

Spinnaker Python Programmer's Guide

Revised 9/20/2023

Fundamentals of Spinnaker	2
Architecture of Spinnaker API	2
Examples	3
Nodes	4
QuickSpin API and Accessing Camera Parameters	4
C# Graphical User Interface API	5
Camera XML	7
Recommended Environment	7
Instantiate Cameras	7
Popular Features in Spinnaker	8
Enumeration	8
Asynchronous Hardware Triggering	8
Setting Black Level	9
Setting Exposure Time	9
Setting Gain	10
Setting Gamma	10
Setting White Balance	11
Accessing Raw Bayer Data	12
Setting Number of Image Buffers	12
Basic Features	13
Event Handling	13
Grabbing Images	14
Image Pointer Class	16
Error Handling	16
Loading and Saving Images	17
Advanced Features	18
Chunk Data	18
Sequencer	18
Logic Block	19
Logging	20

Fundamentals of Spinnaker

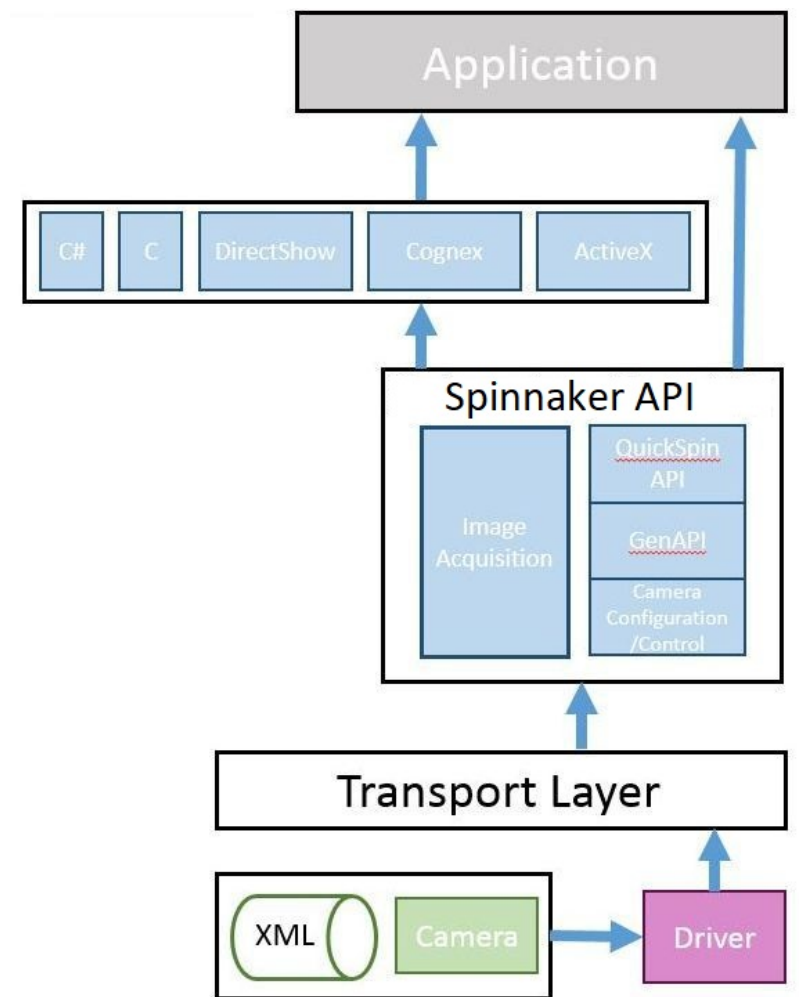
Architecture of Spinnaker API

Spinnaker API is built around the GenICam standard, which offers a generic programming interface for various cameras and interfaces. Spinnaker is an extension of GenAPI. Spinnaker provides quick and easy access to your camera.

Spinnaker API includes two major components:

Image Acquisition—This is the acquisition engine that is responsible for setting up image buffers and image grabbing.

Camera Configuration—This is the configuration engine that is responsible for controlling your camera. This component consists of QuickSpin API, which is a wrapper that makes GenAPI easy to use.



Examples

Included with 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.

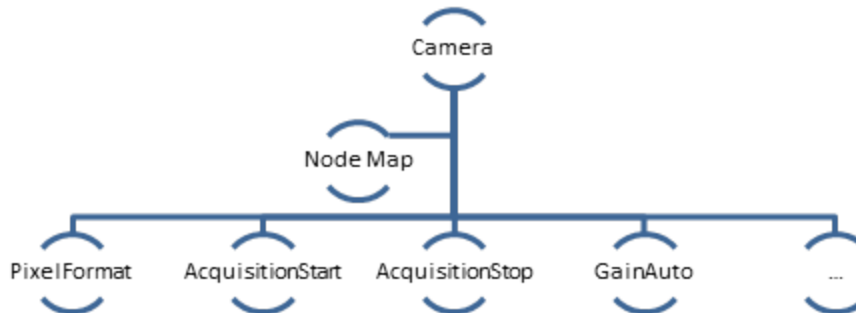
The table below describes the available Spinnaker SDK examples.

Spinnaker Example	Description
Acquisition	Enumerate, start acquisition, and grab images
AcquisitionMultipleCamera	How to capture images from multiple cameras simultaneously
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
ImageEvents	Image events shows how to acquire images using the image event handler.
ImageFormatControl*	Configure a custom image size and format
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
Sequencer (Blackfly S, Forge and Oryx only)	Capture multiple images with different parameters in a sequence
SpinSimpleGUI_DirectShow	Graphical User Interface for evaluating and setting camera parameters using DirectShow
SpinSimpleGUI_MFC	Graphical User Interface for evaluating and setting camera parameters
Trigger*	Trigger shows how to trigger the camera.

*Also available in QuickSpin

Nodes

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:

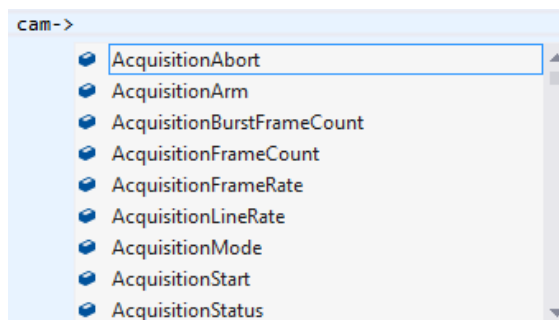
```

C++ GenAPI GenApi::INodeMap & nodeMap = cam.GetNodeMap();
CIntegerPtr width = nodeMap.GetNode("Width");
width->SetValue(new_width_val);
  
```

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	<pre>Spinnaker::GenApi::INodeMap & nodeMap = cam->GetNodeMap(); CFloatPtr GainNode = nodeMap.GetNode("Gain"); Float GainVal = GainNode->GetValue();</pre>
C++ QuickSpin API	<pre>float quickGainVal = cam->Gain.GetValue();</pre>

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 = newGUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

ImageDrawingWindow AcquisitionDrawing =
AcquisitionGUI.GetImageDrawingWindow();

AcquisitionDrawing.Connect(cam);

AcquisitionDrawing.Start();

AcquisitionDrawing.ShowModal();

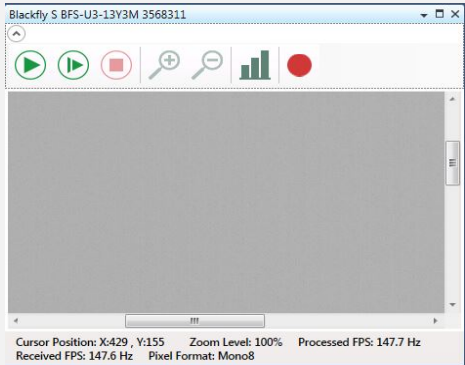
//To show camera selection window

GUIFactory AcquisitionGUI = newGUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

CameraSelectionWindow camSelection =
AcquisitionGUI.GetCameraSelectionWindow();

camSelection.ShowModal(true);
```



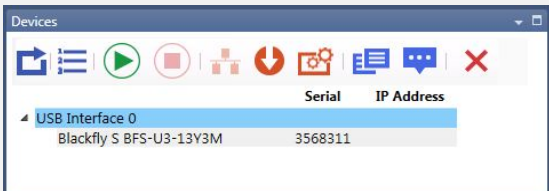
```
//To show camera selection window

GUIFactory AcquisitionGUI = newGUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

CameraSelectionWindow camSelection =
AcquisitionGUI.GetCameraSelectionWindow();

camSelection.ShowModal(true);
```



```
//To show property grid window

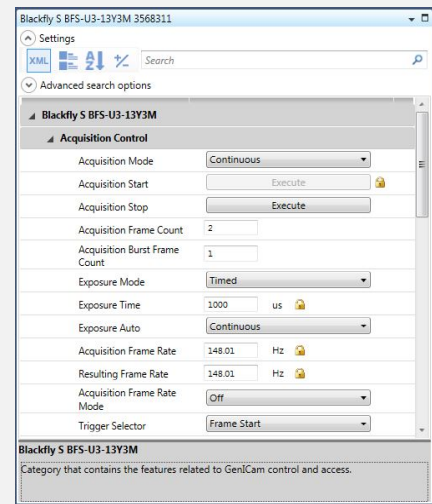
GUIFactory AcquisitionGUI = newGUIFactory ();

AcquisitionGUI.ConnectGUILibrary(cam);

PropertyGridWindow propWindow =
AcquisitionGUI.GetPropertyGridWindow();

propWindow.Connect(cam);

propWindow.ShowModal();
```



Dynamic GUI Control

```
GUIFactory dynamicGUI = newGUIFactory ();

dynamicGUI.ConnectGUILibrary(cam);

// Get dialog name via dynamicGUI.GetDialogNameList
()

Window dlg = dynamicGUI.GetDialogByName(dialogName);

dlg.Owner = Window .GetWindow(this );

dlg.Show();
```



Camera XML

The camera's XML file contains information such as feature naming, register mapping, and dependencies between features. It is typical for GenICam-compliant software to cache the XML file for quicker access to the camera's definition. Spinnaker caches the XML file in a binary format to achieve better performance.

Camera XML files are located in:

C:\ProgramData\Spinnaker\XML

Recommended Environment

Spinnaker supports the following list of operating systems and development environments.

OS Compatibility (32- and 64-bit)	Windows 7 Windows 8.1 Windows 10
Language Support	C C++ C# VB.NET Python
Compiler Support	Visual Studio 2015 Visual Studio 2017 Visual Studio 2019 Visual Studio 2022
Interface Support	USB3 Vision 1.0

Instantiate Cameras

Before you can instantiate a camera, you must create and initialize a system object. The System Singleton object is used to retrieve the list of interfaces (USB 3.1 or GigE) and cameras available. You must call `ReleaseInstance()` at the end of your program to free up the system object.

Multiple cameras can only be instantiated one at a time.

Instantiate multiple cameras (Python)

```
# Retrieve singleton reference to system object
system = PySpin.System.GetInstance()

cam_list = system.GetCameras()
num_Cameras = cam_list.GetSize()

for i, cam in enumerate(cam_list):

    cam.Init()

# Release system
system.ReleaseInstance()
```

Popular Features in Spinnaker

Enumeration

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

Spinnaker Python GenAPI	<pre> system = PySpin.System.GetInstance() cam_list = system.GetCameras() num_cameras = cam_list.GetSize() for cam_idx in range(num_cameras): cam = cam_list.GetByIndex(cam_idx) cam.Init() </pre>
-------------------------	---

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

Spinnaker Python QuickSpin API	<pre> Cam.TriggerMode.SetValue(PySpin.TriggerMode_On) Cam.TriggerSource.SetValue(PySpin.TriggerSource_Line0) Cam.TriggerSelector.SetValue(PySpin.TriggerSelector_FrameStart) Cam.TriggerActivation.SetValue(PySpin.TriggerActivation_RisingEdge) </pre>
Spinnaker Python GenAPI	<pre> node_trigger_mode = PySpin.CEnumerationPtr(node_map.GetNode("TriggerMode")) node_trigger_mode.SetIntValue(node_trigger_mode.GetEntryByName("On").GetValue()) node_trigger_source = PySpin.CEnumerationPtr(node_map.GetNode("TriggerSource")) node_trigger_source.SetIntValue(node_trigger_source.GetEntryByName("Line0").GetValue()) node_trigger_selector = PySpin.CEnumerationPtr(node_map.GetNode("TriggerSelector")) node_trigger_selector.SetIntValue(node_trigger_selector.GetEntryByName("FrameStart").GetValue()) node_trigger_activation = PySpin.CEnumerationPtr(node_map.GetNode("TriggerActivation")) node_trigger_activation.SetIntValue(node_trigger_activation.GetEntryByName("RisingEdge").GetValue()) </pre>

Setting Black Level

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.

Spinnaker Python QuickSpin API	<pre># Brightness is called black level in GenICam cam.BlackLevelSelector.SetValue(PySpin.BlackLevelSelector_All) # Set the absolute value of brightness to 1.5%. cam.BlackLevel.SetValue(1.5)</pre>
Spinnaker Python GenAPI	<pre>node_black_level_selector = PySpin.CEnumerationPtr(node_map.GetNode("BlackLevelSelector")) node_black_level_selector.SetIntValue(node_black_level_selector.GetEntryByName("All").GetValue()) node_black_level = PySpin.CFloatPtr(node_map.GetNode("BlackLevel")) node_black_level.SetValue(1.5)</pre>

Setting Exposure Time

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.

Spinnaker Python QuickSpin API	<pre># Turn off auto exposure cam.ExposureAuto.SetValue(PySpin.ExposureAuto_Off) # Set exposure mode to "Timed" cam.ExposureMode.SetValue(PySpin.ExposureMode_Timed) # Set exposure time to 20000 microseconds cam.ExposureTime.SetValue(20000.0)</pre>
Spinnaker Python GenAPI	<pre># Turn off auto exposure node_exposure_auto = PySpin.CEnumerationPtr(nodemap.GetNode('ExposureAuto')) entry_exposure_auto_off = node_exposure_auto.GetEntryByName('Off') node_exposure_auto.SetIntValue(entry_exposure_auto_off.GetValue()) # Set exposure mode to "Timed" node_exposure_mode = PySpin.CEnumerationPtr(nodemap.GetNode('ExposureMode')) entry_exposure_mode_timed = node_exposure_mode.GetEntryByName('Timed') node_exposure_mode.SetIntValue(entry_exposure_mode_timed.GetValue()) # Set exposure time to 20000 microseconds exposure_time = PySpin.CFloatPtr(nodemap.GetNode('ExposureTime')) exposure_time.SetValue(20000)</pre>

Setting Gain

The following code snippet adjusts gain to 10.5 dB.

Spinnaker Python
QuickSpin API

```
# Turn off auto gain
cam.GainAuto.SetValue(PySpin.GainAuto_Off)

# Set gain to 10.5 dB
cam.Gain.SetValue(10.5)
```

Spinnaker Python
GenAPI

```
# Turn off auto gain
node_gain_auto = PySpin.CEnumerationPtr(nodemap.GetNode('GainAuto'))
entry_gain_auto_off = node_gain_auto.GetEntryByName('Off')
node_gain_auto.SetIntValue(entry_gain_auto_off.GetValue())

# Set gain to 10.5 dB
gain = PySpin.CFloatPtr(nodemap.GetNode('Gain'))
gain.SetValue(10.5)
```

Setting Gamma

The following code snippet adjusts gamma to 1.5.

Spinnaker Python
QuickSpin API

```
# Make sure gamma is enabled
cam.GammaEnable.SetValue(True)

# Set gamma to 1.5
cam.Gamma.SetValue(1.5)
```

Spinnaker Python
GenAPI

```
# Make sure gamma is enabled
node_gamma_enable = PySpin.CBooleanPtr(nodemap.GetNode('GammaEnable'))
node_gamma_enable.SetValue(True)

# Set gamma to 1.5
gamma = PySpin.CFloatPtr(nodemap.GetNode('Gamma'))
gamma.SetValue(1.5)
```

Setting White Balance

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

Spinnaker Python
QuickSpin API

```
#Set auto white balance to off
cam.BalanceWhiteAuto.SetValue(PySpin.BalanceWhiteAuto_Off)

#Select blue channel balance ratio
cam.BalanceRatioSelector.SetValue(PySpin.BalanceRatioSelector_Blue)

#Set the white balance blue channel to 2
node_balance_ratio = PySpin.CFloatPtr(node_map.GetNode("BalanceRatio"))
node_balance_ratio.SetValue(2)

#Set the white balance red channel to 2
cam.BalanceRatioSelector.SetValue(PySpin.BalanceRatioSelector_Red)
node_balance_ratio.SetValue(2)
```

Spinnaker Python
GenAPI

```
node_balance_white_auto = PySpin.CEnumerationPtr(node_map.GetNode(
    "BalanceWhiteAuto"))
node_balance_white_auto.SetIntValue(node_balance_white_auto.GetEntryByName(
    "Off").GetValue())

node_balance_ratio_selector = PySpin.CEnumerationPtr(node_map.GetNode(
    "BalanceRatioSelector"))
node_balance_ratio_selector.SetIntValue(node_balance_ratio_selector.GetEntryByName(
    "Blue").GetValue())

node_balance_ratio = PySpin.CFloatPtr(node_map.GetNode("BalanceRatio"))
node_balance_ratio.SetValue(2)

node_balance_ratio_selector.SetIntValue(node_balance_ratio_selector.GetEntryByName(
    "Red").GetValue())
node_balance_ratio.SetValue(2)
```

Accessing Raw Bayer Data

Raw image data can be accessed programmatically by converting the Spinnaker Image class to numpy array using `GetNDArray`. 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.

Spinnaker Python
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.
```

```
result_image = cam.GetNextImage()  
data = result_image.GetNDArray()
```

```
# Assuming image is 640 x 480  
# data[0,0] = Row 0, Column 0 = red pixel (R)  
# data[0,1] = Row 0, Column 1 = green pixel (G)  
# data[1,0] = Row 1, Column 0 = green pixel (G)  
# data[1,1] = 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).

Spinnaker Python
API

```
s_node_map = cam.GetTLStreamNodeMap()
```

```
node_stream_buffer_count_manual = PySpin.CIntegerPtr(s_node_map.GetNode  
("StreamBufferCountManual"))
```

```
buffer_count = node_stream_buffer_count_manual.GetValue()  
node_stream_buffer_count_manual.SetValue(11)
```

Basic Features

Event Handling

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
Python

```
class InterfaceEventHandler(PySpin.InterfaceEventHandler):

    def OnDeviceArrival(self, camera):

        node_map_tldevice = camera.GetTLDeviceNodeMap()

        node_device_serial_number = PySpin.CStringPtr(node_map_tldevice.GetNode
("DeviceSerialNumber"))

        device_serial_number = node_device_serial_number.GetValue()

        print("A Camera arrived with serial number: {0}.\n".format(device_serial_number))

    def OnDeviceRemoval(self, camera):

        node_map_tldevice = camera.GetTLDeviceNodeMap()

        node_device_serial_number = PySpin.CStringPtr
        device_serial_number = node_device_serial_number.GetValue()

        print("A Camera removed with serial number: {0}.\n".format(device_serial_number))

handler = InterfaceEventHandler()
interface_list = system.GetInterfaces()

interface = interface_list.GetByIndex(0)

interface.RegisterEventHandler(handler)
```

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
Python

```
# Select the Exposure End event
node_event_selector = PySpin.CEnumerationPtr(node_map.GetNode("EventSelector"))
node_event_selector.SetIntValue(node_event_selector.GetEntryByName
("ExposureEnd").GetValue())

# Turn on the Event notification for Exposure End Event
node_event_notification_on = PySpin.CEnumerationPtr(node_map.GetNode
("EventNotification"))

node_event_notification_on.SetIntValue(node_event_notification_on.GetEntryByName
("On").GetValue())

# Once Exposure End Event is detected, the OnDeviceEvent function will be called
class DeviceEventHandler(PySpin.DeviceEventHandler):

    def OnDeviceEvent(self,eventname):

        print(;"\tDevice event %s with ID %i;" % (eventname,self.GetDeviceEventId()))

# Register event handler
device_event_handler = DeviceEventHandler()
cam.RegisterEventHandler(device_event_handler)
```

Grabbing Images

You can grab images using the `GetNextImage()` function. This function returns an image pointer for the current image. The image pointer should be released whenever you are done with the image. Image pointer, being a smart pointer, is automatically released when set to null or when out of scope.

Image Acquisition
Python

```
# Begin acquiring images
cam.BeginAcquisition()

result_image = cam.GetNextImage()

result_image.Release()
```

Grab Result

In almost all cases, you should check to see if the grabbed image has any errors. To do so, you need to call `getImageStatus()`.

To check for errors
in the image
Python

```
image_status = result_image.getImageStatus()
```

Available error
enums

```
# Status of images returned from GetNextImage() call.
```

```
class ImageStatus(Enum):
```

```
    IMAGE_NO_ERROR = 0
```

```
# Image is returned from GetNextImage() call without any errors. */
```

```
    IMAGE_CRC_CHECK_FAILED = 1
```

```
# Image failed CRC check.
```

```
    IMAGE_INSUFFICIENT_SIZE = 2
```

```
# Image size is smaller than expected.
```

```
    IMAGE_MISSING_PACKETS = 3
```

```
# Image has missing packets
```

```
    IMAGE_LEADER_BUFFER_SIZE_INCONSISTENT = 4
```

```
# Image leader is incomplete.
```

```
    IMAGE_TRAILER_BUFFER_SIZE_INCONSISTENT = 5
```

```
# Image trailer is incomplete.
```

```
    IMAGE_PACKETID_INCONSISTENT = 6
```

```
# Image has an inconsistent packet id.
```

```
    IMAGE_DATA_INCOMPLETE = 7
```

```
# Image data is incomplete.
```

```
    IMAGE_UNKNOWN_ERROR = 8
```

```
# Image has an unknown error.
```

Image Pointer Class

The image pointer (ImagePtr) is a smart pointer that points to the image object. You can have multiple image pointers pointing to the same object. Smart pointers automatically manage the life time of the object that it points to. You can also re-use the same image pointer object.

Image pointer should always be assigned before using.

Image Pointer Usage

```
result_image = PySpin.ImagePtr

# Retrieve the next received image
result_image = cam.GetNextImage()

duplicate_image = result_image
```

Image Pointer INCORRECT Usage

```
# Incorrect usage
illegal_image = PySpin.Imageptr
illegal_image.Create(...)
```

Image Pointer CORRECT Usage

```
# Correct usage
good_image = PySpin.Image.Create(...)
```

Error Handling

Spinnaker Python uses a try except block for exception handling.

Spinnaker Python API

```
#Assuming cam = PySpin.Camera
try:

    cam.Init()

except PySpin.SpinnakerException as ex:
    # Exception handling
```


Loading and Saving Images

Loading and saving a raw image (.raw) from disk into Spinnaker library can be achieved via Image class's smart pointer.

Loading and Saving Images

```
offline_image_width = 1280
offline_image_height = 1024
offline_offset_x = 0
offline_offset_y = 0

# Load image from disk into buffer (offline_data)

offline_data = np.fromfile(filename, dtype=np.ubyte)

# Create image using offline_data

load_image = PySpin.Image.Create(offline_image_width, offline_image_height, offline_offset_x, offline_offset_y, PySpin.PixelFormat_BayerRG8, offline_data)

# Convert image to mono8 data format

processor = PySpin.ImageProcessor()
result_image = processor.Convert(load_image, PySpin.PixelFormat_Mono8)

# Save image
result_image.Save("offline.jpg")
```

Advanced Features

Chunk Data

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.

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

Python Enable Chunk Data

```
cam.ChunkSelector.SetValue(PySpin.ChunkSelector_ExposureTime)
cam.ChunkEnable.SetValue(True)
cam.ChunkModeActive.SetValue(True)
```

Python Retrieve Chunk Data

```
chunk_data = rawImage.GetChunkData()
current_exposure = chunkData.GetExposureTime()
```

Sequencer

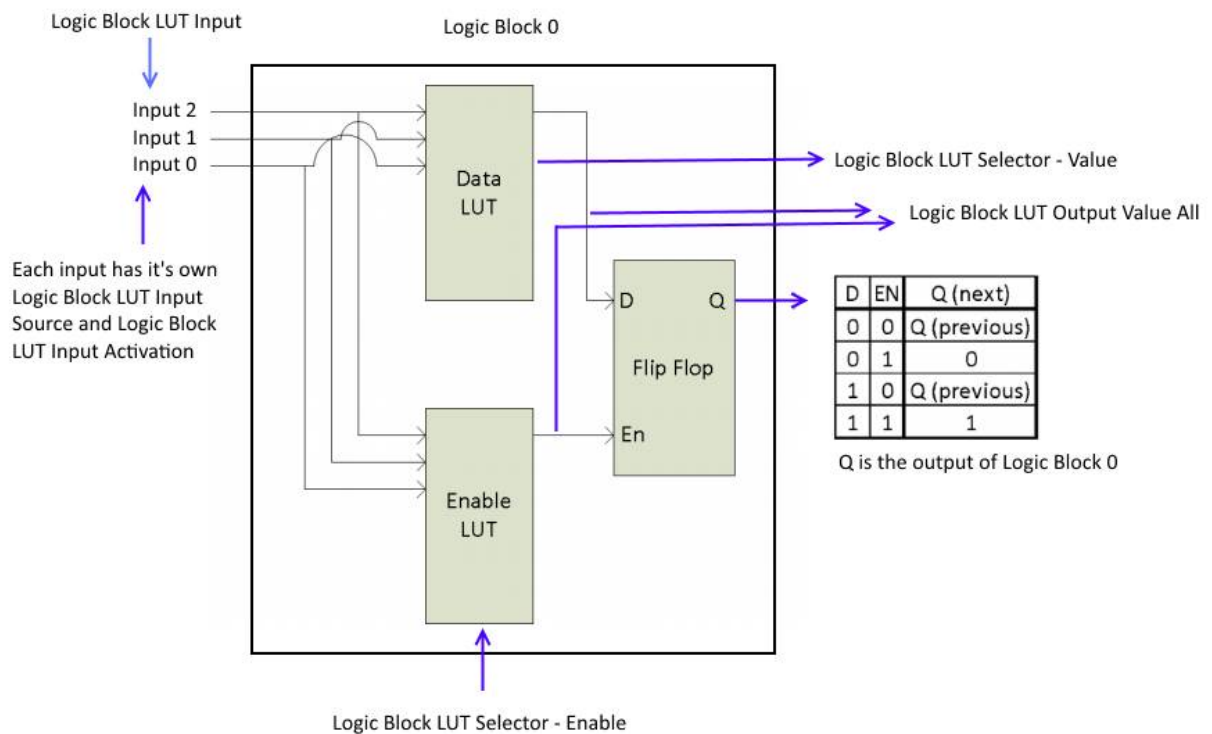
The purpose of a sequencer is to allow you to programmatically control the acquisition parameters of an image sequence. You can define not only how the images are captured (i.e. the camera feature settings) but also when the camera transitions from one acquisition setting to another. This is akin to a state machine diagram where the states correspond to the sequencer set feature settings, and the transition among states corresponds to a particular event that triggers the state machine to move from one state to another.

To configure sequencer on your camera, you can use SpinView's sequencer tab. Or, to programmatically configure it, you can use the python Sequencer source code example that come along with Spinnaker Python package.

Logic Block

A Logic Block is a collection of combinatorial logic and latches that allows the user to create new, custom signals inside the camera. Each Logic Block is comprised of 2 lookup tables (LUT) with programmable inputs, truth tables and a flip flop output. There is a LUT for both the D input (Value LUT) and the enable input (Enable LUT) of the flip flop. Both LUTs have 3 inputs and thus have 8 configuration bits for their truth table.

For more information, see [Using Logic Blocks](#).



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 python example of Logging, please see Spinnaker python package source code examples. By default, Spinnaker SDK's SpinView application saves all logging data to:

C:\ProgramData\Spinnaker\Logs

Register Logging
Python

```
# Retrieve singleton reference to system object
system = PySpin.System.GetInstance()

# Create and register the logging event handler
logging_event_handler = LoggingEventHandler()
system.RegisterLoggingEventHandler(logging_event_handler)

# Set callback priority level
LOGGING_LEVEL = PySpin.SPINAKEER_LOG_LEVEL_ERROR
system.SetLoggingEventPriorityLevel(LOGGING_LEVEL)

class LoggingEventHandler(PySpin.LoggingEventHandler):

    def OnLogEvent(self, logging_event_data):
        ...
```