization and control of individual pixels
NodeMapCallback	Create, register, use, and unregister callbacks
NodeMapInfo	How to retrieve node map information
SaveToAVI	Save images in AVI format *Similar to FlyCapture2: SaveImageToAVIEx*
Sequencer (Blackfly S and Oryx only)	Capture multiple images with different parameters in a sequence *Similar to FlyCapture2: HighDynamicRange*

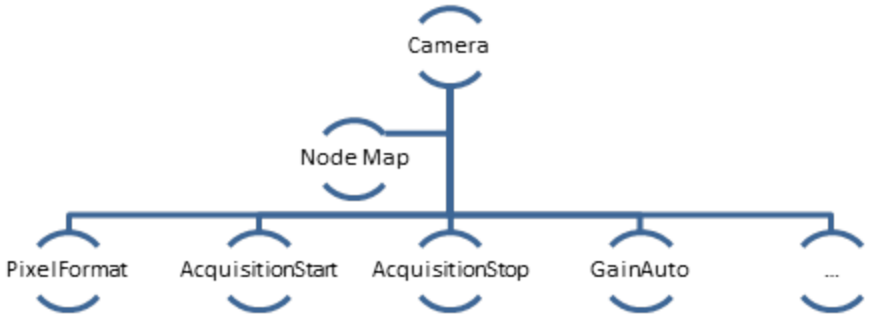
Spinnaker Example	Description
SpinSimpleGUI_MFC	Graphical User Interface for evaluating and setting camera parameters *Similar to FlyCapture2: FlyCapture2GUI*
Trigger*	Trigger shows how to trigger the camera. *Similar to FlyCapture2: AsyncTriggerEx*

*Also available in QuickSpin

Nodes

Nodes are known as properties in FlyCapture2.

Every GenICam compliant camera has an XML description file. The XML describes camera features, their interdependencies, and all other information like availability, access control, and minimum and maximum values. These features include Gain, Exposure Time, Image Format, and others. The elements of a camera description file are represented as software objects called Nodes. A Node map is a list of nodes created dynamically at run time.



To access camera properties such as setting image width:

C++ GenAPI

```
GenApi::INodeMap & nodeMap = cam.GetNodeMap();

CIntegerPtr width = nodeMap.GetNode("Width");

width->SetValue(new_width_val);
```

C# GenAPI

```
NodeMap map = cam.GetNodeMap();
IInteger width = map.GetNode<IInteger>("Width");
width.Value = 320;
```

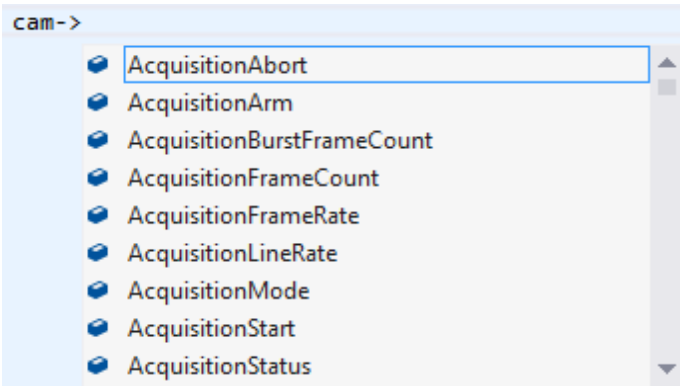
C API

```
spinCameraGetNodeMap(hCam,&hNodeMap); //spinCamera hCam

spinNodeHandle hNode;
int64_t value = 0;
error = spinNodeMapGetNode(hNodeMap,"Width",&hNode);
error = spinIntegerSetValue(hNode, 320);
```

QuickSpin API and Accessing Camera Parameters

Generic programming with GenICam requires developers to know feature names before using them. Spinnaker provides the QuickSpin API, which requires fewer lines of code and allows you to make use of auto completion. The QuickSpin API consists of a list of static functions integrated into the Camera class.

All camera parameters can be accessed through the camera pointer object.



Most camera parameters (all items in camera.h) can be accessed using the QuickSpin API.

For parameters not handled by QuickSpin API, you can access them via GenICam API (GenAPI). GenAPI is the generic programming interface for configuring all kinds of cameras. GenAPI is maintained by the European Machine Vision Association.

Below is an example comparison of inquiring camera gain via GenAPI and QuickSpin API.

C++ GenAPI

```
Spinnaker::GenApi::INodeMap & nodeMap = cam->GetNodeMap();
CFloatPtr GainNode = nodeMap.GetNode("Gain");
Float GainVal = GainNode->GetValue();
```

C++ QuickSpin API

```
float quickGainVal = cam->Gain.GetValue();
```

C# GenAPI

```
INodeMap map = cam.GetNodeMap();
IFloat GainNode = map.GetNode<IFloat>( "Gain");
```

C# QuickSpin API

```
doublequickGainVal = cam.Gain.Value;
```

C Spinnaker API

```
error = spinNodeMapGetNode(hNodeMap,"Gain",&hNode);
error = spinIntegerGetValue(hNode,&value);
```

Event Handling

Events are known as callbacks in FlyCapture2.

Spinnaker introduces two event classes: interface events and device events.

Interface Event

The interface event class is a new feature that is responsible for registering and deregistering user defined interface events such as device arrival and removal.

Interface Event C++

```
class InterfaceEventsHandler : public InterfaceEvent
{

       public :
       InterfaceEventsHandler(){};
       virtual ~InterfaceEventsHandler(){};

       void OnDeviceArrival()
       {

           std::cout<< "A Camera Arrived" << std::endl;

       };

       void OnDeviceRemoval( uint64_tdeviceSerialNumber )
       {

           std::cout<< "A Camera was removed with serial number: " <<
           deviceSerialNumber << std::endl;

       };

};

InterfaceEventsHandler handler;
cam->RegisterEvent(handler);
```

Interface Event C#

```
class InterfaceEventListener : ManagedInterfaceEvent
{

       protectedoverridevoid OnDeviceArrival()
        {

           Console .Out.WriteLine( "A new device has arrived!" );

       }

       protected override void OnDeviceRemoval( UInt64  serialNumber)
       {

           Console.Out.WriteLine( "A device with serial number {0} has been removed!"
           ,
           serialNumber);

       }

}
```

Interface Event C	>Interface event (arrival and removal) handling for C is registered by using the below functions: spinArrivalEventCreate() spinRemovalEventCreate() A detailed example for C interface event is included in Spinnaker source code example: EnumerationEvent_C.cpp

Device Event

The device event class is responsible for registering and deregistering user defined device events such as start or end of exposure.

Device Event C++

```
// Select the Exposure End event
Spinnaker::GenApi:: CEnumerationPtr pEnum = nodeMap .GetNode( "EventSelector" );
pEnum->SetIntValue(pEnum->GetEntryByName( "EventExposureEnd" )->GetValue());

// Turn on the Event notification for Exposure End Event
Spinnaker::GenApi:: CEnumerationPtr pBool = nodeMap .GetNode( "EventNotification" );
pBool->SetIntValue(1);

// Once Exposure End Event is detected, the OnDeviceEvent function will be called
classDeviceEventHandler : publicDeviceEvent
{

       public :
       DeviceEventHandler(){};
       ~DeviceEventHandler(){};

       void OnDeviceEvent( Spinnaker::GenICam::gcstring eventNameeventId )
       {

           std::cout << "Got Device Event with " << eventName << " and ID=" <<
           GetDeviceEventId() << std::endl;

       }

};

// Register event handler
DeviceEventHandler allDeviceEventHandler;
cam->RegisterEvent(allDeviceEventHandler);
```

Device Event C#

```
// Set EventSelector to ExposureEnd
cam.EventSelector.Value = EventSelectorEnums .EventExposureEnd.ToString();

// Set EventNotification to true
cam.EventNotification.Value = EventNotificationEnums .On.ToString();

// After registering the below device event on the camera, OnDeviceEvent will be
automatically called once ExposureEnd event is detected
classManagedDeviceEventHandler : ManagedDeviceEvent
{

       protected override voidOnDeviceEvent( string eventName)
       {

           Console .Out.WriteLine( "Got Device Event with Name=" + eventName + " and ID=
           {0}" , GetDeviceEventId());

       }

}
```

Device Event C

```c
// Create and register ExposureEvent

spinEvent eventExposureEnd = NULL;

error = spinEventCreate(&eventExposureEnd, onSpecificDeviceEvent, NULL);

error = spinCameraRegisterEvent(hCam, eventExposureEnd, "EventExposureEnd");

// Create a function to occur upon specific event occurrences;
 //ensure exact same function signature is used

void onSpecificDeviceEvent(const char* pEventName, void* pUserData)
{
    printf("\t// Specific device event %s...\n", pEventName, (char*)pUserData);
}
```

Chunk Data

Chunk data is known as embedded image info in FlyCapture2.

Chunk data is extra information that the camera can append to each image besides image data. Examples of chunk data include frame counter, image width, image height and exposure time. Spinnaker does not support embedded image info as it was implemented in FlyCapture2.

For a listing of chunk data information supported by your camera, please refer to the camera's Technical Reference manual.

An image is comprised of:

- Leader
- Image Data
- Chunk Information (i.e., gain, exposure, image size)
- Trailer

C++ Enable Chunk Data

```
Cam->ChunkSelector
ChunkSelector.SetValue(ChunkSelectorEnums ::ChunkSelector_ExposureTime) ;

Cam->ChunkEnable.SetValue(true);

Cam->ChunkModeActive.SetValue(true);
```

C++ Retrieve Chunk Data

```
const ChunkData& chunkData = rawImage->GetChunkData();

float64_t currentExposure = chunkData.GetExposureTime();
```

C# Enable Chunk Data

```
cam.ChunkSelector.Value = ChunkSelectorEnums.ExposureTime.ToString();

cam.ChunkEnable.Value = true;

cam.ChunkModeActive.Value = true;
```

C# Retrieve Chunk Data

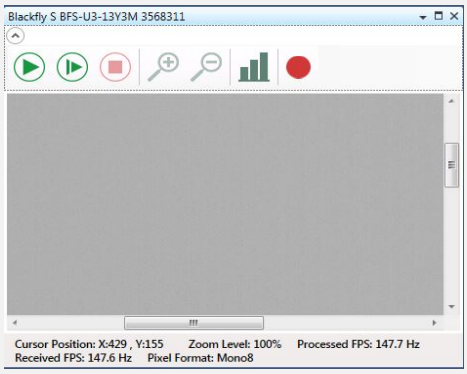
```
String currentExposure = rawImage.ChunkData.ExposureTime.ToString();
```

C# Graphical User Interface API

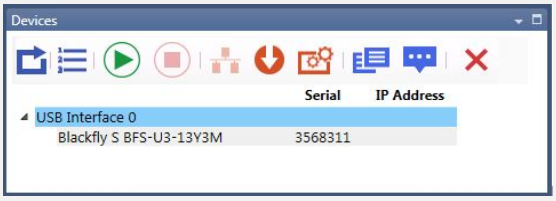
For applications that want to take advantage of Spinnaker's graphical user elements, graphical user interface (GUI) controls are available. GUI controls are divided into static and dynamic categories. Static GUI controls include the CameraSelectionDialog, display window, and property grid window. The GUI dynamically loads the camera's features from the firmware. Therefore, new firmware has the ability to add GUI controls to the same application, without recompiling.

Static GUI Dialogs

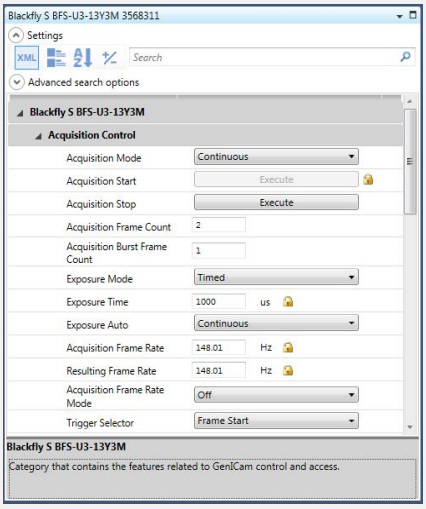
```
//To show image drawing window

GUIFactory AcquisitionGUI = new GUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

ImageDrawingWindow AcquisitionDrawing =
AcquisitionGUI.GetImageDrawingWindow();

AcquisitionDrawing.Connect(cam);

AcquisitionDrawing.Start();

AcquisitionDrawing.ShowModal();
```




```
//To show camera selection window

GUIFactory AcquisitionGUI = newGUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

CameraSelectionWindow camSelection =
AcquisitionGUI.GetCameraSelectionWindow();

camSelection.ShowModal(true);
```

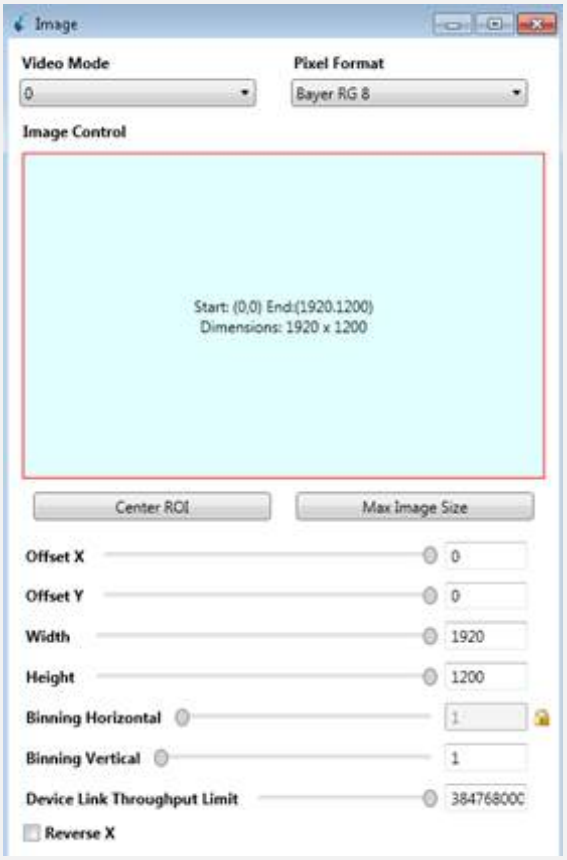



```
//To show property grid window

GUIFactory AcquisitionGUI = new GUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

PropertyGridWindow propWindow =
AcquisitionGUI.GetPropertyGridWindow();

propWindow.Connect(cam);

propWindow.ShowModal();
```



Dynamic GUI Control

```
GUIFactory dynamicGUI = new GUIFactory ();

dynamicGUI.ConnectGUILibrary(cam);

// Get dialog name via dynamicGUI.GetDialogNameList()

Window dlg = dynamicGUI.GetDialogByName(dialogName);

dlg.Owner = Window .GetWindow(this );

dlg.Show();
```



Logging

Spinnaker supports five levels of logging:

- Error—failures that are non-recoverable (this is the default level)
- Warning—failures that are recoverable without user intervention
- Notice—information about events such as camera arrival or disconnect, camera initialize, camera start/stop, or modification of a feature
- Info—information about recurring events that are generated with every image
- Debug—information that can be used to troubleshoot the system

You can define the logging level that you want to monitor. Levels are inclusive, that is, if you monitor debug level error, you also monitor all logging levels above it.

For a complete C++ and C# example of Logging, please see Spinnaker SDK source code examples. By default, Spinnaker SDK's SpinView application saves all logging data to:

C:\ProgramData\Spinnaker\Logs

Register Logging (C++)

```
SystemPtr system = System::GetInstance();

// Register logging callback class
LogCallback callBackClass;
system>RegisterLoggingEvent((Spinnaker::LoggingEvent&)callBackClass);

// Set callback priority level
system->SetLoggingEventPriorityLevel(k_LoggingLevel);

class LogCallback : Spinnaker::LoggingEvent
{

       void OnLogEvent(LoggingEventDataPtr loggingEventDataPtr)
       {
         ...
       }

};
```

Register Logging (C#)

```
// Register logging callback class
LogCallbackHandler callBackClass = newLogCallbackHandler();
system.RegisterLoggingEvent(callBackClass);

// Set callback priority level
system.SetLoggingEventPriorityLevel(LoggingLevel);

classLogCallbackHandler : ManagedLoggingEventHandler
{

    publicoverridevoid OnLogEvent(ManagedLoggingEvent loggingEvent)
    {
    ...
    }

}
```

Register Logging (C)

```c
void onLogEvent(const spinLogEvent self, void* pUserData)
{
        ...
}

// Create log event
spinLogEvent logEvent = NULL;

error = spinLogEventCreate(&logEvent, onLogEvent, NULL);

error = spinSystemRegisterLogEvent(hSystem, logEvent);
```

Programmer's Section

FlyCapture2 Feature Comparison with Spinnaker

For programmers not familiar with GenICam API, you can take a look at EMVA's GenICam Standard.

Spinnaker re-engineered the way we perceived camera features in FlyCapture2. Camera features and properties are named according to standard feature naming convention (SFNC). The table below compares the Spinnaker SDK features with those in the FlyCapture2 SDK.

For example, IIDC register read and write is no longer available. Instead, camera properties are accessed through the GenICam node map.

FlyCapture2 Features	Spinnaker Features	Notes
IIDC register read and write Source Code Example: AsyncTriggerEx	GenICam node map get/set values Source Code Example: Acquisition	This feature is typically used to access camera settings or control the camera's state.
Embedded image info Source Code Example: MultipleCameraWriteToDiskEx	Chunk data Source Code Example: ChunkData	This feature allows the camera to add image metadata to the transmission.
ImageEvents Source Code Example: ImageEventEx	GenICam message or event channel Source Code Example: DeviceEvents	This feature is used to signal the user when certain events such as image arrival has happened.
FC2Config struct such as grab mode and num_buffer Source Code Example: RecordingDialog	Stream Node map Source Code Example: NodeMapInfo	This feature can be used to set buffer mode and the number of buffers.
Error return code Source Code Example: FlyCapture2Test	Exceptions Source Code Example: Acquisition	New error handling approach uses exceptions instead of error codes.
Format7 packet size to control bandwidth Source Code Example: GigEGrabEx	DeviceLinkLayerThroughputLimit	This feature defines the total available bandwidth that can be allocated for the camera.
Imaging mode (i.e., Format7 mode 1)	Binning controls in GenICam	This feature refers to the camera's

FlyCapture2 Features	Spinnaker Features	Notes
Source Code Example: CustomImageEx		binning mode where overall resolution is reduced to achieve faster frame rate or brighter image.
Events callback Source Code Example: BusEventEx_CSharp	Register event class with overloaded functions Source Code Example: EnumerationEvents	This feature refers to bus event callbacks such as camera arrival and camera removal callback.
GUID to identify and track cameras Source Code Example: FlyCapture2Test	Unique camera class Source Code Example: Acquisition	You must use this identifier to access camera features in the SDK.

C++ Features

These tables provide a comparison of popular features used in FlyCapture2 C++ API and Spinnaker C++ API.

Enumeration

The snippet below detects the number of cameras connected and enumerates them from an index.

FlyCapture2 C++ API

```
 BusManager busMgr;
unsigned int numCameras;
Camera camera;
busMgr.GetNumOfCameras(&numCameras);
PGRGuid guid;
for ( unsigned int i = 0; i < numCameras; i++)
{
    busMgr.GetCameraFromIndex( i, &guid );
    camera.Connect(i);
}
```

Spinnaker C++ GenAPI

```
SystemPtr system = System::GetInstance();
CameraList camList = system->GetCameras();
unsigned int numCameras = camList.GetSize();
CameraPtr pCam = NULL;
for (int i = 0; i < numCameras; i++)
{
    pCam = camList.GetByIndex(i);
      pCam->Init();
}
```

Asynchronous Hardware Triggering

The snippet below does the following:

- Enables Trigger Mode
- Configures GPIO0/Line0 as the trigger input source
- Specifies the trigger signal polarity as an active high (rising edge) signal

FlyCapture2 C++ API

```
TriggerMode mTrigger;

mTrigger.mode = 0;
mTrigger.source = 0;
mTrigger.parameter = 0;
mTrigger.onOff = true;
mTrigger.polarity = 1;

cam.SetTriggerMode(&mTrigger);
```

Spinnaker C++ QuickSpin API

```
Cam->TriggerMode.SetValue(Spinnaker::TriggerModeEnums::TriggerMode_On);

Cam->TriggerSource.SetValue(Spinnaker::TriggerSourceEnums::TriggerSource_
Line0);

Cam->TriggerSelector.SetValue
(Spinnaker::TriggerSelectorEnums::TriggerSelector_FrameStart);

Cam->TriggerActivation.SetValue
(Spinnaker::TriggerActivationEnums::TriggerActivation_RisingEdge);
```

Spinnaker C++ GenAPI

```
CEnumerationPtr triggerMode = nodeMap.GetNode("TriggerMode");
triggerMode->SetIntValue(triggerMode->GetEntryByName("On")->GetValue());

CEnumerationPtr triggerSource = nodeMap.GetNode("TriggerSource");
triggerSource->SetIntValue(triggerSource->GetEntryByName("Line0")->GetValue
());

CEnumerationPtr triggerSelector = nodeMap.GetNode("TriggerSelector");
triggerSelector->SetIntValue(triggerSelector->GetEntryByName("FrameStart")-
>GetValue());

CEnumerationPtr triggerActivation = nodeMap.GetNode("TriggerActivation");
triggerActivation->SetIntValue(triggerActivation->GetEntryByName
("RisingEdge")->GetValue());
```

Setting Black Level

Black level is known as brightness in FlyCapture2.

BlackLevel is the GenICam feature that represents the DC offset that is applied to the video signal. This example compares the mechanism used to set this feature in both environments.

FlyCapture2 C++ API

```
//Declare a Property struct.
Property prop;

//Define the property to adjust.
prop.type = BRIGHTNESS;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of brightness to 1.5%.
prop.absValue = 1.5;

//Set the property.
error = cam.SetProperty ( &prop );
```

Spinnaker C++ QuickSpin API

```
// Brightness is called black level in GenICam
pCam->BlackLevelSelector.SetValue
(Spinnaker::BlackLevelSelectorEnums::BlackLevelSelector_All);

//Set the absolute value of brightness to 1.5%.
pCam->BlackLevel.SetValue(1.5);
```

Spinnaker C++ GenAPI

```
CEnumerationPtr blackLevelSelector = nodeMap.GetNode("BlackLevelSelector");
blackLevelSelector->SetValue(SetIntValue(blackLevelSelector->GetEntryByName
("True")->GetValue());

CFloatPtr blackLevel = nodeMap.GetNode("BlackLevel");
blackLevel->SetValue(1.5);
```

Setting Exposure Time

Exposure time is known as shutter in FlyCapture2.

ExposureTime refers to the amount of time that the camera's electronic shutter stays open. This example sets your camera's exposure/shutter time to 20 milliseconds.

FlyCapture2 C++ API

```
//Declare a Property struct.
Property prop;

//Define the property to adjust.
prop.type = SHUTTER;

//Ensure the property is on.
prop.onOff = true;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of shutter to 20 ms.
prop.absValue = 20;

//Set the property.
error = cam.SetProperty( &prop );
```

Spinnaker C++ QuickSpin API

```
// Turn off auto exposure
cam->ExposureAuto.SetValue(Spinnaker::ExposureAutoEnums::ExposureAuto_
Off);

//Set exposure mode to "Timed"
cam->ExposureMode.SetValue(Spinnaker::ExposureModeEnums::ExposureMode_
Timed);

//Set absolute value of shutter exposure time to 20000 microseconds
cam->ExposureTime.SetValue(20000);
```

Spinnaker C++ GenAPI

```
CEnumerationPtr exposureAuto = nodeMap.GetNode("ExposureAuto");
exposureAuto->SetIntValue(exposureAuto->GetEntryByName("Off")->GetValue
());

CEnumerationPtr exposureMode = nodeMap.GetNode("ExposureMode");
exposureMode->SetIntValue(exposureMode->GetEntryByName("Timed")->GetValue
());

CFloatPtr exposureTime = nodeMap.GetNode("ExposureTime");
 exposureTime->SetValue(20000);
```

Setting Gain

The following code snippet adjusts gain to 10.5 dB.

FlyCapture2 C++ API

```
//Declare a Property struct.
Property prop;

//Define the property to adjust.
prop.type = GAIN;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of gain to 10.5 dB.
prop.absValue = 10.5;

//Set the property.
error = cam.SetProperty( &prop );
```

Spinnaker C++ QuickSpin API

```
//Turn auto gain off
cam->GainAuto.SetValue(Spinnaker::GainAutoEnums::GainAuto_Off);

//Set gain to 10.5 dB
cam->Gain.SetValue(10.5);
```

Spinnaker C++ GenAPI

```
CEnumerationPtr gainAuto = nodeMap.GetNode("GainAuto");
gainAuto->SetIntValue(gainAuto->GetEntryByName("Off")->GetValue());

CFloatPtr gainValue = nodeMap.GetNode("Gain");
gainValue->SetValue(10.5);
```

Setting Gamma

The following code snippet adjusts gamma to 1.5.

FlyCapture2 C++ API

```
//Declare a Property struct.
Property prop;

//Define the property to adjust.
prop.type = GAMMA;

//Ensure the property is on.
prop.onOff = true;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of gamma to 1.5.
prop.absValue = 1.5;

//Set the property.
error = cam.SetProperty( &prop );
```

Spinnaker C++ QuickSpin API

```
// Set the absolute value of gamma to 1.5
cam.Gamma.SetValue(1.5);
```

Spinnaker C++ GenAPI

```
CFloatPtr gamma = nodeMap.GetNode("Gamma");
gamma->SetValue(1.5);
```

Setting White Balance

The following code snippet adjusts the white balance's red and blue channels.

FlyCapture2 C++ API

```
//Declare a Property struct.
Property prop;

//Define the property to adjust.
prop.type = WHITE_BALANCE;

//Ensure the property is on.
prop.onOff = true;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Set the white balance red channel to 500.
prop.valueA = 500;

//Set the white balance blue channel to 850.
prop.valueB = 850;

//Set the property.
error = cam.SetProperty( &prop );
```

Spinnaker C++ QuickSpin API

```
//Set auto white balance to off
cam->BalanceWhiteAuto.SetValue
(Spinnaker::BalanceWhiteAutoEnums::BalanceWhiteAuto_Off);

//Select blue channel balance ratio
cam->BalanceRatioSelector.SetValue
(Spinnaker::BalanceRatioSelectorEnums::BalanceRatioSelector_Blue);

//Set the white balance blue channel to 2
CFloatPtr BalanceRatio = nodeMap.GetNode("BalanceRatio");
BalanceRatio->SetValue(2);

//Set the white balance red channel to 2
cam->BalanceRatioSelector.SetValue
(Spinnaker::BalanceRatioSelectorEnums::BalanceRatioSelector_Red);
BalanceRatio->SetValue(2);
```

Spinnaker C++ GenAPI

```
CEnumerationPtr balanceWhiteAuto = nodeMap.GetNode("BalanceWhiteAuto");
balanceWhiteAuto->SetIntValue(balanceWhiteAuto->GetEntryByName("Off")-
>GetValue());

CEnumerationPtr balanceRatioSelector = nodeMap.GetNode
("BalanceRatioSelector");
balanceRatioSelector->SetIntValue(balanceRatioSelector->GetEntryByName
("Blue")->GetValue());

CFloatPtr balanceRatio = nodeMap.GetNode("BalanceRatio");
balanceRatio->SetValue(2);

balanceRatioSelector->SetIntValue(balanceRatioSelector->GetEntryByName("Red")-
>GetValue());
balanceRatio->SetValue(2);
```

Accessing Raw Bayer Data

Raw image data can be accessed programmatically via the getData method of the FlyCapture2 and Spinnaker Image class. In 8 bits per pixel modes such as BayerRG8, the first byte represents the pixel at [row 0, column 0], the second byte at [row 0, column 1], and so on. The top left corner of the image data represents row 0, column 0.

FlyCapture2 C++ API

```
// Read the BAYER_TILE_MAPPING register 0x1040 to determine the current Bayer output
format (RGGB, GRBG, and so on). Using a Bayer format of RGGB, for example, the getData
method returns the following (assuming char* data = rawImage.GetData(); and an Image
object rawImage):

// Assuming image is 640 x 480
// data[0] = Row 0, Column 0 = red pixel (R)
// data[1] = Row 0, Column 1 = green pixel (G)
// data[640 ] = Row 1, Column 0 = green pixel (G)
// data[641] = Row 1, Column 1 = blue pixel (B)
```

Spinnaker C++ API

```
// Assuming image is 640 x 480 resolution. The current pixel format as well as
PixelColorFilter indicate the Bayer Tile Mapping for the camera. For example, BayerRG8 is
RGGB.

ImagePtr pResultImage = cam.GetNextImage();
char* data = (char*)pResultImage->GetData();

// Assuming image is 640 x 480
// data[0] = Row 0, Column 0 = red pixel (R)
// data[1] = Row 0, Column 1 = green pixel (G)
// data[640] = Row 1, Column 0 = green pixel (G)
// data[641] = Row 1, Column 1 = blue pixel (B)
```

Setting Number of Image Buffers

The following code snippet adjusts the number of image buffers that the driver initializes for buffering images on your PC to 11 (default is 10).

FlyCapture2 C++ API

```
FC2Config BufferFrame;
Camera** ppCameras = newCamera*[numCameras];
ppCameras[0]->GetConfiguration(&BufferFrame);
BufferFrame.numBuffers = 11;
ppCameras[0]->SetConfiguration(&BufferFrame);
```

Spinnaker C++ API

```
Spinnaker::GenApi::INodeMap & sNodeMap = cam->GetTLStreamNodeMap();
CIntegerPtr StreamNode = sNodeMap.GetNode("StreamDefaultBufferCount");
INT64 bufferCount = StreamNode->GetValue();
StreamNode->SetValue(11);
```

C# Features

These tables provide a comparison of popular features used in FlyCapture2 C# API and Spinnaker C# API.

Enumeration

The snippet below detects the number of cameras connected and enumerates them from an index.

FlyCapture2 C# API

```
ManagedBusManager busMgr = new ManagedBusManager();
uint numCameras = busMgr.GetNumOfCameras();
for (uint i = 0; i < numOfCameras; i++)
{
    ManagedPGRGuid guid = busMgr.GetCameraFromIndex(i);
      cameras[i].Connect(guid);
}
```

Spinnaker C# GenAPI

```
IList<IManagedCamera> camList = system.GetCameras();
foreach (IManagedCamera managedCamera in camList)
   using (managedCamera)
    {
        managedCamera.Init();
    }
```

Asynchronous Hardware Triggering

The snippet below does the following:

- Enables Trigger Mode
- Configures GPIO0/Line0 as the trigger input source
- Specifies the trigger signal polarity as an active high (rising edge) signal

FlyCapture2 C# API

```
// Get current trigger settings
TriggerMode triggerMode = cam.GetTriggerMode();

// Set camera to trigger mode 0
// A source of 7 means software trigger
triggerMode.onOff = true;
triggerMode.mode = 0;
triggerMode.parameter = 0;

// Set the trigger mode
cam.SetTriggerMode(triggerMode);
```

Spinnaker C# QuickSpin API

```
cam.TriggerMode.Value = TriggerModeEnums.On.ToString();
cam.TriggerSource.Value = TriggerSourceEnums.Line0.ToString();
cam.TriggerSelector.Value = TriggerSelectorEnums.FrameStart.ToString();
cam.TriggerActivation.Value = TriggerActivationEnums.RisingEdge.ToString
();
```

Spinnaker C# GenAPI

```
IEnum triggerMode = nodeMap.GetNode<IEnum>("TriggerMode");
triggerMode.Value = "On";

IEnum triggerSource = nodeMap.GetNode<IEnum>("TriggerSource");
triggerSource.Value = "Line0";

IEnum triggerSelector = nodeMap.GetNode<IEnum>("TriggerSelector");
triggerSelector.Value = "FrameStart";

IEnum triggerActivation = nodeMap.GetNode<IEnum>("TriggerActivation");
triggerActivation.Value = "RisingEdge";
```

Setting Black Level

Black level is known as brightness in FlyCapture2.

BlackLevel is the GenICam feature that represents the DC offset that is applied to the video signal. This example compares the mechanism used to set this feature in both environments.

FlyCapture2 C# API

```
//Declare a Property struct.
CameraProperty prop = newCameraProperty();
prop.type = PropertyType.Brightness;
prop.absControl = true;
prop.absValue = 2;

// Assuming cam is a managedCamera that has been initialized
cam.SetProperty(prop);
```

Spinnaker C# QuickSpin API

```
// Black Level is also referred to as brightness
cam.BlackLevelSelector.Value = BlackLevelSelectorEnums.All.ToString();

// Set Black Level to an absolute value of 1.5%
cam.BlackLevel.Value = 1.5;
```

Spinnaker C# GenAPI

```
IEnum blackLevelSelector = nodeMap.GetNode<IEnum>("BlackLevelSelector");
blackLevelSelector.Value = "All";

IFloat blackLevel = nodeMap.GetNode<IFloat>("BlackLevel");
blackLevel.Value = 1.5;
```

Setting Exposure Time

Exposure time is known as shutter in FlyCapture2.

ExposureTime refers to the amount of time that the camera's electronic shutter stays open. This example sets your camera's exposure/shutter time to 20 milliseconds.

FlyCapture2 C# API

```
//Declare a Property struct.
CameraProperty prop = new CameraProperty();
prop.type = PropertyType.Shutter;
prop.autoManualMode = false;
prop.absControl = true;
prop.absValue = 20;
prop.onOff = true;

// Assuming cam is a managedCamera that has been initialized
cam.SetProperty(prop);
```

Spinnaker C# QuickSpin API

```
// Turn off auto exposure
cam.ExposureAuto.Value = ExposureAutoEnums.Off.ToString();

// Set exposure mode to "Timed"
cam.ExposureMode.Value = ExposureModeEnums.Timed.ToString();

// Set exposure to 20000 microseconds
cam.ExposureTime.Value = 20000;
```

Spinnaker C# GenAPI

```
IEnum exposureAuto = nodeMap.GetNode<IEnum>("ExposureAuto");
exposureAuto.Value = "Off";

IEnum exposureMode = nodeMap.GetNode<IEnum>("ExposureMode");
exposureMode.Value = "Timed";

IFloat exposureTime = nodeMap.GetNode<IFloat>("ExposureTime");
exposureTime.Value = 20000;
```

Setting Gain

The following code snippet adjusts gain to 10.5 dB.

FlyCapture2 C# API

```
//Declare a Property struct.
CameraProperty prop = new CameraProperty();
prop.type = PropertyType.Gain;
prop.autoManualMode = false;
prop.absControl = true;
prop.absValue = 10;
prop.onOff = true;

// Assuming cam is a managedCamera that has been initialized
cam.SetProperty(prop);
```

Spinnaker C# QuickSpin API

```
//Turn auto gain off
cam.GainAuto.Value = GainAutoEnums.Off.ToString();

//Set gain to 10.5 dB
cam.Gain.Value = 10.5;
```

Spinnaker C# GenAPI

```
IEnum gainAuto = nodeMap.GetNode<IEnum>("GainAuto");
gainAuto.Value = "Off";

IFloat gainValue = nodeMap.GetNode<IFloat>("Gain");
gainValue.Value = 10.5;
```

Setting Gamma

The following code snippet adjusts gamma to 1.5.

FlyCapture2 C# API

```
//Declare a Property struct.
CameraProperty prop = new CameraProperty();
prop.type = PropertyType.Gamma;
prop.autoManualMode = false;
prop.absControl = true;
prop.absValue = 2;
prop.onOff = true;

// Assuming cam is a managedCamera that has been initialized
cam.SetProperty(prop);
```

Spinnaker C# QuickSpin API

```
// Set the absolute value of gamma to 1.5
cam.Gamma.Value = 1.5;
```

Spinnaker C# GenAPI

```
IFloat gamma = nodeMap.GetNode<IFloat>("Gamma");
gamma.Value = 1.5;
```

Setting White Balance

The following code snippet adjusts the white balance's red and blue channels.

FlyCapture2 C# API

```
//Declare a Property struct.
CameraProperty prop = new CameraProperty();
prop.type = PropertyType.WhiteBalance;
prop.autoManualMode = false;
prop.absControl = false;
prop.valueA = 500;
prop.onOff = true;

// Assuming cam is a managedCamera that has been initialized
cam.SetProperty(prop);

// Get current trigger settings
TriggerMode triggerMode = cam.GetTriggerMode();
```

Spinnaker C# QuickSpin API

```
// Set auto white balance to off
cam.BalanceWhiteAuto.Value = BalanceWhiteAutoEnums.Off.ToString();

// Select blue channel balance ratio
cam.BalanceRatioSelector.Value = BalanceRatioSelectorEnums.Blue.ToString
();

// Set the white balance blue channel to 2
cam.BalanceRatio.Value = 2;
```

Spinnaker C# GenAPI

```
IEnum balanceWhiteAuto = nodeMap.GetNode<IEnum>("BalanceWhiteAuto");
balanceWhiteAuto.Value = "Off";

IEnum balanceRatioSelector = nodeMap.GetNode<IEnum>
("BalanceRatioSelector");
balanceRatioSelector.Value = "Blue";

IFloat balanceRatio = nodeMap.GetNode<IFloat>("BalanceRatio");
balanceRatio.Value = 2;
```

Accessing Raw Bayer Data

Raw image data can be accessed programmatically via the getData method of the FlyCapture2 and Spinnaker Image class. In 8 bits per pixel modes such as BayerRG8, the first byte represents the pixel at [row 0, column 0], the second byte at [row 0, column 1], and so on. The top left corner of the image data represents row 0, column 0.

FlyCapture2 C# API

```
// Read the BAYER_TILE_MAPPING register 0x1040 to determine the current Bayer output
format (RGGB, GRBG, and so on). Using a Bayer format of RGGB, for example, the getData
method returns the following (assuming byte* data = rawImage.data; and an Image object
data):

// Assuming image is 640 x 480
data = Row 0, Column 0 = red pixel (R)
data + 1 = Row 0, Column 1 = green pixel (G)
data + 640 = Row 1, Column 0 = green pixel (G)
data + 641 = Row 1, Column 1 = blue pixel (B)
```

Spinnaker C# API

```
Unsafe
{
    byte* data = rawImage.data;

   // Assuming image is 640 x 480 resolution with bayer tile RGGB
    // data represents row 0 column 0, red pixel (R)
    // data + 1 represents row 0 column 1, green pixel (G)
    // data + 640 represents row 1 column 0, green pixel (G)
    // data + 641 represents row 1 column 1, blue pixel (G)
}
```

Setting Number of Image Buffers

The following code snippet adjusts the number of image buffers that the driver initializes for buffering images on your PC to 11 (default is 10).

FlyCapture2 C# API

```
FC2Config bufferSetting;
bufferSetting = cam.GetConfiguration();
bufferSetting.numBuffers = 11;
cam.SetConfiguration(bufferSetting);
```

Spinnaker C# API

```
INodeMap sNodeMap = cam.GetStreamNodeMap();
IInteger streamNode = sNodeMap.GetNode<IInteger>("StreamDefaultBufferCount");
long bufferCount = streamNode.Value;
streamNode.Value = 11;
```

C Features

These tables provide a comparison of popular features used in FlyCapture2 C API and Spinnaker C API.

Enumeration

The snippet below detects the number of cameras connected and enumerates them from an index.

FlyCapture2 C API

```
fc2PGRGuid guid;
fc2GetNumOfCameras( context, &numCameras );
for (i = 0; i < numCameras; i++)
{
    fc2GetCameraFromIndex( context, i, &guid );
    fc2Connect( context, &guid );
}
```

Spinnaker C API

```
spinCamera hCamera = NULL;
spinCameraListGetSize(hCameraList, &numCameras);
for (i = 0; i < numCameras; i++)
{
    spinCameraListGet(hCameraList, i, &hCamera);
    spinCameraInit(hCam);
}
```

Asynchronous Hardware Triggering

The snippet below does the following:

- Enables Trigger Mode
- Configures GPIO0/Line0 as the trigger input source
- Specifies the trigger signal polarity as an active high (rising edge) signal

FlyCapture2 C API

```
fc2TriggerMode mTrigger;

mTrigger.mode = 0;
mTrigger.source = 0;
mTrigger.parameter = 0;
mTrigger.onOff = true;
mTrigger.polarity = 1;

fc2SetTriggerMode( context, & mTrigger);
```

Spinnaker C API

```
spinNodeHandle hTriggerMode = NULL;
spinNodeHandle hTriggerModeOn = NULL;
int64_t triggerModeOn = 0;

err = spinNodeMapGetNode(hNodeMap, "TriggerMode", &hTriggerMode);
err = spinEnumerationGetEntryByName(hTriggerMode, "On", &hTriggerModeOn);
err = spinEnumerationEntryGetValue(hTriggerModeOn, &triggerModeOn);
err = spinEnumerationSetIntValue(hTriggerMode, triggerModeOn);

spinNodeHandle hTriggerSource = NULL;
spinNodeHandle hTriggerSourceChoice = NULL;
int64_t triggerSourceChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "TriggerSource", &hTriggerSource);
err = spinEnumerationGetEntryByName(hTriggerSource, "Line0", &hTriggerSourceChoice);
err = spinEnumerationEntryGetValue(hTriggerSourceChoice, &triggerSourceChoice);
err = spinEnumerationSetIntValue(hTriggerSource, triggerSourceChoice);

spinNodeHandle hTriggerSelector = NULL;
spinNodeHandle hTriggerSelectorChoice = NULL;
int64_t triggerSelectorChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "TriggerSelector", &hTriggerSelector);
err = spinEnumerationGetEntryByName(hTriggerSource, "FrameStart",
&hTriggerSelectorChoice);
err = spinEnumerationEntryGetValue(hTriggerSelectorChoice, &triggerSourceChoice);
err = spinEnumerationSetIntValue(hTriggerSelector, triggerSelectorChoice);

spinNodeHandle hTriggerActivation = NULL;
spinNodeHandle hTriggerActivationChoice = NULL;
int64_t triggerActivationChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "TriggerActivation", &hTriggerActivation);
err = spinEnumerationGetEntryByName(hTriggerActivation, "RisingEdge",
&hTriggerActivationChoice);
err = spinEnumerationEntryGetValue(hTriggerActivationChoice, &triggerSourceChoice);
err = spinEnumerationSetIntValue(hTriggerActivation, triggerActivationChoice);
```

Setting Black Level

Black level is known as brightness in FlyCapture2.

BlackLevel is the GenICam feature that represents the DC offset that is applied to the video signal. This example compares the mechanism used to set this feature in both environments.

FlyCapture2 C API

```
//Declare a Property struct.
fc2Property, prop;

//Define the property to adjust.
 prop.type = BRIGHTNESS;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of brightness to 1.5%.
prop.absValue = 1.5;

//Set the property.
error = fc2SetProperty ( context, &prop );
```

Spinnaker C API

```
// Black Level is also referred to as brightness
spinNodeHandle hBlackLevelSelector = NULL;
spinNodeHandle hBlackLevelSelectorChoice = NULL;
int64_t blackLevelSelectorChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "BlackLevelSelector", &hBlackLevelSelector);
err = spinEnumerationGetEntryByName(hBlackLevelSelector, "All",
&hBlackLevelSelectorChoice);
err = spinEnumerationEntryGetIntValue(hBlackLevelSelectorChoice,
&blackLevelSelectorChoice);
err = spinEnumerationSetIntValue(hBlackLevelSelectorChoice,
blackLevelSelectorChoice);

//Set the value of black level to 1.5%.
spinNodeHandle hBlackLevel;
err = spinNodeMapGetNode(hNodeMap, "BlackLevel", &hBlackLevel);
err = spinFloatSetValue(hBlackLevel, 1.5);
```

Setting Exposure Time

Exposure time is known as shutter in FlyCapture2.

ExposureTime refers to the amount of time that the camera's electronic shutter stays open. This example sets your camera's exposure/shutter time to 20 milliseconds.

FlyCapture2 C API

```
//Declare a Property struct.
fc2Property prop;

//Define the property to adjust.
prop.type = SHUTTER;

//Ensure the property is on.
prop.onOff = true;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of shutter to 20 ms.
prop.absValue = 20;

//Set the property.
error = fc2.SetProperty( &prop );
```

Spinnaker C API

```
// Turn off auto exposure
spinNodeHandle hExposureAutoSelector = NULL;
spinNodeHandle hExposureAutoSelectorChoice = NULL;
int64_t exposureAutoSelectorChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "ExposureAuto", &hExposureAutoSelector);
err = spinEnumerationGetEntryByName(hExposureAutoSelector, "Off",
&hExposureAutoSelectorChoice);
err = spinEnumerationEntryGetValue(hExposureAutoSelectorChoice,
&exposureAutoSelectorChoice);
err = spinEnumerationSetIntValue(hExposureAutoSelectorChoice,
exposureAutoSelectorChoice);

//Set exposure mode to "Timed"
spinNodeHandle hExposureModeSelector = NULL;
spinNodeHandle hExposureModeSelectorChoice = NULL;
int64_t exposureModeSelectorChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "ExposureAuto", &hExposureModeSelector);
err = spinEnumerationGetEntryByName(hExposureModeSelector, "Timed",
&hExposureModeSelectorChoice);
err = spinEnumerationEntryGetValue(hExposureModeSelectorChoice,
&exposureModeSelectorChoice);
err = spinEnumerationSetIntValue(hExposureModeSelectorChoice,
exposureModeSelectorChoice);

//Set value of exposure time to 20000 microseconds
spinNodeHandle hExposureTime;
err = spinNodeMapGetNode(hNodeMap, "ExposureTime", &hExposureTime);
err = spinIntegerSetValue(hExposureTime, 20000);
```

Setting Gain

The following code snippet adjusts gain to 10.5 dB.

FlyCapture2 C API

```
//Declare a Property struct.
fc2Property prop;

//Define the property to adjust.
prop.type = GAIN;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of shutter to 20 ms.
prop.absValue = 10.5;

//Set the property.
error = fc2.SetProperty( &prop );
```

Spinnaker C API

```
//Turn auto gain off
spinNodeHandle hGainAutoSelector = NULL;
spinNodeHandle hGainAutoSelectorChoice = NULL;
int64_t gainAutoSelectorChoice = 0;

err = spinNodeMapGetNode(hNodeMap, "GainAuto", &hGainAutoSelector);
err = spinEnumerationGetEntryByName(hGainAutoSelector, "Off", &hGainAutoSelectorChoice);
err = spinEnumerationEntryGetValue(hGainAutoSelectorChoice, &gainAutoSelectorChoice);
err = spinEnumerationSetIntValue(hGainAutoSelectorChoice, gainAutoSelectorChoice);

//Set gain to 10.5 dB
spinNodeHandle hGain;
err = spinNodeMapGetNode(hNodeMap, "Gain", &hGain);
err = spinIntegerSetValue(hGain, 10.5);
```

Setting Gamma

The following code snippet adjusts gamma to 1.5.

FlyCapture2 C API

```
//Declare a Property struct.
fc2Property prop;

//Define the property to adjust.
prop.type = GAMMA;

//Ensure the property is on.
prop.onOff = true;

//Ensure the property is set up to use absolute value control.
prop.absControl = true;

//Set the absolute value of gamma to 1.5.
prop.absValue = 1.5;

//Set the property.
error = fc2.SetProperty( &prop );
```

Spinnaker C API

```
// Enable Gamma
spinNodeHandle hGammaEnable = NULL;

err = spinNodeMapGetNode(hNodeMap, "GammaEnable", &hGammaEnable);
err = spinBooleanSetValue(hGammaEnable, false);

// Set the absolute value of gamma to 1.5
spinNodeHandle hGamma;
err = spinNodeMapGetNode(hNodeMap, "Gamma", &hGamma);
err = spinIntegerSetValue(hGamma, 1.5);
```

Setting White Balance

The following code snippet adjusts the white balance's red and blue channels.

FlyCapture2 C API

```
//Declare a Property struct.
fc2Property prop;

//Define the property to adjust.
prop.type = WHITE_BALANCE;

//Ensure the property is on.
prop.onOff = true;

//Ensure auto-adjust mode is off.
prop.autoManualMode = false;

//Set the white balance red channel to 500.
prop.valueA = 500;

//Set the white balance blue channel to 850.
prop.valueB = 850;

//Set the property.
error = fc2.SetProperty( &prop );
```

Spinnaker C API

```
//Set auto white balance to off
spinNodeHandle hBalanceWhiteAutoSelector = NULL;
spinNodeHandle hBalanceWhiteAutoSelectorChoice = NULL;
int64_t balanceWhiteAutoSelector = 0;

err = spinNodeMapGetNode(hNodeMap, "BalanceWhiteAuto", &hBalanceWhiteAutoSelector);
err = spinEnumerationGetEntryByName(hBalanceWhiteAutoSelector, "Off",
&hBalanceWhiteAutoSelectorChoice);
err = spinEnumerationEntryGetValue(hBalanceWhiteAutoSelectorChoice,
&balanceWhiteAutoSelector);
err = spinEnumerationSetIntValue(hBalanceWhiteAutoSelectorChoice,
balanceWhiteAutoSelector);

//Select blue channel balance ratio
spinNodeHandle hBalanceRatioSelector = NULL;
spinNodeHandle hBalanceRatioSelectorChoice = NULL;
int64_t balanceRatioSelector = 0;

err = spinNodeMapGetNode(hNodeMap, "BalanceRatioSelector", &hBalanceRatioSelector);
err = spinEnumerationGetEntryByName(hBalanceRatioSelector, "Blue",
&hBalanceRatioSelectorChoice);
err = spinEnumerationEntryGetValue(hBalanceRatioSelectorChoice, &balanceRatioSelector);
err = spinEnumerationSetIntValue(hBalanceRatioSelectorChoice, balanceRatioSelector);

//Set the white balance blue channel to 2
spinNodeHandle hBlueRatio;
err = spinNodeMapGetNode(hNodeMap, "BalanceRatio", &hGain);
err = spinIntegerSetValue(hBlueRatio, 2);

//Set the white balance red channel to 2
err = spinNodeMapGetNode(hNodeMap, "BalanceRatioSelector", &hBalanceRatioSelector);
err = spinEnumerationGetEntryByName(hBalanceRatioSelector, "Red",
&hBalanceRatioSelectorChoice);
err = spinEnumerationEntryGetValue(hBalanceRatioSelectorChoice, &balanceRatioSelector);
err = spinEnumerationSetIntValue(hBalanceRatioSelectorChoice, balanceRatioSelector);

spinNodeHandle hRedRatio;
err = spinNodeMapGetNode(hNodeMap, "BalanceRatio", &hRedRatio);
err = spinIntegerSetValue(hRedRatio, 2);
```

Accessing Raw Bayer Data

Raw image data can be accessed programmatically via the getData method of the FlyCapture2 and Spinnaker Image class. In 8 bits per pixel modes such as BayerRG8, the first byte represents the pixel at [row 0, column 0], the second byte at [row 0, column 1], and so on. The top left corner of the image data represents row 0, column 0.

FlyCapture2 C API

```
// Read the BAYER_TILE_MAPPING register 0x1040 to determine the current Bayer output
format (RGGB, GRBG, and so on). Using a Bayer format of RGGB, for example, the
fc2GetImageData method returns the following:

// Assuming image is 640 x 480
data[0] = Row 0, Column 0 = red pixel (R)
data[1] = Row 0, Column 1 = green pixel (G)
data[640] = Row 1, Column 0 = green pixel (G)
data[641] = Row 1, Column 1 = blue pixel (B)
```

Spinnaker C API

```
// Assuming image is 640 x 480 resolution. The current pixel format as well as
PixelColorFilter indicate the Bayer Tile Mapping for the camera. For example, BayerRG8 is
RGGB.

err = spinCameraGetNextImage(hCam, &hResultImage);
size_t imageSize;
spinImageGetBufferSize(hResultImage, &imageSize);

void **data;
data = (void**)malloc(imageSize * sizeof(void*));

spinImageGetData(hResultImage, data);

// Assuming image is 640 x 480
data[0] = Row 0, Column 0 = red pixel (R)
data[1] = Row 0, Column 1 = green pixel (G)
data[640] = Row 1, Column 0 = green pixel (G)
data[641] = Row 1, Column 1 = blue pixel (B)
```

Setting Number of Image Buffers

The following code snippet adjusts the number of image buffers that the driver initializes for buffering images on your PC to 11 (default is 10).

FlyCapture2 C API

```
FC2Config BufferFrame;
error = fc2GetConfiguration( context, &BufferFrame);
BufferFrame.numBuffers = 11;
error = fc2SetConfiguration( context, &BufferFrame);
```

Spinnaker C API

```
spinNodeHandle hGenTLNode = NULL;
int64_t bufferValue;
err = spinNodeMapGetNode
(hNodeMapGenTL, "StreamDefaultBufferCount", &hGenTLNode);
err = spinEnumerationEntryGetValue(hNodeMapGenTL, &bufferValue);
err = spinEnumerationSetIntValue(hNodeMapGenTL, 11);
```

Downloads and support

FLIR endeavors to provide the highest level of technical support possible to our customers. Most support resources can be accessed through the Support section of our website.

The first step in accessing our technical support resources is to obtain a customer login account. This requires a valid name and email address. To apply for a customer login account go to our downloads page.

Customers with a customer login account can access the latest **software** and **firmware** for their cameras from our website. We encourage our customers to keep their software and firmware up-to-date by downloading and installing the latest versions.

Finding information

Spinnaker SDK—The Spinnaker SDK provides API examples and the SpinView camera evaluation application. Available from our Downloads page.

API Documentation—The installation of the Spinnaker SDK comes with API references for C++, C#, and C code. A Programmer's Guide is included in each of the references. Available from:

- Start Menu→All Programs→Point Grey Spinnaker SDK→Documentation
- The SpinView application Help menu

Getting Started with SpinView—A quick guide to using the SpinView camera evaluation application provided in the Spinnaker SDK. Available from:

- Start Menu→All Programs→Point Grey Spinnaker SDK→Documentation
- The SpinView application Help menu
- Camera Reference zip package

Camera Reference—A zip package containing PDF and HTML copies of the camera's references, including: Installation Guide, Technical Reference, and Getting Started. Available from our downloads page.

Knowledge Base—A database of articles and application notes with answers to common questions as well as articles and tutorials about hardware and software systems. Available from our knowledge base.

Learning Center—Our Learning Center contains links to many resources including videos, case studies, popular topics, other application notes, and information on sensor technology.

Contacting technical support

Before contacting Technical Support, have you:

1. Read the product documentation?
2. Searched the knowledge base?
3. Downloaded and installed the latest version of software and/or firmware?

If you have done all the above and still can't find an answer to your question, contact our technical support team.

Additional resources

GenICam—A programming interface for cameras and devices. More information available on the EMVA.org website.

USB3 Vision—A vision standard for the USB 3.1 interface that uses GenICam. More information available on the AIA Vision Online website.