Vision Point API

Version 2019.1

Data Book

May 2019

International Distributors

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
Winnington House, 2 Woodberry Grove
Finchley, London N12 0DR
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk

Contents

1 Figures and Tables 6
2 Revision History 7
3 Introduction 9
3.1 Safety Precautions 9
3.2 Disclaimer 10
4 Vision Point API Overview 11
4.1 Overview 11
4.2 Document structure 11
4.3 API usage in multi-threaded application....12
4.4 Function call sequence....13
5 Frame Grabber Operational Configurations 15
5.1 Frame Grabber General Configuration for Operation....15
5.2 Silent Discovery Mode 15
Komodo 4R4T system configuration example....15
5.3 Segment accumulation....17
6 Connection and Info....18
6.1 KYFG_Scan()....18
6.2 KYFG_Open()....19
6.3 KYFG_OpenEx() 19
6.4 KYFG_SetGrabberConfigurationParameterCallback() (C++ only)....20
6.5 KYFG_GetGrabberConfigurationParameterDefinitions() (C++ only) 20
6.6 KYFG_Close() 22
6.7 KYFG_Pid2Name() (DEPRECATED) 23
6.8 KY_DeviceDisplayName() (DEPRECATED) 23
6.9 KY_DeviceInfo()....23
7 Camera Configurations 25
7.1 KYFG_CameraScan() 25
7.2 KYFG_UpdateCameraList()....25
7.3 KYFG_CameraOpen2()....26
7.4 KYFG_CameraOpen() (DEPRECATED) 26
7.5 KYFG_CameraClose() 27
7.6 KYFG_CameraInfo()....27
7.7 KYFG_CameraGetXML()....28
7.8 KYFG_GetXML() (DEPRECATED)....30
7.9 KYFG_SetCameraConfigurationParameterCallback() (C++ only) 31
7.10 KYFG_GetCameraConfigurationParameterDefinitions() (C++ only)....32
8 Callback Functions....33
8.1 KYFG_CallbackRegister()(DEPRECATED)....33
8.2 KYFG_CallbackUnregister()(DEPRECATED) 33
8.3 KYFG_CameraCallbackRegister()....34
8.4 KYFG_CameraCallbackUnregister() 34
8.5 KYFG_StreamBufferCallbackRegister()....36
8.6 KYFG_StreamBufferCallbackUnregister() 36
8.7 KYFG_AuxDataCallbackRegister()....37
8.8 KYFG_AuxDataCallbackUnregister() 37
8.9 KYDeviceEventCallBackRegister() 38
8.10 KYDeviceEventCallBackUnregister()....38
9 Camera/Frame Grabber Values 40
9.1 KYFG_SetCameraValue() / KYFG_SetGrabberValue()....40
9.1.1 KYFG_SetCameraValueInt() / KYFG_SetGrabberValueInt()....41
9.1.2 KYFG_SetCameraValueFloat() / KYFG_SetGrabberValueFloat() 41
9.1.3 KYFG_SetCameraValueBool() / KYFG_SetGrabberValueBool()....42
9.1.4 KYFG_SetCameraValueEnum() / KYFG_SetGrabberValueEnum() 42
9.1.5 KYFG_ExecuteCommand()/KYFG_ExecuteGrabberCommand() (DEPRECATED)....43

9.1.6 KYFG_CameraExecuteCommand() / KYFG_GrabberExecuteCommand() 44
9.1.7 KYFG_SetCameraValueString() / KYFG_SetGrabberValueString() 44
9.1.8 KYFG_SetCameraValueEnum_ByValueName() / KYFG_SetGrabberValueEnum_ByValueName() 45
9.2 KYFG_GetCameraValueType() / KYFG_GetGrabberValueType() 46
9.3 KYFG_GetCameraValue() / KYFG_GetGrabberValue() 46
9.3.1 KYFG_GetCameraValueInt() / KYFG_GetGrabberValueInt() 47
9.3.2 KYFG_GetCameraValueEnum() / KYFG_GetGrabberValueEnum() 48
9.3.3 KYFG_GetCameraValueFloat() / KYFG_GetGrabberValueFloat() 48
9.3.4 KYFG_GetCameraValueBool() / KYFG_GetGrabberValueBool() 49
9.3.5 KYFG_GetCameraValueStringCopy()/KYFG_GetGrabberValueStringCopy() 49
9.3.6 KYFG_GetCameraValueString() / KYFG_GetGrabberValueString() 51

10 Authentication interface 53

10.1 KY_AuthProgramKey() 53
10.2 KY_AuthVerify() 53

11 Buffer interface (DEPRECATED) 55

11.1 KYFG_BufferAlloc() (DEPRECATED) 55
11.2 KYFG_BufferDelete() (DEPRECATED) 56
11.3 KYFG_BufferGetSize() (DEPRECATED) 56
11.4 KYFG_BufferGetFrameIndex() (DEPRECATED) 56
11.5 KYFG_BufferGetPtr() (DEPRECATED) 57
11.6 KYFG_BufferGetAux() (DEPRECATED) 57

12 Stream Interface 59

12.1 Stream Interface description 59
12.2 KYFG_StreamCreateAndAlloc() 60
12.3 KYFG_StreamCreate() 61
12.4 KYFG_StreamGetInfo() 61
12.5 KYFG_StreamGetSize() 62
12.6 KYFG_StreamGetFrameIndex() 63
12.7 KYFG_StreamGetPtr() 63
12.8 KYFG_StreamGetAux() 64
12.9 KYFG_BufferAllocAndAnnounce() 64
12.10 KYFG_BufferAnnounce() 65
12.11 KYFG_BufferGetInfo() 66
12.12 KYFG_BufferToQueue() 67
12.13 KYFG_BufferQueueAll() 68
12.14 KYFG_StreamDelete() 69
12.15 Example of code using Cyclic buffers 69
12.16 Example of code using Queued buffers 70

13 Data acquisition 73

13.1 KYFG_CameraStart() 73
13.2 KYFG_CameraStop() 73

14 Low level bootstrap access 74

14.1 KYFG_ReadPortReg() 74
14.2 KYFG_ReadPortBlock() 74
14.3 KYFG_WritePortReg() 75
14.4 KYFG_WritePortBlock() 75
14.5 KYFG_CameraReadReg() 76
14.6 KYFG_CameraWriteReg() 76

15 IO configurations 77

15.1 IO selectors 77
15.2 Configuration fields 77

16 Firmware update 80

16.1 KYFG_CheckUpdateFile 80
16.2 KYFG_LoadFirmware 81

17 API Defines and Macros 82

17.1 API handles 82
17.2 KYBOOL 82

17.3 FGCallback ... 82
17.4 StreamBufferCallback ... 82
17.5 CameraCallback ... 82
17.6 FGAuxDataCallback ... 83
17.7 KYDeviceEventCallBack ... 83
17.8 ParameterCallback ... 83
17.9 UPDATE_CALLBACK ... 83
18 Enumerations ... 84
18.1 FGSTATUS ... 84
18.2 CXP_LINK_SPEED ... 85
18.3 KY_CAM_PROPERTY_TYPE ... 85
18.4 VIDEO_DATA_WIDTH ... 85
18.5 VIDEO_DATA_TYPE ... 86
18.6 VIDEO_DATA_SUBTYPE ... 87
19 Structures ... 88
19.1 KYFGCAMERA_INFO ... 88
19.2 KY_STREAM_BUFFER_INFO_CMD ... 88
19.3 VIDEO_PIXELIF ... 88
19.4 KYFG_AUX_DATA ... 89
19.5 KYFG_FRAME_AUX_DATA ... 89
19.6 KYFG_IO_AUX_DATA ... 90
19.7 KYDEVICE_EVENT ... 92
19.8 KYDEVICE_EVENT_CAMERA_START ... 92
19.9 KYDEVICE_EVENT_CAMERA_CONNECTION_LOST ... 93
19.10 NodeDescriptor ... 93
19.10.1 ParameterInterfaceType ... 94
19.10.2 ParameterRepresentation ... 95
19.10.3 NodeDescriptorType ... 95
19.11 KY_AuthKey ... 95
19.12 UPDATE_STATUS ... 95
20 Python API ... 97
20.1 Connection and Info ... 97
20.1.1 KYFG_Open() ... 97
20.1.2 KYFG_Scan() ... 97
20.1.3 KY_DeviceDisplayName() (DEPRECATED) ... 98
20.1.4 KY_DeviceInfo() ... 98
20.2 Camera Configurations ... 101
20.2.1 KYFG_CameraScan() ... 101
20.2.2 KYFG_CameraOpen2() ... 101
20.2.1 KYFG_CameraInfo() ... 102
20.2.1 KYFG_CameraGetXML() ... 102
20.3 Callback functions ... 103
20.3.1 KYFG_CallbackRegister() ... 103
20.3.1 KYFG_StreamBufferCallbackRegister() ... 104
20.4 Camera/Frame Grabber Values ... 105
20.4.1 KYFG_GetCameraValue() ... 105
20.4.2 KYFG_GetGrabberValue() ... 105
20.4.3 KYFG_SetCameraValue() ... 106
20.4.4 KYFG_SetGrabberValue() ... 106
20.4.5 KYFG_CameraExecuteCommand() ... 106
20.4.6 KYFG_GrabberExecuteCommand() ... 107
20.5 Stream Interface ... 107
20.5.1 KYFG_StreamGetFrameIndex() ... 107
20.5.2 KYFG_StreamCreateAndAlloc() ... 108
20.5.1 KYFG_StreamCreate() ... 108
20.5.2 KYFG_StreamGetInfo() ... 109
20.5.3 KYFG_BufferAnnounce() ... 109
20.5.4 KYFG_BufferGetInfo() ... 110

20.5.5 KYFG_BufferToQueue() 111
20.5.6 KYFG_BufferQueueAll() 112
20.6 Data acquisition 112
20.6.1 KYFG_CameraStart() 112
20.6.2 KYFG_CameraStop() 113
20.6.3 KYFG_StreamGetPtr() 113
21 .NET API 114
21.1 public ref class Lib 114
21.1.1 Lib() 114
21.1.2 int Lib::Scan(); 114
21.1.3 System::String Lib.DeviceDisplayName() (DEPRECATED) 114
21.1.4 DEVICE_INFO Lib.DeviceInfo () 114
21.1.5 IGrabber Lib.Open(); 115
21.1.6 IGrabber Lib.OpenEx() 115
21.2 public interface class IDevice 116
21.2.1 System.Collections.Generic.List<ICamera> IDevice.CameraScan(); 116
21.2.2 void IDevice.Close() 116
21.2.3 void IDevice.SetValue(System.String paramName, System.Object paramValue) 116
21.2.4 System.Object IDevice.GetValue(System.String paramName); 116
21.2.5 void IDevice.ExecuteCommand(System.String paramName) 117
21.2.6 void IDevice.WritePortBlock(int port, System.UInt64 address, array<System.Byte> buffer) 117
21.2.7 void IDevice.WritePortReg(int port, System.UInt64 address, System.UInt32 data) 117
21.2.8 array<System.Byte> IDevice.ReadPortBlock(int port, System.UInt64 address, System.UInt32 size) 117
21.2.9 System::UInt32 IDevice.ReadPortReg(int port, System.UInt64 address) 118
21.2.10 void IDevice.AuthProgramKey(array<System.Byte> key, int lock) 118
21.2.11 int IDevice.AuthVerify(array<System.Byte> key) 118
21.2.12 void IDevice.AuxDataCallbackRegister(FGAuxDataCallback delegator, Object userContext) 118
21.2.13 void IDevice.AuxDataCallbackUnregister(FGAuxDataCallback delegator) 119
21.3 public interface class ICamera 119
21.3.1 CAMERA_INFO^ KYFG_CameraInfo() 119
21.3.2 void ICamera.Open(System.String xml_file_path) 120
21.3.3 void ICamera.Close() 120
21.3.4 void ICamera.Start(IStream streamHandle, int frames) 120
21.3.5 void ICamera.Stop() 120
21.3.6 void ICamera.SetValue(System.String paramName, Object paramValue) 120
21.3.7 Object ICamera::GetValue(System.String paramName) 121
21.3.8 void ICamera.ExecuteCommand(System.String paramName) 121
21.3.9 IStream ICamera.StreamCreateAndAlloc(int frames) 121
21.3.10 IStream ICamera.StreamCreate() 122
21.3.11 System.Tuple<byte[], KYBOOL> ICamera.GetXML() 122
21.3.12 System.UInt32 ICamera.WriteReg(System.UInt64 address, array<System.Byte> buffer) 123
21.3.13 array<System.Byte> ICamera.ReadReg(System.UInt64 address, System.UInt32 size) 123
21.3.14 void ICamera.CameraCallbackRegister(CameraCallback delegator, Object userContext) 123
21.3.15 void ICamera.CameraCallbackUnregister(CameraCallback delegator) 123
21.4 public interface class IStream 124
21.4.1 void IStream.BufferCallbackRegister(StreamBufferCallback^ delegator, Object^ userContext) 124
21.4.2 void IStream.BufferCallbackUnregister(StreamBufferCallback^ delegator) 124
21.4.3 Object^ IStream.GetPtr(int buffIndex) 124
21.4.4 void IStream.Delete() 124
21.4.5 Object IStream.GetInfo(KY_STREAM_INFO_CMD info) 125
21.4.6 AuxData IStream.GetAux(int frame) 125
21.4.7 IStreamBuffer IStream.BufferAllocAndAnnounce(System.UInt64 nBufferSize) 125
21.4.8 IStreamBuffer IStream.BufferAnnounce(array<System.Byte> pBuffer) 125
21.4.9 void IStream.BufferQueueAll(KY_ACQ_QUEUE_TYPE srcQueue, KY_ACQ_QUEUE_TYPE dstQueue) 126
21.5 public interface class IStreamBuffer 126
21.5.1 int IStreamBuffer.GetFrameIndex(); 126
21.5.2 Object IStreamBuffer.GetPtr(); 126
21.5.3 Object IStreamBuffer.GetInfo(KY_STREAM_BUFFER_INFO_CMD info); 126

21.5.4 System.UInt64 IStreamBuffer.getSize() 127
21.5.5 void IStreamBuffer.BufferToQueue(KY_ACQ_QUEUE_TYPE dstQueue) 127
22 Building an API example 128
22.1 API example for Windows 128
22.2 API example for Linux 130
23 Appendices 132
23.1 Firmware version 1.xx line selector enumeration 132
23.2 Firmware version 1.xx I/O source enumeration 132
24 Troubleshooting 134
24.1 Updating the device firmware using Vision Point Application 134
24.2 Updating the device firmware using pre-built utility for Linux 135
24.3 Updating the device firmware using pre-built utility for Windows 136
24.4 Collecting log files 137
24.5 Technical Support and Professional Services 139
24.6 Submitting a support request 139

1 Figures and Tables

Index of Tables

TABLE 1 : REVISION HISTORY 8
TABLE 2 : MESSAGEID POSSIBLE VALUES 89
TABLE 3 : IO CONTROLLER AUXILIARY DATA BIT MASK INTERPRETATION........ 92
TABLE 4 : DEVICE EVENT ID POSSIBLE VALUES........ 92
TABLE 5 : LINE SELECTION OPTIONS (FIRMWARE VERSION 1.XX) 132
TABLE 6 : FRAME GRABBER I/O SOURCE (FIRMWARE VERSION 1.XX)........ 133

Index of Figures

FIGURE 1 : FUNCTION CALL SEQUENCE........ 13
FIGURE 2 : SILENT CAMERA DISCOVERY EXAMPLE 16
FIGURE 3 : FIRMWARE UPDATE SELECTION WINDOW 134
FIGURE 4 : FIRMARE UPDATE CONFIRMATION WINDOW........ 134
FIGURE 5 : FIRMARE UPDATE PROCESS WINDOW........ 135
FIGURE 6 : FIRMARE UPDATE VIA TERMINAL PROCESS WINDOW........ 136
FIGURE 7 : FIRMARE UPDATE VIA COMMAND LINE PROCESS WINDOW 137
FIGURE 8 : LOG FILES FOLDER FROM THE QUICK START MENU PATH........ 138
FIGURE 9 : LOG FILES FOLDER FROM VISION POINT HELP MENU PATH 138

2 Revision History

Version	Date	Notes
1.0	08/2014	Initial release
1.1	09/2014	Predator API release 1.0.5.1 • Fix parameter types signature. • Additional working examples.
1.2	12/2014	Predator API release 1.0.5.136 • New Frame Grabber configuration parameters. • Additional working examples for I/O configuration.
1.3	07/2015	Predator API release 1.0.8.362 • New camera specific callback KYFG_CameraCallbackRegister() • Change of KYFG_SetGrabberValue() / KYFG_GetGrabberValue() device handle (support backward compatibility)
2.0	10/2015	Predator API release 2.0.1.484 • New functions KYFG_CameraGetXML() and KYFG_GetCameraValueStringCopy() were added to overcome development environment allocation issues • I/O selector and source enumeration values were reordered to support firmware release 2.xx • Additional configuration parameters to Digital I/O triggers, encoders and camera triggers
3.0	03/2016	Vision Point API release 3.0.0 • New function KYFG_OpenEx() has been added to support save of Frame Grabber configurations before camera discovery. • KYFG_CameraOpen() has been deprecated and replaced with KYFG_CameraOpen2() • KYFG_AuxDataCallbackRegister() for Auxiliary data retrieval, from varies sources and components, has been added. Consequently, additional data structures and definitions were added for this purpose. • KYFG_BufferGetAux() was added to extract the Frame Auxiliary data upon new frame arrival. Frame Auxiliary data includes frame sequence number and its timestamp.
4.0	11/2016	Visio Point API release 4.0.0 • KYFG_ReadPortBlock() and KYFG_WritePortBlock() functions were added to support direct block read/write in opposed to single register operation. • KYFG_Pid2Name() was deprecated and replaced with KY_DeviceDisplayName() • Authentication Chip interface was added. Please see Authentication Interface chapter • Old buffer handling interface was deprecated and renamed. Please see Buffer Interface chapter for more details. • New Queued buffer interface supporting user buffers was added. Please see Stream interface chapter for new functions description.
4.1	07/2017	Visio Point API release 4.1 • Support for both Camera Simulator and Frame Grabber • Bug fixes and improvements
4.2	09/2017	Visio Point API release 4.2 • Added API functions for JetCam HS Camera support • New troubleshoot and API example sections
4.3	04/2018	Visio Point API release 4.3 • New Python API • New .NET API
4.4	09/2018	Visio Point API release 4.4 • Support for Gen<i>Cam IRegister type in GUI • Saving video buffer - additional file output formats

5.0	03/2019	Visio Point API release 5.0 • Windows service “KYService” and display name “KAYA Instruments Service” installation. • Automatic monitoring and management of PoCXP for CoaXPress cameras. Note: The software stack requires “KYService” to be running, otherwise KYFG_Scan() will return 0 and KYFG_Open()/KYFG_OpenEx() will return INVALID_FGHANDLE. • KYFGLib_Initialize() - optional call before “KYFGScan” and reserved for future usage. • Genicam and OpenCV libraries are not installed to Vision Point's "bin" folder which is added to system's PATH. Instead, will be installed into a sub-folder for internal use only. If these libraries are needed by user's application, it should be installed separately. • Visual Studio 2017 flavor support. User will be able to use our libraries linked to Visual Studio 2017 flavor on run-time: KYFGLib_vc141.dll clserkyi_vc141.dll
5.0.1	05/2019	Visio Point API release 5.0.1 • New function KYFG_UpdateCameraList() updates the list of cameras connected to the device. Currently open camera handles are not affected by this function. • Event callback when a camera lost connection • GigE: Fix Issue with packed/unpacked PixelFormat Remote device communication enhancement (similar to CLHS) Control the source port on each channel • CoaXPress: Option to overwrite ALL size of ControlPacketDataSize via Grabber configurations • Virtual Grabber: Add new versioning register to support new suppresion of old device version • GenTL: Reset STREAM_INFO_NUM_DELIVERED on each new stream start Improved mechanism for EventGetData() function Override camera's xml file using exteranl KYFGLib.json and x.fgprj file

Table 1 : Revision History

3 Introduction

3.1 Safety Precautions

With your KAYA’s Frame Grabber board in hand, please take a minute to read carefully the precautions listed below in order to prevent unnecessary injuries to you or other personnel or cause damage to property.

- **Before using the product, read these safety precautions carefully to assure correct use.**
- **These precautions contain serious safety instructions that must be observed.**
- **After reading through this manual, be sure to act upon it to prevent misuse of product.**

⚠ Caution

In the event of a failure, disconnect the power supply. If the product is used as is, a fire or electric shock may occur. Disconnect the power supply immediately and contact our sales personnel for repair.
If an unpleasant smell or smoking occurs, disconnect the power supply. If the product is used as is, a fire or electric shock may occur. Disconnect the power supply immediately. After verifying that no smoking is observed, contact our sales personnel for repair.
Do not disassemble, repair or modify the product. Otherwise, a fire or electric shock may occur due to a short circuit or heat generation. For inspection, modification or repair, contact our sales personnel.
Do not touch a cooling fan. As a cooling fan rotates in high speed, do not put your hand close to it. Otherwise, it may cause injury to persons. Never touch a rotating cooling fan.
Do not place the product on unstable locations. Otherwise, it may drop or fall, resulting in injury to persons or failure.
If the product is dropped or damaged, do not use it as is. Otherwise, a fire or electric shock may occur.
Do not touch the product with a metallic object. Otherwise, a fire or electric shock may occur.
Do not place the product in dusty or humid locations or where water may splash. Otherwise, a fire or electric shock may occur.
Do not get the product wet or touch it with a wet hand. Otherwise, the product may break down or it may cause a fire, smoking or electric shock.
Do not touch a connector on the product (gold-plated portion). Otherwise, the surface of a connector may be contaminated with sweat or skin oil, resulting in contact failure of a connector or it may cause a malfunction, fire or electric shock due to static electricity.
Do not use or place the product in the following locations. • Humid and dusty locations

- Airless locations such as closet or bookshelf
- Locations which receive oily smoke or steam
- Locations close to heating equipment
- Closed inside of a car where the temperature becomes high
- Static electricity replete locations
- Locations close to water or chemicals

Otherwise, a fire, electric shock, accident or deformation may occur due to a short circuit or heat generation.

Do not place heavy things on the product.

Otherwise, the product may be damaged.

3.2 Disclaimer

This product should be used for CoaXPress video acquisition and camera signals and triggers control. KAYA Instruments assumes no responsibility for any damages resulting from the use of this product for purposes other than those stated.

Even if the product is used properly, KAYA Instruments assumes no responsibility for any damages caused by the following:

- Earthquake, thunder, natural disaster or fire resulting from the use beyond our responsibility, acts caused by a third party or other accidents, the customer’s willful or accidental misuse or use under other abnormal conditions.
- Secondary impact arising from use of this product or its unusable state (business interruption or others).
- Use of this product against the instructions given in this manual or malfunctions due to connection to other devices.

KAYA Instruments assumes no responsibility or liability for:

- Erasure or corruption of data arising from use of this product.
- Any consequences or other abnormalities arising from use of this product, or damage of this product not due to our responsibility or failure due to modification.

4 Vision Point API Overview

4.1 Overview

The purpose of this document is to list and demonstrate the provided functionality of KAYA Frame Grabbers' API.

This API is to be used with KAYA's Frame Grabbers hardware provided by KAYA Instruments.

This is a high level API for connecting, configuring and capturing a CoaXPress stream of data over 1, 2 or 4 CXP channels. KAYA's Frame Grabbers are capable of connecting to various CoaXPress cameras at various speeds and topologies.

4.2 Document structure

This API guide is divided into few major topics each responsible for different functionalities:

- Connection and Info
  - ✓ Connect/disconnect to a specific Frame Grabber
- Camera Configurations
  - ✓ Scan, connect and get camera information
  - ✓ Use camera native XML or override with another XML file
- Callback Functions
  - ✓ Callback functions for data acquisition
- Camera/Frame Grabber values
  - ✓ Use XML fields to configure camera/Frame Grabber parameters, according to Gen<i>Cam standard naming and XML field definition and type
- Buffer data interface
  - ✓ Access and handle of each allocated buffer in memory
- Data acquisition
  - ✓ Acquisition of data stream
- Low level bootstrap access
  - ✓ Write/read to camera bootstrap space directly with no enforcement
- IO Configurations
  - ✓ Control of external IO pins: inputs, user outputs, triggers, timers and encoders.
- Defines, Macros, Structures and Enumerations
  - ✓ All available parameters types and definitions that can be also used in the host application

These can be found under “<installation folder>/Vision Point/include”.

- Configuration parameters
  - ✓ KAYA additional Gen<i>Cam configuration parameters for controlling, analyzing and configuring the system.

4.3 API usage in multi-threaded application

Vision Point API is NOT thread-safe. This means that if a calling application accesses the resources listed below from multiple threads, the serialization of such accesses should be implemented by that application.

Resources that require serialized access are:

- Device accessed via an instance of FGHANDLE.
- Camera accessed via an instance of CAMHANDLE
- Stream assessed via an instance of STREAM_HANDLE
- A frame buffer accessed via an instance of STREAM_BUFFER_HANDLE

4.4 Function call sequence

In order for the API to carry out the desired results the following sequence of function calls should be followed:

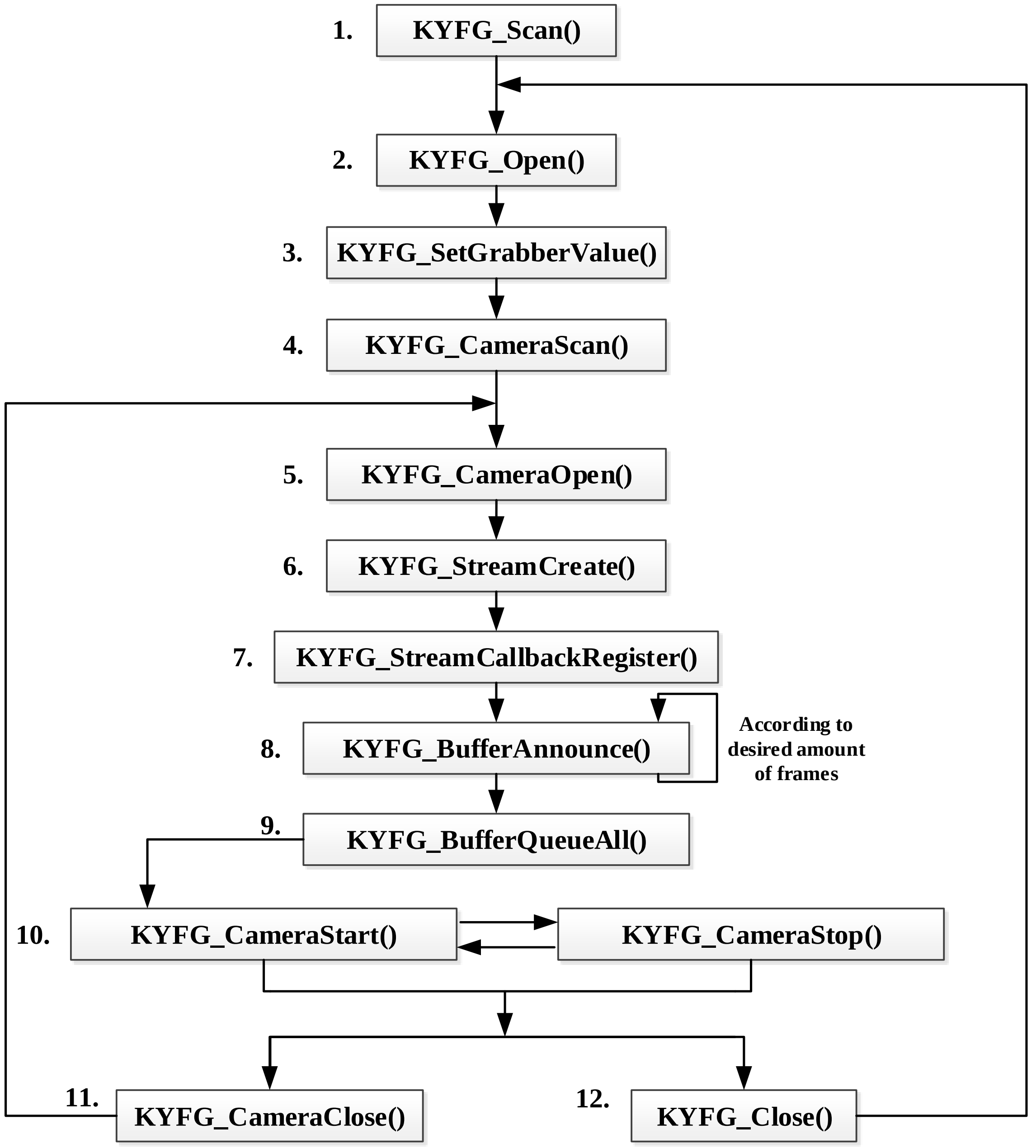


Figure 1 : Function call sequence

1. Scan for Frame Grabbers currently connected to the PC. This will return an array of found hardware and Virtual device PID’s.
2. Open a connection to a selected Frame Grabber. Use the index corresponding to a Frame Grabber from the array acquired in previous step.

3. Scan for currently connected cameras. There is no restriction on connection topology or CoaXPress camera speed.
4. A callback is an optional function which can be registered after camera was found in the scan process. In order to work properly, the callback should be registered before actually starting the stream acquisition. This callback will be called upon each frame reception from a specific camera. With the callback function, a handle to the relevant buffer will arrive. Use the Buffer interface functions to retrieve currently acquired frame. KYFG_CameraOpen() and KYFG_CameraClose() doesn't invalidate the callback registration.
5. Open a connection to selected camera and load native or external XML file.
6. Set the desired values to determine the camera parameters using the different available methods.
7. Allocate the memory required for acquiring the video stream. Several frames should be allocated in order not to immediately run over previously received data.
8. Start/Stop the acquisition from camera.
9. If you no longer want to continue acquisition from active camera, close the connection to chosen camera.

   To connect back to the camera and only if this camera wasn't physically disconnected, no camera scan is needed, and KYFG_CameraOpen () can be called.

5 Frame Grabber Operational Configurations

5.1 Frame Grabber General Configuration for Operation

In general, there is no need to configure the Frame Grabber to achieve basic operation.

To activate certain advanced features, few Frame Grabber configurations are required. These can be done using the KYFG_SetGrabberValue() function or one of the provided sub-functions.

The complete set of Frame Grabber configuration parameters can be found in "KAYAs_Frame_Grabber_Programming_Start-up_Guide" document.

5.2 Silent Discovery Mode

Silent camera discovery process is mainly used for retransmit applications. A silent scan for connected cameras is made without resetting any camera parameters (i.e. no writes are made to the camera. Nevertheless multiple reads are made).

If needed, camera Reset sequence and speed configuration should be performed from external source before a camera scan can be initiated using this mode.

To activate the Silent Discovery Mode the following steps should be taken:

1. Scan and connect to a chosen Frame Grabber.
2. Set the "SilentDiscovery" value to "On" (int value: 1) using the KYFG_SetGrabberValue() function or one of the provided sub-functions. Please see "KAYA's_Frame_Grabber_Programming_Start-up_Guide" document for more details.
3. Make sure camera is already configured and ready to be connected to. Take under account that no camera Reset or connection reconfiguration commands will be sent.
4. Now camera scan can be initiated using the KYFG_CameraScan() function.

Komodo 4R4T system configuration example

This configuration should be used on the Komodo or Predator Frame Grabber when setting up the Komodo4R4T transmit channels towards the Frame Grabber receive channels.

1. Insert the Komodo/Predator Frame Grabber and the Komodo4R4T Frame Grabber into a PC and connect the power connector to the Komodo4R4T Frame Grabber device.

   The Komodo/Predator Frame Grabber and the Komodo4R4T Frame Grabber can be installed in a single or in two different computer devices.

2. Connect a CXP camera or the Chameleon Simulator to one or more of the 4 top DIN connectors (channels 0-3) of the Komodo4R4T using 4 DIN cables.
3. Connect the same bottom DIN connectors (channels 4-7) of Komodo4R4T to Komodo/Predator Frame Grabber using DIN cables.
4. Make sure the Komodo4R4T links connected in the same order (link 0 of the will be retransmitted to link 4). See image below as reference.
5. Open Vision Point application and choose the Komodo4R4T board
6. Open additional window of Vision Point application and choose the Komodo/Predator Frame Grabber board.
7. Activate the “Silent Discovery Mode” for Komodo/Predator Frame Grabber. This option located in Frame Grabber tab -> Device control category -> Silent Discovery Mode - ON
8. Scan camera on the Komodo4R4T – this will initiate camera correctly to be ready for silent discovery

   NOTE: For Chameleon Simulator configuration, one should open Vision Point application and configure the link number for the Simulator to 1-4 links in Camera tab -> CXP category, prior step no. 5
9. Scan camera on the Komodo Frame Grabber
10. Press start acquisition on Komodo Frame Grabber – this won’t start the acquisition yet
11. Press start acquisition for Komodo4R4T Frame Grabber – this will initiate acquisition on both Frame Grabbers

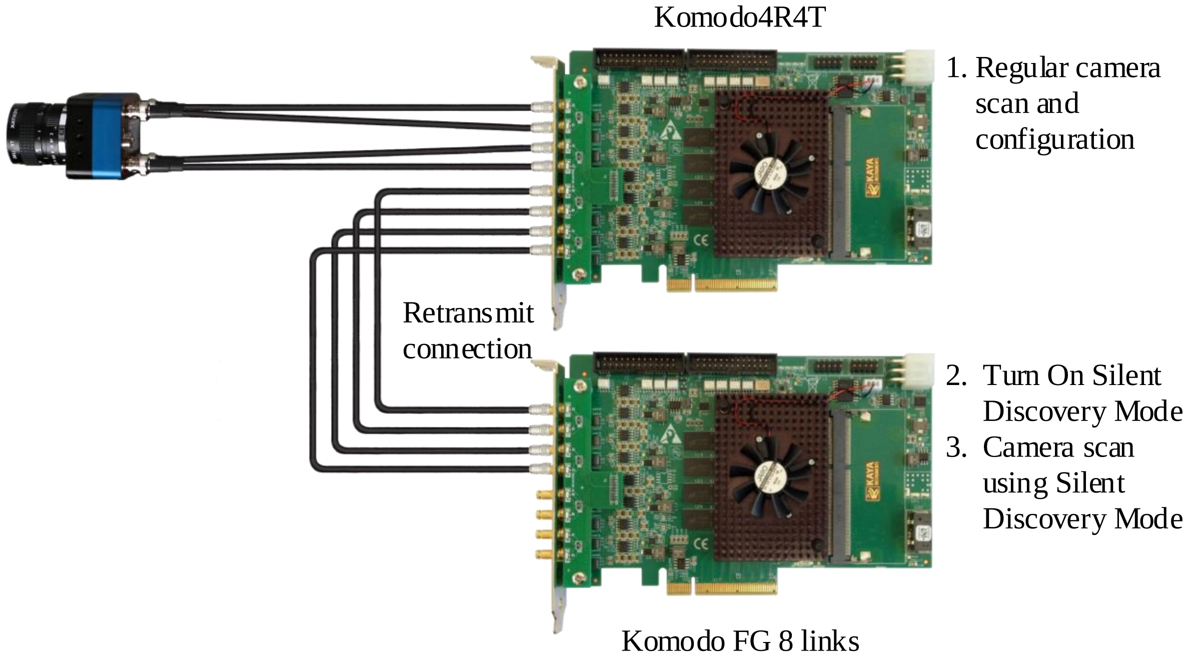

Komodo4R4T
1. Regular camera scan and configuration
Retransmit connection
2. Turn On Silent Discovery Mode
3. Camera scan using Silent Discovery Mode
Komodo FG 8 links

Figure 2 : Silent camera discovery example

5.3 Segment accumulation

Configure the Frame Grabber to capture several frames/lines before an indication is received in software.

This feature is mainly used for LineScan cameras – several lines are accumulated before software receives indication on new data acquisition. This prevents the software from receiving frames too frequently thus relieving the CPU operation.

To configure number of frames to be accumulated, the “SegmentsPerBuffer” parameter should be set for Frame Grabber after a Camera has already been connected and opened.

By default, the “SegmentsPerBuffer” parameter value is 1 which means that software indication will occur on every frame/line captured.

To modify and achieve the mentioned functionality the following steps should be taken:

1. Scan and connect to a chosen Frame Grabber.
2. Scan and connect to a chosen Camera.
3. Due to the fact that the feature is configurable per camera, the “CameraSelector” value should be set using the KYFG_SetGrabberValue() function or one of the provided sub-functions. This will choose the specific camera for which to set the “SegmentsPerBuffer” value.
4. Set the “SegmentsPerBuffer” value using the KYFG_SetGrabberValue() function or one of the provided sub-functions. Please see “KAYA’s_FG_Programming_Start-up_Guide” document for more details.

6.1 KYFG_Scan()

Scans for KAYA devices currently connected to the PC PCIe slots and available virtual devices and optionally fills array with devise IDs.

Note: The software stack requires “KYService” to be running, otherwise KYFG_Scan() will return 0.

```
int  KYFG_Scan(
          unsigned int *pids_info,
          int count) ;
```

Parameter name	Type	Description
pids_info	unsigned int*	Pointer to <pid> array of scanned devices. ***Please see remarks!***
count	int	Number of devices to assign to pids_info array (assume pids_info array is valid)

Return value

Returns the number of connected hardware and virtual devices.

If pids_info is not NULL the pointed array is filled with each Device Product ID (pid).

Remarks

If pids_info parameter is called with NULL, pids_info array will not be filled and the function will only return number of connected and virtual Frame Grabbers.

Example code

```
unsigned int * info = NULL;
unsigned int infosize = 0;

infosize = KYFG_Scan(NULL, 0); // Returns number of found devices
info = (unsigned int *) malloc ( sizeof(unsigned int) * infosize );
if(info != NULL)
{
    KYFG_Scan(info, infosize); // Fills *‘info’ with IDs of devices
}
```

6.2 KYFG_Open()

Connect to a specific Frame Grabber and initializes all required components.

Note: The software stack requires “KYService” to be running, otherwise KYFG_Open() will return INVALID_FGHANDLE.

```
FGHANDLE KYFG_Open(
        int index);
```

Parameter name	Type	Description
index	int	The index, from scan result array acquired with KYFG_Scan() function, of the Frame Grabber device to open. ***Please see remarks!***

Return value

Returns an API handle to Frame Grabber device. INVALID_FGHANDLE will indicate a wrong, impossible or unsupported connection.

Remarks

When calling the function with index of -1, a connection to the first found Frame Grabber will be established, such function call eliminates the need for KYFG_Scan() function call.

6.3 KYFG_OpenEx()

Connect to a specific Frame Grabber and initializes all required components. Project file may be passed here in order to initialize Frame Grabber and Camera parameters with previously saved values.

Note: The software stack requires “KYService” to be running, otherwise KYFG_OpenEx() will return INVALID_FGHANDLE.

```
FGHANDLE KYFG_OpenEx(
        int index,
        const char* projectFile);
```

Parameter name	Type	Description
index	int	The index, from scan result array acquired with KYFG_Scan() function, of the Frame Grabber device to open. ***Please see remarks! (1)***
projectFile	const char*	(optional) Full path of a project file with saved values. Input value can be NULL. ***Please see***

		remarks! (2)

Return value

Returns an API handle to Frame Grabber device. INVALID_FGHANDLE will indicate a wrong, impossible or unsupported connection.

Remarks

1. When calling the function with index of -1, a connection to the first found Frame Grabber will be established, such function call eliminates the need for KYFG_Scan() function call.
2. A project file with previously saved values can be passed in order to initialize camera parameters. For additional information regarding project file please refer to Vision Point application user guide: “Vision_Point_App_User_Guide”.

6.4 KYFG_SetGrabberConfigurationParameterCallback() (C++ only)

Registers a parameter callback function. This function will be called during execution of KYFG_GetGrabberConfigurationParameterDefinitions() with pointer to NodeDescriptor. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
FGSTATUS KYFG_SetGrabberConfigurationParameterCallback (
        FGHANDLE handle,
        ParameterCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	ParameterCallback	Pointer to callback function.
userContext	void*	(optional) Pointer to user context. Afterwards this pointer is retrieved when the callback is issued.

Return value

FGSTATUS - Status and error report.

6.5 KYFG_GetGrabberConfigurationParameterDefinitions() (C++ only)

Iterates over all available grabber parameters and for each parameter invokes callback function that was previously set with KYFG_SetGrabberConfigurationParameterCallback() call.

```
FGSTATUS KYFG_GetGrabberConfigurationParameterDefinitions (FGHANDLE handle);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber

Return value

FGSTATUS - Status and error report.

Example code

Example of receiving a callback for a new acquired frame and copying it to local buffer.

```
FGHANDLE grabberHandle = 0;

void NewParameter(const NodeDescriptor &nodeDescriptor, int grouppingLevel)
{
        if (nodeDescriptor.interfaceType == ParameterInterfaceType::intfICategory
                &&
                0 == grouppingLevel)
        {
                // ignore root node
                return;
        }
        if (nodeDescriptor.interfaceType != ParameterInterfaceType::intfIEnumEntry)
        {
                cout << "Parameter '" << nodeDescriptor.paramName << "': "
                        << "type - " << (int)nodeDescriptor.interfaceType
                        << endl;
        }
        else
        {
                cout << "    "; // just visual indentation for enum entries
                cout << "Enumeration entry '" << nodeDescriptor.paramName << "': "
                        << "value - " << (int)nodeDescriptor.curIntValue
                        << endl;
        }
}

void KYFG_CALLCONV ParameterCallbackImpl(void* userContext, NodeDescriptor* pNodeDescriptor, int
grouppingLevel)
{
  if(nullptr == pNodeDescriptor)
  {
    cout << "Received request from grabber to refresh all camera parameter values" << endl;
    return;
  }

  switch (pNodeDescriptor->descriptorType)
  {
  case NodeDescriptorType::NewNode:
    // no break intentionaly here
  case NodeDescriptorType::NewEnumEntry:
    NewParameter(*pNodeDescriptor, grouppingLevel);
```

```
        break;

    case NodeDescriptorType::UpdateNode:
        //NewParameterValue(*pNodeDescriptor);
                //cout << "Value of parameter '" << pNodeDescriptor->paramName << "' has been changed"
<< endl;
        break;
    }
}

void PrintParameters()
{
        KYFG_GetGrabberConfigurationParameterDefinitions(grabberHandle);
}

int main(int argc, char* argv[])
{

        if ( FGSTATUS_OK != KYFG_SetGrabberConfigurationParameterCallback(grabberHandle,
                                                    ParameterCallbackImpl,
                                                    nullptr))
        {
        printf("Cannot register parameter callback for grabber\n");
        }
        else
        {
                PrintParameters();
        }
}
```

6.6 KYFG_Close()

Close Frame Grabber specified by its handle. Stops data acquisition of all opened cameras, disconnects from all connected cameras and deletes previously created buffers associated with these cameras.

```
FGSTATUS  KYFG_Close(
        FGHANDLE handle);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber

Return value

FGSTATUS - Status and error report.

6.7 KYFG_Pid2Name() (DEPRECATED)

This function is deprecated. New applications should use function KY_DeviceDisplayName().

Converts Product ID (PID) to its designated device name.

```
const char*  KYFG_Pid2Name(
          unsigned int pid);
```

Parameter name	Type	Description
pid	unsigned int	Product id

Return value

The name of Frame Grabber issued by the specified PID.

6.8 KY_DeviceDisplayName() (DEPRECATED)

This function is deprecated. New applications should use function KY_DeviceInfo() and use pInfo.szDeviceDisplayName to retrieve device name.

Retrieve device name for the specified index.

```
const char*  KY_DeviceDisplayName(
          int index);
```

Parameter name	Type	Description
index	int	Discovered device index

Return value

The name of Frame Grabber issued by the specified index.

6.9 KY_DeviceInfo()

Fills KY_DEVICE_INFO structure with info about the relevant device.

```
FGSTATUS KY_DeviceInfo(
          int index, KY_DEVICE_INFO* pInfo);
```

Parameter name	Type	Description
index	int	Discovered device index
pInfo	KY_DEVICE_INFO*	pointer to empty struct

Return value

FGSTATUS - Status and error report.

```
typedef struct _KY_DEVICE_INFO
{
            char     szDeviceDisplayName[256];
            int      nBus;
            int      nSlot;
            int      nFunction;
            uint32_t DevicePID;
            KYBOOL   isVirtual;
}KY_DEVICE_INFO;
```

7.1 KYFG_CameraScan()

The Frame Grabber scans for connected cameras, establishes connection and defines the default speed for each camera, on every connected channel.

```
FGSTATUS KYFG_CameraScan(
        FGHANDLE handle,
        CAMHANDLE * camHandleArray,
        int *detectedCameras);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
camHandleArray	CAMHANDLE*	Array of API camera handles of detected cameras
detectedCameras	int*	Number of detected cameras

Return value

FGSTATUS - Status and error report.

FGSTATUS_EXCEEDED_MAX_CAMERA_CONNECTIONS - value will indicate that number of connected cameras exceeds the maximum allowed connected cameras.

Example code

```
        CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
        int detectedCamerasNum = 0;
        …
        KYFG_CameraScan(fgHandle ,CamHandleArray, &detectedCamerasNum);
        printf(“Found %d cameras connected to Frame Grabber”, detectedCamerasNum);
        …
```

7.2 KYFG_UpdateCameraList()

The Frame Grabber updates list of cameras connected to the device. Currently open camera handles are not affected by this function and retained at the same places of array where they were returned by previous call except for camera(s) that were closed between calls.

```
FGSTATUS KYFG_UpdateCameraList(
        FGHANDLE handle,
        CAMHANDLE *pCamHandleArray,
        int *pArraySize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
pCamHandleArray	CAMHANDLE*	Pointer to array of CAMHANDLE elements
pArraySize	int*	Pointer to integer. Must be set to number of elements allocated in the 'pCamHandleArray'. After successful function return indicates number of elements that were filled

Return value

FGSTATUS - Status and error report.

7.3 KYFG_CameraOpen2()

Opens a connection to chosen camera, retrieves native XML file or uses external XML file provided to override the native one.

```
FGSTATUS KYFG_CameraOpen2(
        CAMHANDLE camHandle,
        const char *xml_file_path);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
xml_file_path	const char*	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

An XML file can be loaded to override the native XML of the camera. Otherwise NULL should be passed in order to retrieve camera’s native XML file.

7.4 KYFG_CameraOpen() (DEPRECATED)

This function is deprecated. New applications should use function KYFG_CameraOpen2()

Opens a connection to chosen camera, retrieves native XML file or uses external XML file provided to override the native one. Project file can also be passed here in order to initialize camera parameters with previously saved values.

```
FGSTATUS KYFG_CameraOpen(
        CAMHANDLE camHandle,
        const char *xml_file_path,
        const char *project_file_path);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
xml_file_path	const char*	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. ***Please see remarks! (1)***
project_file_path	const char*	(optional) Path to previously saved values file. Value can be NULL. ***Please see remarks! (2)***

Return value

FGSTATUS - Status and error report.

Remarks

1. An XML file can be loaded to override the native XML of the camera. Otherwise NULL should be passed in order to retrieve camera's native XML file.
2. A project file with previously saved values can be passed in order to initialize camera parameters. For additional information regarding project file please refer to Vision Point application user guide: "Vision_Point_App_User_Guide".

7.5 KYFG_CameraClose()

Close a connection to the selected camera. Stops data acquisition and deletes previously created buffers associated with the camera. The connection information is preserved, so a new connection can be established later.

```
FGSTATUS KYFG_CameraClose(
          CAMHANDLE camHandle);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera

Return value

FGSTATUS - Status and error report.

7.6 KYFG_CameraInfo()

Retrieves current information about the chosen camera. The camera info includes general device information and connectivity topology. This function can be called before KYFG_CameraOpen2().

```
FGSTATUS KYFG_CameraInfo(
          CAMHANDLE camHandle,
          KYFGCAMERA_INFO *info);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
info	KYFGCAMERA_INFO*	Pointer to chosen camera information

Return value

FGSTATUS - Status and error report.

Example code

```
          CAMHANDLE CamHandleArray[KY_MAX_CAMERAS] = {0};
          KYFGCAMERA_INFO cameraInfo;
          …
          KYFG_CameraInfo(CamHandleArray[0], &cameraInfo);
          printf(“Camera model: %s, its manufacturer is %s”,
                    cameraInfo.deviceModelName, cameraInfo.deviceVendorName);
          …
```

7.7 KYFG_CameraGetXML()

Extracts native XML file from chosen camera and fills user allocated buffer. The size (in bytes) and file type (.xml or .zip) are also retrieved even if buffer isn’t large enough to hold all file data.

```
FGSTATUS KYFG_CameraGetXML(
          CAMHANDLE camHandle,
          char* buffer,
          KYBOOL *isZipFile,
          unsigned long long *bufferSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
buffer	char*	Pointer to user allocated buffer. ***Please see remarks!***
isZipFile	KYBOOL*	Pointer to indicator whether the camera’s XML file is in ZIP or XML format.
bufferSize	[in,out] unsigned long long*	Pointer to size value of the file to be extracted. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

FGSTATUS_BUFFER_TOO_SMALL – value will indicate that provided buffer size is too small to hold the file to be extracted.

Remarks

1. bufferSize [in] value will determine the size of the provided buffer. bufferSize [out] will hold the actual size of the file to extract
2. If bufferSize value is smaller than the actual needed size, or buffer value is NULL, the buffer will not be filled at all. Nevertheless bufferSize and isZipFile will be returned as expected.
3. bufferSize should reflect the actual size of the provided buffer otherwise it might cause a severe crash.

Example code

```
CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
char* buffer;
unsigned long long bufferSize = 0;
KYBOOL isZip = KYFALSE;
FILE* fileOut = NULL;
…
// Get the size of the buffer to allocate.
bufferSize = 0;
if(FGSTATUS_BUFFER_TOO_SMALL== KYFG_CameraGetXML( CamHandleArray[0],
                                                        NULL, &isZip, &bufferSize))
{
        buffer = (char*)malloc(bufferSize);          // allocate memory for buffer
        // extract camera’s native XML file
        if(FGSTATUS_OK == KYFG_CameraGetXML( CamHandleArray[0], buffer,
                                                &isZip, &bufferSize))
        {
                if(KYTRUE == isZip)
                        fileOut = fopen("camera_xml.zip","wb");    // camera XML file in zip format
                else
                        fileOut = fopen("camera_xml.xml","wb");    // camera XML file in xml format

                if (NULL != fileOut)
                {
                        fwrite(buffer, bufferSize, 1, fileOut);
                        fclose(fileOut);
```

```
                    }
          }
          free(buffer);                                             // free buffer after use
}
```

7.8 KYFG_GetXML() (DEPRECATED)

This function is deprecated. New applications should use function KYFG_CameraGetXML().

Extracts a native XML file from chosen camera and stores it into buffer. The size (in bytes) and file type (.xml or .zip) are also retrieved.

```
FGSTATUS KYFG_GetXML(
          CAMHANDLE camHandle,
          char** buffer,
          unsigned long long *bufferSize,
          KYBOOL *isZipFile);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
buffer	char**	Pointer to pointer of buffer that will hold the camera's native XML file. ***Please see remarks!***
bufferSize	unsigned long long*	Pointer that will return the allocated buffer size.
isZipFile	KYBOOL*	Pointer to indicator whether the camera's XML file is in ZIP or XML format.

Return value

FGSTATUS - Status and error report.

Remarks

This function allocates memory to hold the content of camera native XML file, therefore caller is responsible for releasing this memory.

Known issues:

If used in environment other than Visual Studio 2012, there could be a runtime library conflict issue. The pointer might become corrupted and free() function might cause a crash. To avoid this issue please use KYFG_CameraGetXML().

Example code

```
CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
          char* buffer;
```

```
        unsigned long long bufferSize = 0;
        KYBOOL isZip = KYFALSE;
        FILE* fileOut = NULL;
        …
        // extract camera’s native XML file
        If(FGSTATUS_OK == KYFG_GetXML(CamHandleArray[0], &buffer, &bufferSize,
&isZip))
        {
                if(KYTRUE == isZip)
                        fileOut = fopen("camera_xml.zip","wb");         // camera XML file in zip format
                else
                        fileOut = fopen("camera_xml.xml","wb");         // camera XML file in xml format

                if (NULL != fileOut)
                {
                        fwrite(buffer, bufferSize, 1, fileOut);
                        fclose(fileOut);
                }

                free(buffer);                                           // free buffer after use
        }
```

7.9 KYFG_SetCameraConfigurationParameterCallback() (C++ only)

Registers a parameter callback function. This function will be called during execution of KYFG_GetCameraConfigurationParameterDefinitions() with pointer to NodeDescriptor. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
FGSTATUS KYFG_SetCameraConfigurationParameterCallback (
        CAMHANDLE handle,
        ParameterCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen Camera
userFunc	ParameterCallback	Pointer to callback function.
userContext	void*	(optional) Pointer to user context. Afterwards this pointer is retrieved when the callback is issued.

Return value

FGSTATUS - Status and error report.

7.10 KYFG_GetCameraConfigurationParameterDefinitions() (C++ only)

Iterates over all available camera parameters and for each parameter invokes callback function that was previously set with KYFG_SetCameraConfigurationParameterCallback() call.

FGSTATUS KYFG_GetCameraConfigurationParameterDefinitions (CAMHANDLE camHandle);

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen Camera

Return value

FGSTATUS - Status and error report.

Example

See example code of KYFG_GetGrabberConfigurationParameterDefinitions().

8 Callback Functions

8.1 KYFG_CallbackRegister()(DEPRECATED)

This function is deprecated. New applications should use functions KYFG_CameraCallbackRegister() or KYFG_StreamBufferCallbackRegister().

Register a general runtime acquisition callback function. The callback (userFunc) will be called upon each new received frame of a valid stream, with appropriate BUFFHANDLE. Callback call is not necessarily serialized, which means different streams might generate concurrent calls before end of previous callback execution.

Use the Buffer Interface functions to handle received data. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
FGSTATUS KYFG_CallbackRegister(
        FGHANDLE handle,
        FGCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	FGCallback	Pointer to callback function
userContext	void*	(optional) Pointer to user context. Afterwards this pointer is retrieved when the callback is issued. Helps to determine the origin of stream in host application.

Return value

FGSTATUS - Status and error report.

8.2 KYFG_CallbackUnregister()(DEPRECATED)

This function is deprecated. New applications should use functions KYFG_CameraCallbackUnregister() or KYFG_StreamBufferCallbackUnregister()

Unregisters a previously registered general runtime acquisition callback function.

```
FGSTATUS KYFG_CallbackUnregister(
        FGHANDLE handle,
        FGCallback userFunc);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber

userFunc	FGCallback	Callback function prototype

Return value

FGSTATUS - Status and error report.

8.3 KYFG_CameraCallbackRegister()

Register a camera runtime acquisition callback function. The callback (userFunc) will be called upon new received frame, of a valid stream from specific camera, with appropriate STREAM_HANDLE. Each camera's callback is serialized and will be held until end of callback execution. The different camera callbacks are working concurrently. Use the Stream interface functions to handle received data. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
FGSTATUS KYFG_CameraCallbackRegister(
        CAMHANDLE camHandle,
        CameraCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
userFunc	CameraCallback	Pointer to callback function
userContext	void*	(optional) Pointer to user context. Afterwards this pointer is retrieved when the callback is issued. Helps to determine the origin of stream in host application.

Return value

FGSTATUS - Status and error report.

8.4 KYFG_CameraCallbackUnregister()

Unregisters a previously registered camera runtime acqusition callback function.

```
FGSTATUS KYFG_CallbackUnregister(
        CAMHANDLE camHandle,
        CameraCallback userFunc);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
userFunc	CameraCallback	Callback function prototype

Return value

FGSTATUS - Status and error report.

Example code

Example of receiving a callback for a new acquired frame and copying it to local buffer.

```
void Stream_callback_func(void* userContext, BUFFHANDLE buffHandle)
{
    static void* data = NULL;
    static KYBOOL copyingDataFlag = KYFALSE;
    long long width = 0, height = 0, totalFrames = 0, buffSize = 0;
    void* buffData;

    if(0 == buffHandle)              // callback with indicator for acquisition stop
    {
        copyingDataFlag = KYFALSE;
        return;
    }

    width = KYFG_GetCameraValueInt(buffHandle, “Width”);
    height = KYFG_GetCameraValueInt(buffHandle, “Height”);
    totalFrames = KYFG_GetGrabberValueInt(buffHandle, "RXFrameCounter");
    buffSize = KYFG_BufferGetSize(buffHandle);              // get buffer size
    buffIndex = KYFG_BufferGetFrameIndex(buffHandle);
    buffData = KYFG_BufferGetPtr(buffHandle, buffIndex);// get pointer of buffer data

    if(KYFALSE == copyingDataFlag)
    {
        copyingDataFlag = KYTRUE;
        data = (void*)realloc(data, buffSize);                  // allocate size for local buffer
        if (NULL == data)
        {
            return;
        }
        printf(“Callback of buffer %X, width: %d, height: %d, total frames acquired: %d”,
                                                   buffHandle, width, height, totalFrames);
        memcpy(data, buffData, buffSize);                              // copy data to local buffer
        //... Show Image with data …
        copyingDataFlag = KYFALSE;
    }
}
```

```
    int main(int argc, char* argv[])
    {
        FGHANDLE handle;
        CAMHANDLE CamHandleArray[4] = {0};
        int nDetectedCameras = 0;
         …
        KYFG_CameraScan(handle, CamHandleArray, &nDetectedCameras);
        if ( nDetectedCameras > 0 )
        {
            KYFG_CameraCallbackRegister(CamHandleArray[0], Stream_callback_func, NULL);
        }
         …
        while(1){}
        return 0;
    }
```

8.5 KYFG_StreamBufferCallbackRegister()

Register a stream runtime acquisition callback function. The callback (userFunc) will be called upon new received frame, of a valid stream, with appropriate STREAM_BUFFER_HANDLE. Each stream's callback is serialized and will be held until end of callback execution. The different stream callbacks are working concurrently. Use the Stream interface functions to handle received data. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
FGSTATUS KYFG_StreamBufferCallbackRegister (
        STREAM_HANDLE streamHandle,
        StreamBufferCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream
userFunc	StreamBufferCallback	Pointer to callback function
userContext	void*	Pointer to user context. Afterwards this pointer is retrieved when the callback is issued. Helps to determine the origin of stream in host application.

Return value

FGSTATUS - Status and error report.

8.6 KYFG_StreamBufferCallbackUnregister()

Unregisters a previously registered stream callback function.

```
FGSTATUS KYFG_StreamBufferCallbackUnregister(
        STREAM_HANDLE streamHandle,
        StreamBufferCallback userFunc);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream
userFunc	StreamBufferCallback	Pointer to callback function

Return value

FGSTATUS - Status and error report.

8.7 KYFG_AuxDataCallbackRegister()

Register run-time callback for receiving auxiliary data. The callback will be called when various auxiliary data is generated.

```
FGSTATUS KYFG_AuxDataCallbackRegister(
        FGHANDLE handle,
        FGAuxDataCallback userFunc,
        void* userContext);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device ***Please see remarks at the beginning of this section!***
userFunc	FGAuxDataCallback	Pointer to callback function implementation.
userContext	void*	Pointer to user context. This pointer will be passed the callback function. Helps to determine the origin of function call in host application

Return value

FGSTATUS - Status and error report.

8.8 KYFG_AuxDataCallbackUnregister()

Unregister run-time auxiliary data callback.

```
FGSTATUS KYFG_AuxDataCallbackUnregister(
        FGHANDLE handle,
        FGAuxDataCallback userFunc);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this section!***
userFunc	FGAuxDataCallback	Callback function prototype. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

Due to the fact that several Auxiliary data retrieval functions may be registered, userFunc parameter should be passed to the un-registering function to determine which specific function to un-register.

8.9 KYDeviceEventCallBackRegister()

Register a generic runtime callback function. The callback (userFunc) will be called to inform user application about various events in the system. See KYDEVICE_EVENT for more details.

```
FGSTATUS KYDeviceEventCallBackRegister (
        FGHANDLE handle,
        KYDeviceEventCallBack userFunc,
        void* userContext);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle that represents PCI device
userFunc	KYDeviceEventCallBack	Pointer to callback function implementation.
userContext	void*	Pointer to user context. This pointer is passed to the user's callback function as first parameter.

Return value

FGSTATUS - Status and error report.

8.10 KYDeviceEventCallBackUnregister()

Unregisters a previously registered camera simulation runtime callback function.

```
FGSTATUS KYDeviceEventCallBackUnregister (
        FGHANDLE handle,
        FGAuxDataCallback userFunc);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle that represents PCI device

userFunc	KYDeviceEventCallBack	Pointer to callback function. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

Due to the fact that several callback functions may be registered, userFunc parameter should be passed to the un-registering function to determine which specific function to un-register.

NOTE: Any of the callback function described in this section should perform a minimal necessary tasks and return as soon as possible, avoiding a long running I/O operations. For example, KYFG_CameraWriteReg is an I/O operation that involves signal round-trip to camera and waiting for camera's acknowledge. It is advised to move the operations to a separated thread using function, such as KYFG_CameraWriteReg, for a direct write data buffer to the selected camera.

Remarks

1. KYFG_SetGrabberValue() / KYFG_GetGrabberValue() and all of their sub functions are used to handle both general Frame Grabber configurations (e.g IO configurations), and camera stream specific parameters (e.g camera stream RX packets). For setting/getting camera stream specific parameters, the CameraSelector should be first chosen. Please refer to "KAYAs_FG_Programming_Start-up_Guide" document for full parameters list and examples.
2. KYFG_SetCameraValue() / KYFG_GetCameraValue() and all of their sub functions are used to handle connected camera parameters. These are extracted from internal or external camera xml file.

9.1 KYFG_SetCameraValue() / KYFG_SetGrabberValue()

Set camera/Frame Grabber configuration field value. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValue(
        CAMHANDLE camHandle,
        const char *paramName,
        void *paramValue);
```

```
FGSTATUS  KYFG_SetGrabberValue(
        FGHANDLE handle,
        const char *paramName,
        void *paramValue);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this section!***
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value

FGSTATUS - Status and error report.

9.1.1 KYFG_SetCameraValueInt() / KYFG_SetGrabberValueInt()

Set camera/Frame Grabber configuration field value of Integer type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueInt(
          CAMHANDLE camHandle,
          const char *paramName,
          long long value);
```

```
FGSTATUS  KYFG_SetGrabberValueInt(
          FGHANDLE handle,
          const char *paramName,
          long long value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
value	long long	Value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

9.1.2 KYFG_SetCameraValueFloat() / KYFG_SetGrabberValueFloat()

Set camera/Frame Grabber configuration field value of Float type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueFloat(
          CAMHANDLE camHandle,
          const char *paramName,
          double value);
```

```
FGSTATUS  KYFG_SetGrabberValueFloat(
          FGHANDLE handle,
          const char *paramName,
          double value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
value	double	Floating point value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

9.1.3 KYFG_SetCameraValueBool() / KYFG_SetGrabberValueBool()

Set camera/Frame Grabber configuration field value of Boolean type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueBool(
          CAMHANDLE camHandle,
          const char *paramName,
          KYBOOL value);
```

```
FGSTATUS  KYFG_SetGrabberValueBool(
          FGHANDLE handle,
          const char *paramName,
          KYBOOL value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
value	KYBOOL	Boolean value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

9.1.4 KYFG_SetCameraValueEnum() / KYFG_SetGrabberValueEnum()

Set camera/Frame Grabber configuration field value of Enumeration type by their numeric value. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueEnum(
          CAMHANDLE camHandle,
          const char *paramName,
          long long value);
```

```
FGSTATUS  KYFG_SetGrabberValueEnum(
          FGHANDLE handle,
          const char *paramName,
          long long value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
value	long long	Enumeration value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

Example code

The following example shows how to set the pixel format to mono 8bit (numeric value of 0x101 according to Gen<i>Cam standard).

```
          CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
          …
          long long pixel_format_value = 0x101;                    // mono 8bit format
          KYFG_SetCameraValueEnum(CamHandleArray[0], “PixelFormat”,  pixel_format_value);
          …
```

9.1.5 KYFG_ExecuteCommand()/KYFG_ExecuteGrabberCommand() (DEPRECATED)

This function is deprecated. New applications should use KYFG_CameraExecuteCommand()/ KYFG_GrabberExecuteCommand()

Execute camera/Frame Grabber command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_ExecuteCommand(
          CAMHANDLE camHandle,
          const char *paramName);
```

```
FGSTATUS KYFG_ExecuteGrabberCommand (
          FGHANDLE handle,
          const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

FGSTATUS - Status and error report.

9.1.6 KYFG_CameraExecuteCommand() / KYFG_GrabberExecuteCommand()

Execute camera/Frame Grabber command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_CameraExecuteCommand(
          CAMHANDLE camHandle,
          const char *paramName);
```

```
FGSTATUS KYFG_GrabberExecuteCommand (
          FGHANDLE handle,
          const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

FGSTATUS - Status and error report.

9.1.7 KYFG_SetCameraValueString() / KYFG_SetGrabberValueString()

Set camera/Frame Grabber configuration field value of String type. According to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueString(
          CAMHANDLE camHandle,
          const char *paramName,
          const char* value);
```

```
FGSTATUS  KYFG_SetGrabberValueString(
        FGHANDLE handle,
        const char *paramName,
        const char* value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
value	const char*	String value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

9.1.8 KYFG_SetCameraValueEnum_ByValueName() / KYFG_SetGrabberValueEnum_ByValueName()

Set camera/Frame Grabber configuration enumeration field by field name and enumeration name, according to Gen<i>Cam standard naming and xml field definition and type.

```
FGSTATUS  KYFG_SetCameraValueEnum_ByValueName(
        CAMHANDLE camHandle,
        const char *paramName,
        const char *paramValueName);
```

```
FGSTATUS  KYFG_SetGrabberValueEnum_ByValueName(
        FGHANDLE handle,
        const char *paramName,
        const char *paramValueName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
paramValueName	const char*	Name of parameter enumeration choice

Return value

FGSTATUS - Status and error report.

Example code

This example demonstrates how to set the Gen<i>Cam enumeration field named “AcquisitionMode” to one of its enumeration options “Continuous”.

```
        CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected

        KYFG_SetCameraValueEnum_ByValueName(CamHandleArray[0],
                                                                    “AcquisitionMode”,
                                                                    “Continuous”);
```

9.2 KYFG_GetCameraValueType() / KYFG_GetGrabberValueType()

Get the camera/Frame Grabber configuration field type.

```
KY_CAM_PROPERTY_TYPE  KYFG_GetCameraValueType(
          CAMHANDLE camHandle,
          const char *paramName);
```

```
KY_CAM_PROPERTY_TYPE  KYFG_GetGrabberValueType(
          FGHANDLE handle,
          const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

Type of camera parameter as described in KY_CAM_PROPERTY_TYPE enumeration.

9.3 KYFG_GetCameraValue() / KYFG_GetGrabberValue()

Get camera/Frame Grabber configuration field value.

```
FGSTATUS  KYFG_GetCameraValue(
          CAMHANDLE camHandle,
          const char *paramName,
          void *paramValue);
```

```
FGSTATUS  KYFG_GetGrabberValue(
          FGHANDLE handle,
          const char *paramName,
          void *paramValue);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value

FGSTATUS - Status and error report.

9.3.1 KYFG_GetCameraValueInt() / KYFG_GetGrabberValueInt()

Get camera/Frame Grabber configuration value of Integer type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
long long KYFG_GetCameraValueInt(
          CAMHANDLE camHandle,
          const char *paramName);
```

```
long long KYFG_GetGrabberValueInt(
          FGHANDLE handle,
          const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this section!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

Integer value of camera configuration field of integer type. In case of an error INT_MAX will be returned.

9.3.2 KYFG_GetCameraValueEnum() / KYFG_GetGrabberValueEnum()

Get camera/Frame Grabber configuration value of Enumeration type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
long long KYFG_GetCameraValueEnum(
        CAMHANDLE camHandle,
        const char *paramName);
```

```
long long KYFG_GetGrabberValueEnum(
        FGHANDLE handle,
        const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

Integer value of camera configuration field of enumeration type. In case of an error INT_MAX will be returned.

Example code

```
        CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
        long long linkconfig = 0;
        …
        linkconfig = KYFG_GetCameraValueEnum (CamHandleArray[0], “LinkConfig”);
        …
```

9.3.3 KYFG_GetCameraValueFloat() / KYFG_GetGrabberValueFloat()

Get camera/Frame Grabber configuration value of Float type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
double  KYFG_GetCameraValueFloat(
        CAMHANDLE camHandle,
        const char *paramName);
```

```
double  KYFG_GetGrabberValueFloat(
        FGHANDLE handle,
        const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

Floating point value of camera configuration field of Float type. In case of an error MAX_FLOAT_VALUE will be returned.

static const int MAX_FLOAT_VALUE = INT_MAX;   // float value in case of error

9.3.4 KYFG_GetCameraValueBool() / KYFG_GetGrabberValueBool()

Get camera/Frame Grabber configuration value of Boolean type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
KYBOOL  KYFG_GetCameraValueBool(
        CAMHANDLE camHandle,
        const char *paramName);
```

```
KYBOOL  KYFG_GetGrabberValueBool(
        FGHANDLE handle,
        const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter

Return value

KYBOOL - Boolean value of camera configuration field of Boolean type.

9.3.5 KYFG_GetCameraValueStringCopy()/KYFG_GetGrabberValueStringCopy()

Get camera/Frame Grabber configuration value of String type field. Value is copied to user allocated char array. ***Please see remarks!***

```
FGSTATUS KYFG_GetCameraValueStringCopy(
        CAMHANDLE camHandle,
        const char *paramName,
        char *stringPtr,
        unsigned int *stringSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
stringPtr	char*	Pointer user char array that will be filled with value of chosen parameter
stringSize [in,out]	unsigned int*	Pointer to size value of the file to be extracted. ***Please see remarks!***

Return value

At function return, stringSize will hold the desired string length including NULL termination character.

FGSTATUS - Status and error report.

FGSTATUS_BUFFER_TOO_SMALL – value will indicate that provided buffer size is too small to hold the requested string value.

Remarks

1. stringSize [in] value will determine the size of the provided char array. stringSize [out] will hold the actual size of the string to extract
2. If stringSize value is smaller than the actual needed size, or stringPtr value is NULL, the char array will not be filled at all. Nevertheless stringSize will be returned as expected.
3. stringSize should reflect the actual size of the provided char array otherwise it might cause a severe crash.

Example code

```
CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
char* stringValue = NULL;
unsigned int stringSize = 0;
…
if (FGSTATUS_BUFFER_TOO_SMALL == KYFG_GetCameraValueStringCopy(
                                        CamHandleArray[0], "DeviceVendorName", NULL, &stringSize))
{
        stringValue = (char*)malloc(stringSize);                // allocate memory for buffer
        if(FGSTATUS_OK == KYFG_GetCameraValueStringCopy(
                                      CamHandleArray[0], "DeviceVendorName", stringValue,
&stringSize))
        {
                  printf("Camera's vendor name is: %s", stringValue);
        }
        free(stringValue);
}
```

9.3.6 KYFG_GetCameraValueString() / KYFG_GetGrabberValueString()

Get camera/Frame Grabber configuration value of String type field. ***Please see remarks!***

```
FGSTATUS KYFG_GetCameraValueString(
          CAMHANDLE camHandle,
          const char *paramName,
          char** ptr);
```

```
FGSTATUS KYFG_GetGrabberValueString(
          FGHANDLE handle,
          const char *paramName,
          char** ptr);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera ***Please see remarks at the beginning of this chapter!***
handle	FGHANDLE	API handle to chosen Frame Grabber ***Please see remarks at the beginning of this chapter!***
paramName	const char*	Name of configuration parameter
ptr	char**	Pointer to pointer of string value for chosen parameter

Return value

FGSTATUS - Status and error report.

Remarks

This function allocates memory for char array and caller is responsible for releasing this memory using free() function.

Known issues:

If used in environment other than Visual Studio 2012, there could be a runtime library conflict issue. The pointer might become corrupted and free() function might cause a crash. To avoid this issue please use KYFG_GetCameraValueStringCopy() .

Example code

```
CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
char* stringValue;
…
if (FGSTATUS_OK == KYFG_GetCameraValueString(CamHandleArray[0],
                                              "DeviceVendorName",
                                              &stringValue))
{
    printf("Camera's vendor name is: %s", stringValue);
    free(stringValue);
}
```

Authentication API is used to authenticate Frame Grabber device. Use of this API is subject to firmware support. Programing grabber with a lock value set to 1 is an irreversible operation, and result that the grabber couldn't be reprogrammed.

10.1 KY_AuthProgramKey()

Program provided key to the grabber.

```
FGSTATUS KY_AuthProgramKey (
        FGHANDLE handle,
        KY_AuthKey* pKey,
        int lock);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to a Frame Grabber
pKey	KY_AuthKey *	Pointer to KY_AuthKey structure containing information to be programmed into Frame Grabber
lock	int	If this parameter is 0 the grabber can be re-programmed with a different key later. If this parameter is 1 then provided key is locked in the Frame Grabber and following call of this function will fail.

Return value

FGSTATUS - Status and error report.

10.2 KY_AuthVerify()

Verify provided key against one already programmed to the grabber.

```
FGSTATUS KY_AuthVerify (
        FGHANDLE handle,
        KY_AuthKey* pKey);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to a Frame Grabber
pKey	KY_AuthKey *	Pointer to KY_AuthKey structure to be verified with Frame Grabber

Return value

FGSTATUS - Status and error report.

Example code

```
KY_AuthKey key;

… // Fill key with desired content and program it into grabber
fgStatus = KY_AuthProgramKey(handle, &key, 0);
if (FGSTATUS_OK == fgStatus)
{
        printf("KY_AuthProgramKey succeeded\n");
}
else
{
        printf("KY_AuthProgramKey failed with status %0X\n", fgStatus);
}

….

// Verify saved key with grabber
fgStatus = KY_AuthVerify(handle, &key);
if (FGSTATUS_OK == fgStatus)
{
        printf("KY_AuthVerify succeeded\n");
}
else
{
        printf("KY_AuthVerify failed with status %0X\n", fgStatus);
}
```

11 Buffer interface (DEPRECATED)

The API described in this section is deprecated. New applications should use API described in the next chapter "Stream Interface"

11.1 KYFG_BufferAlloc() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamCreateAndAlloc()

A new buffer will be allocated for the chosen camera. The buffer will hold the data of acquired frames. Buffer acquisition mechanism and buffer size calculations are handled internally. Buffer frame size is calculated with consideration of specified number of frames, in addition to camera and grabber configuration parameters set previously to this function call. Changing certain camera/grabber parameters, after successfully buffer allocation, might result in unstable software operation, memory leaks and even total system crash.

```
FGSTATUS KYFG_BufferAlloc(
        CAMHANDLE camHandle,
        BUFFHANDLE *buffHandle ,
        uint32_t frames);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
buffHandle	BUFFHANDLE*	Pointer to API handle to data buffer for selected camera. ***Please see remarks!***
frames	uint32_t	Number of frames that should be allocated for this buffer. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

1. Multiple buffers can be allocated for each connected camera. Nevertheless, no more than 1 buffer can be active at any given moment.
2. It's advisable to allocate several frames to allow the continuity of data flow and handle by the host application. This is most important for support in case of large frame rate.

Example code

```
        CAMHANDLE CamHandleArray[4] = {0};  // maximum 4 cameras can be connected
        BUFFHANDLE buffHandle = 0;
        …
        if (FGSTATUS_OK == (KYFG_BufferAlloc(CamHandleArray[0], &buffHandle, 16))
        {
            printf("New buffer was allocated with handle %X", buffHandle);
        }
```

11.2 KYFG_BufferDelete() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamDelete()

Delete a previously allocated buffer. This will drop the buffer from its associated camera's buffer pool, therefore it will no longer be available for use with camera.

```
FGSTATUS KYFG_BufferDelete(
        BUFFHANDLE buffHandle);
```

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to data buffer for selected camera

Return value

FGSTATUS - Status and error report.

11.3 KYFG_BufferGetSize() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamGetSize()

Retrieves the size of a frame in the chosen buffer.

```
int64_t KYFG_BufferGetSize (
        BUFFHANDLE buffHandle);
```

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to data buffer for selected camera

Return value

Size of each frame in the chosen buffer. In case of an error -1 will be returned.

11.4 KYFG_BufferGetFrameIndex() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamGetFrameIndex()

Retrieves the index of the last acquired frame from chosen buffer.

```
int KYFG_BufferGetFrameIndex(
        BUFFHANDLE buffHandle);
```

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to data buffer for selected camera

Return value

Index of the last acquired frame from chosen buffer. In case of an error -1 will be returned.

11.5 KYFG_BufferGetPtr() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamGetPtr()

Retrieves a pointer to data memory space of 1 frame in the chosen buffer.

```
void* KYFG_BufferGetPtr(
        BUFFHANDLE buffHandle,
        uint32_t frame);
```

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to data buffer for selected camera
frame	uint32_t	Frame index of data pointer to be retrieved

Return value

Pointer to data buffer according to selected frame. NULL will be retrieved if frame index is out of range or other operation failure.

Example code

```
        BUFFHANDLE buffHandle = 0;
        int buffIndex = 0;
        void* buffData = NULL;

        if( -1 != (buffIndex =  KYFG_StreamGetFrameIndex(buffHandle))  ){
                buffData = KYFG_BufferGetPtr(buffHandle , buffIndex);
        }
```

11.6 KYFG_BufferGetAux() (DEPRECATED)

This function is deprecated. New applications should use KYFG_StreamGetAux()

Retrieves pointer to Auxiliary data of specified frame.

```
void* KYFG_BufferGetAux (
        BUFFHANDLE buffHandle,
        int frame,
        KYFG_AUX_DATA* pAuxData);
```

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to data buffer for selected camera
frame	int	Frame index of data pointer to be retrieved
pAuxData	KYFG_AUX_DATA*	Pointer to auxiliary data of specified frame

Return value

FGSTATUS - Status and error report.

Example code

```
BUFFHANDLE buffHandle = 0;
int buffIndex = 0;
KYFG_AUX_DATA auxData;

if( -1 != (buffIndex =  KYFG_StreamGetFrameIndex(buffHandle))  )
{
        KYFG_BufferGetAux(buffHandle, buffIndex, &auxData);
        printf("Auxiliary Data: {sequence_number = %u, timestamp = %lu}",
                auxData.frame_data.sequence_number, auxData.frame_data.timestamp);
}
```

12.1 Stream Interface description

Stream interface functions are used to handle received data. There are two modes in which data can be received and handled:

- **Cyclic frame buffers organization and continuous data filling:**

  In this mode, memory buffer for each frame is being filled continuously with acquired data. The frames are processed one after another, regardless of whether a frame was already read and processed by software or not.

  This mode is best for quick automatic buffer handling on the Hardware level, preventing software potential latency in data acquisition. Nevertheless, using this buffer handling mode can potentially lead to overlap in frame memory being read by application, while simultaneously being filled with new data by Hardware. To avoid application data corruption, stream callback function, implemented by user application, should process frame memory as fast as possible and return control to library.

  This mode is supported for streams created with KYFG_StreamCreateAndAlloc() function. The code sample of using this mode - using Cyclic buffers

- **Queued buffers organization:**

  In this mode, only memory buffer of frames which were placed in the *Input Queue* can be filled by Hardware. When an individual frame memory is filled, it is moved to *Output Queue* and a callback to user application is issued. This frame memory will not be affected until it is returned to *Input Queue* using KYFG_BufferToQueue() function call. User application is responsible for putting frames to *Input Queue* for each frame supplied to host application through Stream callback. If host application fails to do so then *Input Queue* will eventually became empty and new acquired data will be dropped until additional frames are moved to *Input Queue*.

  This mode is used for streams created with KYFG_StreamCreate() function. The code sample of using this mode - using Queued buffers

  To check if this mode is supported by Hardware, DEVICE_QUEUED_BUFFERS_SUPPORTED grabber parameter should be read using KYFG_GetGrabberValueInt() function call. If returned value is 1 then queued buffers mode is supported, otherwise all API functions related this mode will return the following error: FGSTATUS_QUEUED_BUFFERS_NOT_SUPPORTED.

12.2 KYFG_StreamCreateAndAlloc()

A new stream will be allocated for specified camera. The created stream buffers will hold the data of acquired frames. Stream buffer acquisition mechanism and buffer size calculations are handled internally. Buffer frame size is calculated with consideration of specified number of frames, in addition to camera and grabber configuration parameters set previously to this function call. Changing certain camera/grabber parameters, after successfully stream allocation, might result in unstable software operation, memory leaks and even total system crash.

```
FGSTATUS KYFG_StreamCreateAndAlloc (
        CAMHANDLE camHandle,
        STREAM_HANDLE *pStreamHandle,
        uint32_t frames,
        int streamIndex);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
pStreamHandle	STREAM_HANDLE*	Output parameter - pointer to STREAM_HANDLE variable that will hold handle of newly created stream
frames	uint32_t	Number of frames that should be allocated for this stream.
streamIndex	int	Index of stream. Currently unused and must be 0.

Return value

FGSTATUS - Status and error report.

Example code

```
CAMHANDLE camHandleArray[4] = {0};  // maximum 4 cameras can be connected
STREAM_HANDLE streamHandle = 0;
…
if (FGSTATUS_OK == (KYFG_StreamCreateAndAlloc (camHandleArray[0],
                                                &streamHandle,
                                                16,
                                                0)))
{
    printf("New stream was allocated with handle %X", streamHandle);
}
```

12.3 KYFG_StreamCreate()

A new stream will be created for the chosen camera. The stream will manage frame buffers allocated either by user or by library. Frame buffers will be organized in queues – input, output, automatic – and in a set of unqueued frame buffers.

```
FGSTATUS KYFG_StreamCreate(
        CAMHANDLE camHandle,
        STREAM_HANDLE * pStreamHandle,
        int streamIndex);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
pStreamHandle	STREAM_HANDLE*	Output parameter - pointer to STREAM_HANDLE variable that will hold handle of newly created stream
streamIndex	int	Index of stream. Currently unused and must be 0.

Return value

FGSTATUS - Status and error report.

12.4 KYFG_StreamGetInfo()

Retrieves information about specified stream.

```
FGSTATUS KYFG_StreamGetInfo (
        STREAM_HANDLE streamHandle,
        KY_STREAM_INFO_CMD cmdStreamInfo,
        void *pInfoBuffer,
        size_t *pInfoSize,
        KY_DATA_TYPE *pInfoType);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	Handle of a stream
cmdStreamInfo	KY_STREAM_INFO_CMD	Specifies what information is being requested. Possible values are: • KY_STREAM_INFO_PAYLOAD_SIZE – The function will return size of memory required for single frame buffer. ‘pInfoBuffer’ must be NULL or point to size_t variable. • KY_STREAM_INFO_BUF_ALIGNMENT – The function will return required alignment of memory

		allocated for a buffer. 'pInfoBuffer' must be NULL or point to size_t variable.
pInfoBuffer	void *	Pointer to user variable that will be filled with required information. Can be NULL. ***Please see remarks!***
pInfoSize	size_t *	Pointer to size of provided pInfoBuffer. Can be NULL. In: size of the provided pInfoBuffer in bytes. Out: minimum required size of pInfoBuffer to hold requested information. ***Please see remarks!***
pInfoType	KY_DATA_TYPE *	Pointer to data type of pInfoBuffer content. Can be NULL. Out: data type of pInfoBuffer for requested information. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

1. If information type is known, provide a valid buffer of the correct size to fill in requested information. In this case pInfoSize and pInfoType can be NULL.
2. Alternatively, the information request can be done in two steps:
   a. Check which size (pInfoSize) and data type (pInfoType) should pInfoBuffer represent. Provide valid pInfoSize and/or pInfoType and call function with pInfoBuffer = NULL.
   b. Provide a valid buffer to fill in the requested information. pInfoSize should be NULL or size of specified pInfoBuffer.
3. If pInfoType is a string, the size includes the termination 0.

12.5 KYFG_StreamGetSize()

Retrieves the size of the last acquired frame acquired from specified stream.

```
int64_t KYFG_StreamGetSize (
        STREAM_HANDLE buffHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value

Size of each frame of specified stream. In case of an error -1 will be returned.

12.6 KYFG_StreamGetFrameIndex()

Retrieves the index of the last acquired frame acquired from specified stream.

```
int KYFG_StreamGetFrameIndex(
        STREAM_HANDLE streamHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value

Index of the last acquired frame from specified stream. In case of an error -1 will be returned.

12.7 KYFG_StreamGetPtr()

Retrieves a pointer to data memory space of 1 frame in the chosen buffer.

```
void* KYFG_StreamGetPtr (
        STREAM_HANDLE streamHandle,
        uint32_t frame);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream
frame	uint32_t	Frame index of data pointer to be retrieved

Return value

Pointer to data of specified frame. NULL will be retrieved if frame index is out of range or other operation failure.

Example code

```
        STREAM_HANDLE streamHandle = 0;
        int frameIndex = 0;
        void* frameData = NULL;

        if( -1 != (frameIndex =  KYFG_StreamGetFrameIndex(streamHandle))  )
        {
                frameData = KYFG_StreamGetPtr (streamHandle , frameIndex);
        }
```

12.8 KYFG_StreamGetAux()

Retrieves pointer to Auxiliary data of specified frame.

```
void* KYFG_StreamGetAux (
        STREAM_HANDLE streamHandle,
        int frame,
        KYFG_AUX_DATA* pAuxData);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream
frame	int	Frame index of data pointer to be retrieved
pAuxData	KYFG_AUX_DATA*	Pointer to auxiliary data of specified frame

Return value

FGSTATUS - Status and error report.

Example code

```
        STREAM_HANDLE buffHandle = 0;
        int frameIndex = 0;
        KYFG_AUX_DATA auxData;

        if( -1 != (frameIndex =  KYFG_StreamGetFrameIndex(streamHandle))  )
        {
                KYFG_StreamGetAux(streamHandle, frameIndex, &auxData);
                printf("Auxiliary Data: {sequence_number = %u, timestamp = %lu}",
                        auxData.frame_data.sequence_number, auxData.frame_data.timestamp);
        }
```

12.9 KYFG_BufferAllocAndAnnounce()

This function is used to allocate and announce a buffer and bind it to a stream.

The memory size should correspond to a single acquisition frame. This size can be retrieved using function KYFG_StreamGetInfo() with KY_STREAM_INFO_PAYLOAD_SIZE info command.

The library is responsible for managing allocated memory and will be freed when the buffer is deleted.

Initially, the buffer will be added to the set of unqueued buffers of that stream. To make the buffer available for incoming data, user should add it to incoming queue using function

KYFG_BufferToQueue(). Alternatively, function KYFG_BufferQueueAll() can be used after all desired user buffers are announced.

```
FGSTATUS KYFG_BufferAllocAndAnnounce(
        STREAM_HANDLE streamHandle,
        size_t nBufferSize,
        void* pPrivate,
        STREAM_BUFFER_HANDLE * pBufferHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to connected camera
nBufferSize	size_t	The size of allocated memory. Currently this parameter MUST be equal to size returned by KYFG_StreamGetInfo() with info command KY_STREAM_INFO_PAYLOAD_SIZE
pPrivate	void*	This parameter is currently ignored
pBufferHandle	STREAM_BUFFER_HANDLE *	Output parameter - pointer to STREAM_BUFFER_HANDLE variable that will hold handle of newly announced frame buffer

Return value

FGSTATUS - Status and error report.

12.10 KYFG_BufferAnnounce()

This function is used to announce a buffer allocated by user and bind it to a stream.

The memory size should correspond to a single acquisition frame. This size can be retrieved using function KYFG_StreamGetInfo() with KY_STREAM_INFO_PAYLOAD_SIZE info command. Also, any virtual memory allocated by user should be aligned to the value retrieved using function KYFG_StreamGetInfo() with KY_STREAM_INFO_BUF_ALIGNMENT info command.

The user remains the owner of memory – the memory will NOT be freed by library and MUST stay valid until stream is deleted.

Initially, the buffer will be added to the set of unqueued buffers of that stream. To make the buffer available for incoming data, user should add it to incoming queue using function KYFG_BufferToQueue(). Alternatively, function KYFG_BufferQueueAll() can be used after all desired user buffers are announced.

```
FGSTATUS KYFG_BufferAnnounce(
        STREAM_HANDLE streamHandle,
        void * pBuffer,
        size_t nBufferSize,
        void* pPrivate,
        STREAM_BUFFER_HANDLE * pBufferHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to connected camera
pBuffer	void*	Input parameter - pointer to memory allocated by user
nBufferSize	size_t	The size of allocated memory. Currently this parameter MUST be equal to size returned by KYFG_StreamGetInfo() with info command KY_STREAM_INFO_PAYLOAD_SIZE
pPrivate	void*	This parameter is currently ignored
pBufferHandle	STREAM_BUFFER_HANDLE *	Output parameter - pointer to STREAM_BUFFER_HANDLE variable that will hold handle of newly announced frame buffer

Return value

FGSTATUS - Status and error report.

12.11 KYFG_BufferGetInfo()

Retrieves information about previously announced buffer.

```
FGSTATUS KYFG_BufferGetInfo(
        STREAM_BUFFER_HANDLE streamBufferHandle,
        KY_STREAM_BUFFER_INFO_CMD cmdStreamBufferInfo,
        void *pInfoBuffer,
        size_t *pInfoSize,
        KY_DATA_TYPE *pInfoType);
```

Parameter name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE	Handle of a stream buffer
cmdStreamBufferInfo	KY_STREAM_BUFFER_INFO_CMD	Specifies what information is being requested. Possible values are: • KY_STREAM_BUFFER_INFO_BASE. The function will return Base address of the buffer memory. ‘pInfoBuffer’ must be NULL or point to a pointer variable. • KY_STREAM_BUFFER_INFO_SIZE – reserved for future

		enhancements • KY_STREAM_BUFFER_INFO_USER_PTR – reserved for future enhancements • KY_STREAM_BUFFER_INFO_TIMESTAMP – reserved for future enhancements • KY_STREAM_BUFFER_INFO_ID – Unique ID of buffer in the stream
pInfoBuffer	void *	Pointer to user variable that will be filled with required information. Can be NULL. ***Please see remarks!***
pInfoSize	size_t *	Pointer to size of provided pInfoBuffer. Can be NULL. In: size of the provided pInfoBuffer in bytes. Out: minimum required size of pInfoBuffer to hold requested information. ***Please see remarks!***
pInfoType	KY_DATA_TYPE *	Pointer to data type of pInfoBuffer content. Can be NULL. Out: data type of pInfoBuffer for requested information. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

1. If information type is known, provide a valid buffer of the correct size to fill in requested information. In this case pInfoSize and pInfoType can be NULL.
2. Alternatively, the information request can be done in two steps:
   a. Check which size (pInfoSize) and data type (pInfoType) should pInfoBuffer represent. Provide valid pInfoSize and/or pInfoType and call function with pInfoBuffer = NULL.
   b. Provide a valid buffer to fill in the requested information. pInfoSize should be NULL or size of specified pInfoBuffer.
3. If pInfoType is a string, the size includes the termination 0.

12.12 KYFG_BufferToQueue()

Moves a previously announced buffer to specified queue.

```
FGSTATUS KYFG_BufferToQueue(
        STREAM_BUFFER_HANDLE streamBufferHandle,
        KY_ACQ_QUEUE_TYPE dstQueue);
```

Parameter name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE	Handle of a stream buffer
dstQueue	KY_ACQ_QUEUE_TYPE	Destination queue: • KY_ACQ_QUEUE_INPUT – buffers in this queue are ready to be filled with data. • KY_ACQ_QUEUE_OUTPUT - buffers in this queue have been filled and awaiting user processing. This queue is filled internally by library. An ability for application to move buffers to this queue is reserved for future library enhancements and currently will result in error code FGSTATUS_DESTINATION_QUEUE_NOT_SUPPORTED • KY_ACQ_QUEUE_UNQUEUED – reserved for future enhancements • KY_ACQ_QUEUE_AUTO – reserved for future enhancements

Return value

FGSTATUS - Status and error report.

12.13 KYFG_BufferQueueAll()

Moves all frame buffers bound to specified stream from one queue to another queue.

```
FGSTATUS KYFG_BufferQueueAll(
        STREAM_HANDLE streamHandle,
        KY_ACQ_QUEUE_TYPE srcQueue
        KY_ACQ_QUEUE_TYPE dstQueue);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	Handle of a stream
srcQueue	KY_ACQ_QUEUE_TYPE	Source queue. See KYFG_BufferToQueue() description for possible values
dstQueue	KY_ACQ_QUEUE_TYPE	Destination queue. See KYFG_BufferToQueue() description for possible values

Return value

FGSTATUS - Status and error report.

12.14 KYFG_StreamDelete()

Deletes a stream. Any memory allocated by user is NOT freed by this function. All memory allocated by library is freed and all API handles bound to the stream became invalid.

```
FGSTATUS KYFG_StreamDelete(
        STREAM_HANDLE streamHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream

Return value

FGSTATUS - Status and error report.

12.15 Example of code using Cyclic buffers

```
   void Stream_callback_func(void* userContext, STREAM_HANDLE streamHandle)
   {
        static void* data = NULL;
        static KYBOOL copyingDataFlag = KYFALSE;
        long long width = 0, height = 0, totalFrames = 0, buffSize = 0;
        void* buffData;
        if(0 == streamHandle)          // callback with indicator for acquisition stop
        {
            copyingDataFlag = KYFALSE;
            return;
        }
        width = KYFG_GetCameraValueInt(streamHandle, “Width”);
        height = KYFG_GetGrabberValueInt(streamHandle, “Height”);
        totalFrames = KYFG_GetGrabberValueInt(streamHandle, "RXFrameCounter");
        buffSize = KYFG_BufferGetSize(streamHandle);          // get buffer size
        buffIndex = KYFG_BufferGetFrameIndex(streamHandle);
        buffData = KYFG_BufferGetPtr(streamHandle, buffIndex);// get pointer of buffer data

        if(KYFALSE == copyingDataFlag)
        {
            copyingDataFlag = KYTRUE;
            data = (void*)realloc(data, buffSize);                // allocate size for local buffer
            if (NULL == data)
            {
                return;
            }
            printf("Callback of buffer %X, width: %d, height: %d, total frames acquired: %d",
                                                  streamHandle, width, height, totalFrames);
            memcpy(data, buffData, buffSize);                        // copy data to local buffer
            //... Show Image with data …
            copyingDataFlag = KYFALSE;
        }
    }

    int main(int argc, char* argv[])
    {
        FGHANDLE handle;
        CAMHANDLE CamHandleArray[4] = {0};
        int nDetectedCameras = 0;
         …
        KYFG_CameraScan(handle, CamHandleArray, &nDetectedCameras);
        if ( nDetectedCameras > 0 )
        {
            KYFG_CameraCallbackRegister(CamHandleArray[0], Stream_callback_func, NULL);
        }
         …
        while(1){}
        return 0;
    }
```

12.16 Example of code using Queued buffers

```
    void Stream_callback_func(STREAM_BUFFER_HANDLE streamBufferHandle,
                              void* userContext)
    {
        // process data associated with given stream buffer
        unsigned char* pFrameMemory;
        KYFG_BufferGetInfo(streamBufferHandle,
                              KY_STREAM_BUFFER_INFO_BASE,
                              &pFrameMemory,
                              NULL,
                              NULL);
        …
        // return stream buffer to input queue
        KYFG_BufferToQueue(streamBufferHandle, KY_ACQ_QUEUE_INPUT);
    }
```

```
int main(int argc, char* argv[])
{
    FGHANDLE handle;
    CAMHANDLE CamHandleArray[4] = {0};
    STREAM_HANDLE cameraStreamHandle;
    int nDetectedCameras = 0;
    size_t frameDataSize, frameDataAligment;
    static const int allocFrames = 16;
    STREAM_BUFFER_HANDLE streamBufferHandle[allocFrames] = {0};
    …
    KYFG_CameraScan(handle, CamHandleArray, &nDetectedCameras);

    if ( nDetectedCameras > 0 )
    {
        … // update camera/grabber buffer dimensions parameters before stream creation
        // create stream and assign appropriate runtime acquisition callback function
        KYFG_StreamCreate(CamHandleArray[0], &cameraStreamHandle, 0);
        KYFG_StreamBufferCallbackRegister(cameraStreamHandle,
                                          Stream_callback_func,
                                          NULL);

        // Retrieve information about required frame buffer size and alignment
        KYFG_StreamGetInfo(cameraStreamHandle,
                           KY_STREAM_INFO_PAYLOAD_SIZE,
                           &frameDataSize,
                           NULL,
                           NULL);
        KYFG_StreamGetInfo(cameraStreamHandle,
                           KY_STREAM_INFO_BUF_ALIGNMENT,
                           &frameDataAligment,
                           NULL,
                           NULL);
        // allocate memory for desired number of frame buffers
        for (iFrame = 0; iFrame < allocFrames; iFrame++)
        {
            void * pBuffer = _aligned_malloc(frameDataSize, frameDataAligment);
            KYFG_BufferAnnounce(cameraStreamHandle,
                                pBuffer,
                                frameDataSize,
                                NULL,
                                &streamBufferHandle[iFrame]);
        }
        // put all buffers to input queue
        KYFG_BufferQueueAll(cameraStreamHandle,
                            KY_ACQ_QUEUE_UNQUEUED,
                            KY_ACQ_QUEUE_INPUT);
    }
    // start acquisition
    KY_CameraStart (CamHandleArray[0], cameraStreamHandle, 0)
     …
    while(1){}
    return 0;
}
```

Note:

The queued buffers example code above shows an allocation of 16 frames (allocFrames = 16). Since we allocated 16 frames, their IDs are running from 0 till 15 and then wrap over, i.e. frames are reused during acquisition, thus the total number of frames may be bigger than the actual frames in queue, for example: (ID: 15, total frames: 29).

13.1 KYFG_CameraStart()

Starts transmission for the chosen camera. The chosen stream would be filled with data from the camera. Only 1 stream can be active at a time, per camera. Number of frames to be acquired may be set, while 0 frames indicate continues acquisition mode.

```
FGSTATUS KYFG_CameraStart(
        CAMHANDLE camHandle,
        STREAM_HANDLE streamHandle,
        int frames);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
streamHandle	STREAM_HANDLE	API handle to data stream for selected camera
frames	Int	Number of frames to be acquired. After the specified number of frames were acquired, the camera would be stopped. 0 for continues acquisition mode.

Return value

FGSTATUS - Status and error report.

13.2 KYFG_CameraStop()

Stops transmission for the chosen camera.

```
FGSTATUS KYFG_CameraStop (CAMHANDLE camHandle);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera

Return value

FGSTATUS - Status and error report.

14.1 KYFG_ReadPortReg()

Read bootstrap registers from specific port, 32bit value each time. This function access the link directly disregarding the camera connection topology.

```
FGSTATUS KYFG_ReadPortReg(
        FGHANDLE handle,
        int port,
        uint64_t address,
        uint32_t * pData);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
port	int	Frame Grabber port index
address	uint64_t	Address of the register
pData	uint32_t *	Pointer to register that will hold read data

Return value

FGSTATUS - Status and error report.

14.2 KYFG_ReadPortBlock()

Read buffer of specified size from specific port. This function access the link directly disregarding the camera connection topology.

```
FGSTATUS KYFG_ReadPortBlock(
        FGHANDLE handle,
        int port,
        uint64_t address,
        void * pBuffer,
        uint32_t * pSize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
port	int	Device port index
address	uint64_t	Start address of the data to read
pBuffer	uint32_t *	Pointer to buffer that will hold read data
pSize	uint32_t *	Pointer to size of the data buffer. In: size in bytes of buffer to read Out: size of read bytes

Return value

FGSTATUS - Status and error report.

14.3 KYFG_WritePortReg()

Write bootstrap registers from specific port, 32bit value each time. This function access the link directly disregarding the camera connection topology.

```
FGSTATUS KYFG_WritePortReg(
        FGHANDLE handle,
        int port,
        uint64_t address,
        uint32_t data);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
port	int	Frame Grabber port index
address	uint64_t	Address of the register
data	uint32_t	Bootstrap registers value

Return value

FGSTATUS - Status and error report.

14.4 KYFG_WritePortBlock()

Write buffer of specified size to specific port. This function access the link directly disregarding the camera connection topology.

```
FGSTATUS KYFG_WritePortBlock(
        FGHANDLE handle,
        int port,
        uint64_t address,
        const void * pBuffer,
        uint32_t * pSize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
port	int	Frame Grabber port index
address	uint64_t	Start address of the data to write
pBuffer	const void *	Pointer to buffer data to write
pSize	uint32_t *	Pointer to size of the data buffer. In: size in bytes of buffer to write Out: size of written bytes

Return value

FGSTATUS - Status and error report.

14.5 KYFG_CameraReadReg()

Direct read data buffer from the selected camera.

```
FGSTATUS KYFG_CameraReadReg(
        CAMHANDLE camHandle,
        uint64_t address,
        void* pBuffer,
        uint32_t * pSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
address	uint64_t	Start address of the data to read
pBuffer	void *	Pointer to buffer that will hold read data
pSize	uint32_t *	Pointer to size of the data buffer. In: size in bytes of buffer to read Out: size of read bytes

Return value

FGSTATUS - Status and error report.

14.6 KYFG_CameraWriteReg()

Direct write data buffer to the selected camera.

```
FGSTATUS KYFG_CameraWriteReg(
        CAMHANDLE camHandle,
        uint64_t address,
        const void* pBuffer,
        uint32_t * pSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
address	uint64_t	Start address of the data to read
pBuffer	const void *	Pointer to buffer data to write
pSize	uint32_t *	Pointer to size of the data buffer. In: size in bytes of buffer to read Out: size of read bytes

Return value

FGSTATUS - Status and error report.

15 IO configurations

15.1 IO selectors

KAYA's Frame Grabbers provide a vast variety of configurable I/Os. This is subject to device hardware, firmware and software capabilities:

- 4 Encoders
- Per channel Frame Grabber Triggers and Camera Triggers
- 8 Timers
- IO Outputs
- IO Inputs

Parameter naming is according to "Gen<i>Cam Standard Features Naming Convention version 2.1".

I/Os are configurable via Gen<i>Cam interface using selectors to specify the specific I/O to be configured. In order to configure the specific I/O feature follow these steps:

1. Set selector value of specific group (Timer, Stream Trigger, Camera Triggers, Encoder or GPIO) of the required item to be configured.
2. Set the selected I/O properties to the required value.

In order to configure the selector and values use KYFG_SetGrabberValue() and KYFG_GetGrabberValue() (or one of the provided sub-functions).

15.2 Configuration fields

Complete available triggers list and possible configurations can be found in "KAYAs FG Programming Start-up Guide" document.

Remark

*For I/O source options of Firmware Version 1.xx follow Table 5 and **Table 6** located in Appendices section.

Example code

1. This example shows how to set the Timer 2 trigger source to be in continues mode.

```
FGHANDLE handle;  // maximum 4 cameras can be connected
long long timerSelectorValue = 2;

// select the timer to configure
KYFG_SetGrabberValueEnum(handle,"TimerSelector", timerSelectorValue);
// select the timer trigger source configuration
KYFG_SetGrabberValueEnum_ByValueName(handle, "TimerTriggerSource",
                                                  "KY_CONTINUOUS");
```

2. The following example illustrates how to configure TTL 0 to be an output signal source generating a square wave using connected Timer 0 at 10Hz frequency (every 100ms).

   Also configuring TTL 1 to be an input for incoming signal, which then will become the source of camera trigger over the coax master channel of the camera.

   To complete the setup TTL 0 is connected to TTL 1 in order to pass the signal from Timer 0 to the camera over coax.

   This setup both generates a trail of trigger packets to camera and a way to connect an oscilloscope, for example, to TTL 0 and sample the generated signals.

```
/** Setup:
 * 1) Connect oscilloscope to TTL 0 pin
 * 2) Connect TTL 0 to TTL 1 pin
 * 3) TTL 1 input will be sending triggers to camera over coax
 */
FGHANDLE handle;
// 1) Configure timer 0 to create square wave
// select the timer trigger source configuration
KYFG_SetGrabberValueEnum_ByValueName(handle,"TimerSelector", "Timer0" );
// set continues mode
KYFG_SetGrabberValueEnum_ByValueName(handle, "TimerTriggerSource", "KY_CONTINUOUS");
// example with 10Hz:
KYFG_SetGrabberValueFloat(handle, "TimerDelay", 50000.0);  // 50ms
KYFG_SetGrabberValueFloat(handle, "TimerDuration", 50000.0);  // 50ms
// example with 10KHz:
// KYFG_SetGrabberValueFloat(handle, "TimerDelay", 50.0);  // 50us
// KYFG_SetGrabberValueFloat(handle, "TimerDuration", 50.0);  // 50us
// explanation:
// 50,000us Delay + 50,000us Duration = 100ms total clock pulse = 10Hz
// 50us Delay + 50us Duration = 100us total clock pulse = 10KHz

// >------------------------------------------------------------------------------------------------------------------------<
// 2) Setting timer to be source of TTL 0
// select the TTL 0 settings
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineSelector", "KY_TTL_0");
// set TTL 0 direction to output
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineMode", "Output");
// select TTL 0 source as timer 0
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineSource", "KY_TIMER_ACTIVE_0");

// >------------------------------------------------------------------------------------------------------------------------<
// 3) Configuring TTL 1 to be input
// select TTL 1 settings
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineSelector", "KY_TTL_1");
// set TTL 1 direction to input
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineMode", "Input");
// disable TTL 1 source
KYFG_SetGrabberValueEnum_ByValueName(handle, "LineSource", "KY_DISABLED");

// >------------------------------------------------------------------------------------------------------------------------<
// 4) Setting TTL 1 to be trigger for camera
// select the camera trigger settings
KYFG_SetGrabberValueEnum(handle, "CameraSelector", 0);          // select the camera to work with
// enable camera trigger
KYFG_SetGrabberValueEnum_ByValueName(handle, "CameraTriggerMode", "On");
// select camera trigger source as TTL 1
KYFG_SetGrabberValueEnum_ByValueName(handle, "CameraTriggerSource", "KY_TTL_1");
```

16.1 KYFG_CheckUpdateFile

Retrieves information about firmware contained in supplied binary file and current firmware of the card.

```
FGSTATUS KYFG_CheckUpdateFile (
        FGHANDLE handle,
        const char* file,
        uint16_t *pFlashMinorRev,
        uint16_t *pFlashMajorRev,
        uint16_t *pFileMinorRev,
        uint16_t *pFileMajorRev,
        uint16_t *pFlashVendorId,
        uint16_t *pFlashBoardId,
        uint16_t *pFileVendorId,
        uint16_t *pFileBoardId);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Video Processor
file	const char*	Full path to a firmware update file
pFlashMinorRev	uint16_t *	Pointer to uint16_t that will be filled with minor revision number of current firmware
pFlashMajorRev	uint16_t *	Pointer to uint16_t that will be filled with major revision number of current firmware
pFileMinorRev	uint16_t *	Pointer to uint16_t that will be filled with minor revision number of firmware contained in the update file
pFileMajorRev	uint16_t *	Pointer to uint16_t that will be filled with major revision number of firmware contained in the file
pFlashVendorId	uint16_t *	Pointer to uint16_t that will be filled with vendor ID of current firmware
pFlashBoardId	uint16_t *	Pointer to uint16_t that will be filled with board ID of current firmware
pFileVendorId	uint16_t *	Pointer to uint16_t that will be filled with vendor ID of firmware contained in the file
pFileBoardId	uint16_t *	Pointer to uint16_t that will be filled with board ID of firmware contained in the file

Return value

FGSTATUS - Status and error report.

16.2 KYFG_LoadFirmware

Updates device firmware from supplied binary file. Progress is reporterd via supplied callback function.

```
FGSTATUS KYFG_LoadFirmware (
        FGHANDLE handle,
        const char* file,
        UPDATE_CALLBACK callback,
        void* context);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Video Processor
callback	UPDATE_CALLBACK	Pointer to callback function to be called during update
context	void*	User context will passed as parameter to each call of ‘callback’

Note: The software detects an outdated firmware and disables all sets of operation, except firmware update, which should be performed in order to proceed.

17.1 API handles

- FGHANDLE – API handle for Frame Grabber’s components functionality.

  INVALID_FGHANDLE – Markup for invalid Frame Grabber API handle.
- CAMHANDLE – API handle for Camera’s components functionality.

  INVALID_CAMHANDLE – Markup for invalid Camera API handle.
- BUFFHANDLE – API handle for acquisition buffer components functionality

  INVALID_BUFFHANDLE – Markup for invalid Buffer API handle.
- STREAM_HANDLE – API handle for acquisition stream components functionality

  INVALID_STREAMHANDLE – Markup for invalid Stream API handle.

17.2 KYBOOL

Definition for simple C code implementation for Boolean values.

```
typedef unsigned char KYBOOL;
        #define KYTRUE      1     // determine a true value
        #define KYFALSE 0  // determine a false value
```

17.3 FGCallback

General runtime transmission callback function prototype. Callbacks are issued whenever a new frame acquisition or generation is complete. Data can be retrieved using the Buffer interface (DEPRECATED) functions. A callback with buffHandle zero, indicates the stop of acquisition for camera associated with current buffer.

```
typedef void(KYFG_CALLCONV *FGCallback)(
        BUFFHANDLE buffHandle,
        void* userContext);
```

17.4 StreamBufferCallback

Runtime acquisition or generation callback function prototype for a specific stream. Callbacks are issued whenever a new frame acquisition or generation is complete from selected stream. Data can be retrieved using the Stream interface functions. A callback with streamBufferHandle zero indicates the stop of acquisition for stream associated with registered callback function.

```
typedef void(KYFG_CALLCONV * StreamBufferCallback)(
        STREAM_BUFFER_HANDLE streamBufferHandle,
        void* userContext);
```

17.5 CameraCallback

Runtime acquisition callback function prototype for a specific camera. Callbacks are issued whenever a new frame acquisition is complete from selected camera. Data can be retrieved using the Buffer interface (DEPRECATED) functions. A callback with buffHandle zero, indicates the stop of acquisition for camera associated with registered callback function.

```
typedef void(KYFG_CALLCONV *CameraCallback)(
        void* userContext
        BUFFHANDLE buffHandle);
```

17.6 FGAuxDataCallback

Callback function which will be called when various auxiliary data is generated. Auxiliary data is passed to user callback as pointer to KYFG_AUX_DATA structure parameter. Each Auxiliary data will be interpreted differently according to context and message ID.

```
typedef void (KYFG_CALLCONV *FGAuxDataCallback)(
        KYFG_AUX_DATA* pData,
        void* context);
```

17.7 KYDeviceEventCallBack

Callback function which will be called when various device events are generated. Details of an event is passed to user callback as pointer to KYDEVICE_EVENT structure. Each event data will be interpreted differently according to context and event ID.

```
typedef void (KYFG_CALLCONV * KYDeviceEventCallBack)(
        void* context,
        KYDEVICE_EVENT* pEvent);
```

17.8 ParameterCallback

Callback function which will be called during execution of

KYFG_GetGrabberConfigurationParameterDefinitions() or

KYFG_GetCameraConfigurationParameterDefinitions()

```
typedef void (KYFG_CALLCONV *ParameterCallback)
        (void* userContext,
        NodeDescriptor* nodeDescriptor,
        int grouppingLevel);
```

17.9 UPDATE_CALLBACK

Runtime firmware update callback function prototype. Callbacks are issued during firmware update to report progress.

```
typedef KYBOOL(*UPDATE_CALLBACK)(const UPDATE_STATUS* UpdateStatus, void*
context);
```

18.1 FGSTATUS

Execution of system error and status. Defines the status returned after each function execution. While some error statuses are general some point to a specific error.

Enumeration Field	Value	Description
FGSTATUS_OK	0x3000	The operation has successfully executed
FGSTATUS_UNKNOWN_HANDLE	0x3001	Unknown API handle
FGSTATUS_HW_NOT_FOUND	0x3002	Error with hardware. Hardware function failed to execute.
FGSTATUS_BUSY	0x3003	Can’t execute function at the current moment, the FG is busy
FGSTATUS_FILE_NOT_FOUND	0x3004	Wasn’t able to open file in given file path
FGSTATUS_FILE_READ_ERROR	0x3005	Wasn’t able to read file, error in file or the file is to long
FGSTATUS_CONFIG_NOT_LOADED	0x3006	Can’t load current camera configuration
FGSTATUS_INVALID_VALUE	0x3007	The value given as parameter is out of acceptable range
FGSTATUS_MAX_CONNECTIONS	0x3008	No more devices can be connected to system
FGSTATUS_MEMORY_ERROR	0x3009	General memory error or an allocation has failed
FGSTATUS_WRONG_PARAMETER_NAME	0x300A	Parameter name wasn’t found (names are defined by XML file)
FGSTATUS_WRONG_PARAMETER_TYPE	0x300B	Unsupported parameter type
FGSTATUS_GENICAM_EXCEPTION	0x300C	General Gen<i>Cam exception
FGSTATUS_OUT_OF_RANGE_ADDRESS	0x300D	The specified address is not suitable for writing
FGSTATUS_COULD_NOT_START	0x300E	FG couldn’t start acquisition
FGSTATUS_COULD_NOT_STOP	0x300F	FG couldn’t stop the acquisition
FGSTATUS_XML_FILE_NOT_LOADED	0x3010	No valid XML file source was found
FGSTATUS_INVALID_VALUES_FILE	0x3011	Unsupported values file was loaded
FGSTATUS_NO_REQUIRED_PARAMETERS_SECTION	0x3012	Corrupted values save file
FGSTATUS_WRONG_PARAMETERS_SECTION	0x3013	Saved values configurations for loading wasn’t found
FGSTATUS_VALUE_HAS_NO_SELECTOR	0x3014	The parameter is not a part of a selector type field
FGSTATUS_CALLBACK_NOT_ASSIGNED	0x3015	No callback is assigned for data retrieval
FGSTATUS_HANDLE_DOES_NOT_MATCH_CONFIG	0x3016	The value of Camera Selector doesn't match the provided Camera handle This will indicate that a wrong

		CAMHANDLE is introduced for set/get Grabber parameter functions, which contradictory the “CameraSelector” currently set.
FGSTATUS_BUFFER_TOO_SMALL	0x3017	Provided buffer length is too small to hold the amount of information needed to be filled in the provided buffer
FGSTATUS_EXCEEDED_MAX_CAMERA_CONNECTIONS	0x3100	Number of connected cameras exceeds the maximum allowed connected cameras
FGSTATUS_UNKNOWN_ERROR	0x3FFF	Unknown error

18.2 CXP_LINK_SPEED

Available CoaXPress speed values.

Enumeration Field	Value	Description
LINK_SPEED_CXP1	0x28	CXP1 – 1. 25Gbps
LINK_SPEED_CXP2	0x30	CXP2 – 2.5Gbps
LINK_SPEED_CXP3	0x38	CXP3 – 3.125Gbps
LINK_SPEED_CXP5	0x40	CXP5 – 5Gbps
LINK_SPEED_CXP6	0x48	CXP6 – 6.25Gbps
LINK_SPEED_CXP10	0x50	CXP10 – 10Gbps
LINK_SPEED_CXP12	0x58	CXP12 – 12.5Gbps

18.3 KY_CAM_PROPERTY_TYPE

Gen<i>Cam field type. Camera configuration field type as stated in loaded XML file.

Enumeration Field	Value	Description
PROPERTY_TYPE_INT	0x00	Camera configuration of Integer type
PROPERTY_TYPE_BOOL	0x01	Camera configuration of Boolean type
PROPERTY_TYPE_STRING	0x02	Camera configuration of String type
PROPERTY_TYPE_FLOAT	0x03	Camera configuration of Floating Point type
PROPERTY_TYPE_ENUM	0x04	Camera configuration of Enumeration type
PROPERTY_TYPE_COMMAND	0x05	Camera configuration of Command type
PROPERTY_TYPE_REGISTER	0x06	Camera configuration of Register type
PROPERTY_TYPE_UNKNOWN	-1	Camera configuration of an unknown type

18.4 VIDEO_DATA_WIDTH

Data width of the pixel, defined in section 9.4.1.2 of the JIIA CXP standard document

Enumeration Field	Value	Description
DATA_WIDTH_UNKNOWN	0x00	Unknown number of bits per pixel data
DATA_WIDTH_8BIT	0x01	8 bit per pixel data
DATA_WIDTH_10BIT	0x02	10 bit per pixel data
DATA_WIDTH_12BIT	0x03	12 bit per pixel data
DATA_WIDTH_14BIT	0x04	14 bit per pixel data
DATA_WIDTH_16BIT	0x05	16 bit per pixel data

18.5 VIDEO_DATA_TYPE

Data types of the pixel, defined in section 9.4.1 of the JIIA CXP standard document.

Enumeration Field	Value	Description
DATA_TYPE_MONO	0x01	This is used for luminance data. This has no sub-types. This is defined in Table 27 of JIIA CXP standard document
DATA_TYPE_PLANAR	0x02	This is used for planar data, such as individual red, green or blue planes, additional alpha (overlay) planes, or the separate planes in YUV420. This is defined in Table 28 JIIA CXP standard document. Subtypes include all the DATA_SUBTYPE_PLANAR_xx
DATA_TYPE_BAYER	0x03	This is used for Bayer data. This is defined in Table 29 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_BAYER_xx
DATA_TYPE_RGB	0x04	This is used for RGB data, transmitted in the order red, green, blue. This has no sub-types. This is defined in Table 30 JIIA CXP standard document.
DATA_TYPE_RGBA	0x05	This is used for RGBA data, where “A” is the alpha (or overlay) plane, transmitted in the order red, green, blue, alpha. This has no sub-types. This is defined in Table 31 JIIA CXP standard document.
DATA_TYPE_YUV	0x06	This is used for YUV data. This is defined in Table 32 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_YUV_xxx
DATA_TYPE_YCBCR601	0x07	This is used for YCbCr data, as specified by ITU-R BT.601.This is defined in Table 33 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_UCBCR_xxx
DATA_TYPE_YCBCR709	0x08	This is used for YCbCr data, as specified by ITU-R BT.709. This is defined in Table 34 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_UCBCR_xxx

18.6 VIDEO_DATA_SUBTYPE

Data sub-types of the pixel, defined in section 9.4.1 of the JIIA CXP standard document.

Enumeration Field	Value	Description
DATA_SUBTYPE_NONE	0x00	None
DATA_SUBTYPE_PLANAR_RY	0x01	Standard usage: R, Y
DATA_SUBTYPE_PLANAR_GUCB	0x02	Standard usage: G, U, Cb
DATA_SUBTYPE_PLANAR_BVCR	0x03	Standard usage: B, V, Cr
DATA_SUBTYPE_BAYER_GR	0x01	1st line transmission order G, R. 2nd line transmission order B, G
DATA_SUBTYPE_BAYER_RG	0x02	1st line transmission order R, G. 2nd line transmission order G, B
DATA_SUBTYPE_BAYER_GB	0x03	1st line transmission order G, B. 2nd line transmission order R, G
DATA_SUBTYPE_BAYER_BG	0x04	1st line transmission order B, G. 2nd line transmission order G, R
DATA_SUBTYPE_YUV_411	0x01	Transmission order Y, Y, U, Y, Y, V
DATA_SUBTYPE_YUV_422	0x02	Transmission order Y, U, Y, V
DATA_SUBTYPE_YUV_444	0x03	Transmission order Y, U, V
DATA_SUBTYPE_YCBCR_411	0x01	Transmission order Y, Y, Cb, Y, Y, Cr
DATA_SUBTYPE_YCBCR_422	0x02	Transmission order Y, Cb, Y, Cr
DATA_SUBTYPE_YCBCR_444	0x03	Transmission order Y, Cb, Cr

19.1 KYFGCAMERA_INFO

Camera configuration information. These fields are updated when the camera is connected using KYFG_CameraOpen2(). Changed values are being updated in runtime.

Structure Field	Type	Description
master_link	unsigned char	The master link channel
link_mask	unsigned char	The mask of connected links
link_speed	CXP_LINK_SPEED	The current connection speed (according to CoaXPress specification)
deviceVersion	char [31..0]	Camera version
deviceVendorName	char [31..0]	Vendor name
deviceManufacturerInfo	char [47..0]	Additional manufacturer info
deviceModelName	char [31..0]	Camera model name
deviceID	char [15..0]	Device id
deviceUserID	char [15..0]	Device user id
outputCamera	KYBOOL	KYTRUE if this is output camera, i.e. used for stream generation, KYFALSE otherwise
virtualCamera	KYBOOL	KYTRUE if this is virtual camera, i.e. internally implemented stream, which does not represent any physical camera, KYFALSE otherwise. This parameter can be KYTRUE only in case of custom firmware implementations.

19.2 KY_STREAM_BUFFER_INFO_CMD

Stream Buffer configuration information.

Enumeration Field	Value	Description
KY_STREAM_BUFFER_INFO_BASE	0	Base address of the buffer memory
KY_STREAM_BUFFER_INFO_SIZE	1	Size of the buffer in bytes.
KY_STREAM_BUFFER_INFO_USER_PTR	2	Private data pointer for the stream buffer.
KY_STREAM_BUFFER_INFO_TIMESTAMP	3	Timestamp the buffer was acquired.
KY_STREAM_BUFFER_INFO_INSTANTFPS	4	Instant FPS calculated from this and previous timestamp.
KY_STREAM_BUFFER_INFO_ID	1000	Unique id of buffer in the stream

19.3 VIDEO_PIXELIF

The pixel format code is formed as shown in Table 24 of JIIA CXP document. Note that the value 0x0000 is reserved for “RAW” data that does not match any defined format, such as user-specific formats.

```
typedef struct _video_pixelif
{
        VIDEO_DATA_WIDTH      data_width    : 4;      // Data Width
        VIDEO_DATA_SUBTYPE  data_subtype : 4;      // Sub-type
        VIDEO_DATA_TYPE        data_type      : 8;      // Data Type
}VIDEO_PIXELIF;
```

Structure Field	Type	Description
data_width	VIDEO_DATA_WIDTH	Data Width (4 bit value)
data_subtype	VIDEO_DATA_SUBTYPE	Sub-type (4 bit value)
data_type	VIDEO_DATA_TYPE	Data Type (8 bit value)

19.4 KYFG_AUX_DATA

Auxiliary data structure holding information of received Auxiliary message.

Structure Field	Type	Description
messageID	uint32_t	Unique ID of received Auxiliry data message. For possible values see Table 2.
dataSize	size_t	Size, in Bytes, of data delivered with Auxiliary message
union:		
data	uint8_t [AUX_DATA_MAX_SIZE]	Data portion interpretation with no context.
io_data	KYFG_IO_AUX_DATA	Data portion interpretation, in case of callback issued by IO controller.
frame_data	KYFG_FRAME_AUX_DATA	Data portion interpretation, in case of new frame arrival.

Message identifier	Description
KYFG_AUX_MESSAGE_ID_IO_CONTROLLER	The message has been generated by IO Controller.

Table 2 : MessageID possible values

19.5 KYFG_FRAME_AUX_DATA

Data portion interpretation of Auxiliary data structure, in case of new frame arrival.

Structure Field	Type	Description
sequence_number	uint32_t	sequential index of the frame within allocated frame buffer
timestamp	uint64_t	frame arrival timestamp in units of nano-seconds (nsec)

19.6 KYFG_IO_AUX_DATA

Data portion interpretation of Auxiliary data structure, in case of callback issued by IO controller.

Structure Field	Type	Description
masked_data	uint64_t	Indicates the state of the I/O controller feature that can generate an event according to Table 3. ***Please see remarks!***
timestamp	uint64_t	Event timestamp in units of nano-seconds (nsec)

Remarks

Additional macros are provided to help extract specific IO controller elements:

1. KYFG_IO_CONTROLLER_MASKED_IO – extract mask of GPIO triggers from IO Auxiliary masked data
2. KYFG_IO_CONTROLLER_MASKED_ENCODERS – extract mask of encoders triggers from IO Auxiliary masked data
3. KYFG_IO_CONTROLLER_MASKED_TIMERS – extract mask of timers triggers from IO Auxiliary masked data
4. KYFG_IO_CONTROLLER_MASKED_CAMERA_TRIGGERS – extract mask of camera triggers from IO Auxiliary masked data
5. KYFG_IO_CONTROLLER_MASKED_TRIGGERS – extract mask of stream triggers from IO Auxiliary masked data

Bit	Function
0	OptoCoupled Input 0
1	OptoCoupled Input 1
2	OptoCoupled Input 2
3	OptoCoupled Input 3
4	OptoCoupled Input 4
5	OptoCoupled Input 5
6	OptoCoupled Input 6
7	OptoCoupled Input 7
8	LVDS Input 0
9	LVDS Input 1
10	LVDS Input 2
11	LVDS Input 3
12	TTL 0
13	TTL 1

14	TTL 2
15	TTL 3
16	TTL 4
17	TTL 5
18	TTL 6
19	TTL 7
20	LVTTL 0
21	LVTTL 1
22	LVTTL 2
23	LVTTL 3
24	LVTTL 4
25	LVTTL 5
26	LVTTL 6
27	LVTTL 7
28	OptoCoupled Output 0
29	OptoCoupled Output 1
30	OptoCoupled Output 2
31	OptoCoupled Output 3
32	OptoCoupled Output 4
33	OptoCoupled Output 5
34	OptoCoupled Output 6
35	OptoCoupled Output 7
36	LVDS Output 0
37	LVDS Output 1
38	LVDS Output 2
39	LVDS Output 3
40	Encoder 0
41	Encoder 1
42	Encoder 2
43	Encoder 3
44	Timer 0
45	Timer 1
46	Timer 2
47	Timer 3
48	Timer 4
49	Timer 5
50	Timer 6
51	Timer 7
52	Camera Trigger 0
53	Camera Trigger 1
54	Camera Trigger 2
55	Camera Trigger 3
56	Camera Trigger 4
57	Camera Trigger 5
58	Camera Trigger 6

59	Camera Trigger 7
60	Acquisition Trigger 0
61	Acquisition Trigger 1
62	Acquisition Trigger 2
63	Acquisition Trigger 3

Table 3 : IO controller Auxiliary data bit mask interpretation

19.7 KYDEVICE_EVENT

Device event structure holds information of a received event. Pointer to this base structure is passed to KYDeviceEventCallBack function. Its ‘eventId’ field should be used to determine what concrete event is passed and the pointer should be C-casted to pointer to a corresponding derived structure.

Structure Field	Type	Description
eventId	KYDEVICE_EVENT_ID	Unique ID of received device event. For possible values see Table 4.

KYDEVICE_EVENT_CAMERA_START_REQUEST	Device detected a remote request to start transmission on a camera.
KYDEVICE_EVENT_CAMERA_CONNECTION_LOST_ID	Device detected a remote request to start transmission on a lost camera.

Table 4 : Device event ID possible values

19.8 KYDEVICE_EVENT_CAMERA_START

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_CAMERA_START_REQUEST.

This structure is derived from KYDEVICE_EVENT and passed to KYDeviceEventCallBack function when ‘eventId’ field is KYDEVICE_EVENT_CAMERA_START_REQUEST. This event is sent when there is a remote request to start acquisition on a camera. Normally application should use KY_CameraStart() function to start acquisition on the specified camera after performing application-specific preparation.

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	Base part of the structure.
camHandle	CAMHANDLE	API handle to a camera

19.9 KYDEVICE_EVENT_CAMERA_CONNECTION_LOST

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_CAMERA_CONNECTION_LOST_ID.

This structure is derived from KYDEVICE_EVENT and passed to KYDeviceEventCallBack function when 'eventId' field is KYDEVICE_EVENT_CAMERA_CONNECTION_LOST_ID. This event is sent when a link loss occur on a detected camera.

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	Base part of the structure.
camHandle	CAMHANDLE	API handle to a camera for which device detected remote request to start acquisition
iDeviceLink	size_t	Size of the expected data in Bytes - device's lost link
iCameraLink	size_t	Size of the expected data in Bytes - camera's lost link

19.10 NodeDescriptor

Descriptor of a node passed to ParameterCallback function.

Structure Field	Type	Description
interfaceType	ParameterInterfaceType	Type of node
paramName	const char*	Machine name of parameter. This name should be used as argument 'paramName' for KYFG_SetGrabberValueXXX() and KYFG_GetGrabberValueXXX() calls
paramDisplayName	const char*	Human readable name of parameter used in GUI
toolTip	const char*	Visual tooltip explaining parameter meaning in GUI
isWritable	bool	`true' if parameter is writable, i.e. KYFG_SetGrabberValueXXX() can be called for it; 'false' otherwise – attempt to set it will result in error FGSTATUS_PARAMETER_NOT_WRITABLE
representation	ParameterRepresentation	Indicates type of GUI element suggested for this parameter representation
visibility	KY_PROPERTY_VISIBILITY	Visibility level. Used in GUI for filtering list visible parameters
descriptorType	NodeDescriptorType	See NodeDescriptorType description

minIntValue	int64_t	Minimum possible / allowed value in case parameter has ‘intfIInteger’ interfaceType
maxIntValue	int64_t	Maximum possible / allowed value in case parameter has ‘intfIInteger’ interfaceType
incIntValue	int64_t	Single increment / decrement step in case parameter has ‘intfIInteger’ interfaceType
curIntValue	int64_t	Current value in case parameter has ‘intfIInteger’ interfaceType
minFloatValue	double	Minimum possible / allowed value in case parameter has ‘intfIFloat’ interfaceType
maxFloatValue	double	Maximum possible / allowed value in case parameter has ‘intfIFloat’ interfaceType
incFloatValue	double	Single increment / decrement step in case parameter has ‘intfIFloat’ interfaceType
floatDisplayPrecision	int64_t	Decimal precision in case parameter has ‘intfIFloat’ interfaceType
curFloatValue	double	Current value in case parameter has ‘intfIFloat’ interfaceType
curBoolValue	bool	Current value in case parameter has ‘intfIBoolean’ interfaceType
curStringValue	const char*	Current value in case parameter has ‘intfIString’ interfaceType
isSelector	bool	`true’ if this node acts as selector for other nodes, ‘false’ otherwise
selectorName	const char*	name of another node that acts as selector for this node, NULL if this node is not selected
pParentNode	NodeDescriptor*	Pointer to parent node in nodes hierarchy

19.10.1 ParameterInterfaceType

intfIValue	Not currently used
intfIBase	Not currently used
intfIInteger	The parameter is of integer type. Get/Set operations expect ‘int64_t’ C type
intfIBoolean	The parameter is of boolean type. Get/Set operations expect ‘bool’ C type
intfICommand	Not currently used
intfIFloat	The parameter is of float type. Get/Set operations expect ‘double’ C type
intfIString	The parameter is of string type. Get/Set operations expect ‘char *’ C type

intfIRegister	Not currently used
intfICategory	The category of parameter. Used for parameters groupping
intfIEnumeration	The parameter is of enumeration type. Get/Set operations expect ‘int64_t’ C type
intfIEnumEntry	The parameter represents one of the possible values of its parent parameter that must be enumeration type. Get/Set operations expect ‘int64_t’ C type
intfIPort	Not currently used

19.10.2 ParameterRepresentation

This enumeration is used to signal a most suitable representation of this parameter in GUI

Linear	A linear slider
Logarithmic	A logarithmic slider
Boolean	A check box
PureNumber	A decimal number edit control (possibly with spins)
HexNumber	A hexadecimal edit control (possibly with spins)
IPV4Address	A IPV4 Address editor
MACAddress	A MAC Address editor
_UndefinedRepresentation	No suggested representation

19.10.3 NodeDescriptorType

The type of NodeDescriptor can be of the following:

Invalid	An error has occurred
NewNode	New node is being announced during enumeration of all nodes
NewEnumEntry	A new entry of previously announced node of type ‘intfIEnumeration’ is being announced
UpdateNode	Current value of described parameter has been changed

19.11 KY_AuthKey

Authentication key secret character array.

Structure Field	Type	Description
secret	unsigned char [32]	Authentication key

19.12 UPDATE_STATUS

Firmware update progress supplied via parameter of UPDATE_CALLBACK

Structure Field	Type	Description
struct_version	int	Currently code initializes this with "1". If more fields will be added to this stuct in future code will be changed and initialize it with "2", etc.

link_mask	uint64_t	bytes already sent.
link_speed	uint64_t	firmware file size.
is_writing	KYBOOL	Indicates current phase: KYTRUE - writing new firmware, KYFALSE - validating new firmware.

20.1 Connection and Info

20.1.1 KYFG_Open()

Connect to a specific Frame Grabber and initializes all required components.

```
def  KYFG_Open( index )
```

Parameter name	Type	Description
index	int	The index, from scan result list acquired with KYFG_Scan() function, of the Frame Grabber device to open. ***Please see remarks!***

Return value

handle - An API handle to Frame Grabber device. Type: FGHANDLE

INVALID_FGHANDLE will indicate a wrong, impossible or unsupported connection.

Remarks

When calling the function with index of -1, a connection to the first found Frame Grabber will be established, such function call eliminates the need for KYFG_Scan() function call.

20.1.2 KYFG_Scan()

Scans for KAYA devices currently connected to the PC PCIe slots and available virtual devices and optionally fills array with devise IDs.

```
def  KYFG_Scan( count )
```

Parameter name	Type	Description
count	int	Number of devices to assign to pids_info list (assume pids_info array is valid)

Return value

Status - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / KYFGLIB_DLL_NOT_FOUND

n - Number of connected hardware and virtual devices. Type: int
pids_info - List of scanned devices. Type: list

If count is not 0 the returned list is filled with each Device Product ID (pid).

Remarks

If count parameter is called with 0, pids_info list will not be filled and the function will only return number of connected and virtual Frame Grabbers.

20.1.3 KY_DeviceDisplayName() (DEPRECATED)

This function is deprecated. New applications should use function KY_DeviceInfo() and use pInfo.szDeviceDisplayName to retrieve device name.

Retrieve device name for the specified index.

```
def  KY_DeviceDisplayName( index )
```

Parameter name	Type	Description
index	int	Discovered device index

Return value

Status - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / other error

fg_name - The name of Frame Grabber issued by the specified index.

20.1.4 KY_DeviceInfo()

Retrieve device Information

```
def KY_DeviceInfo(index)
```

Parameter name	Type	Description
index	int	Discovered device index

Return value

Status – Status of function

pInfo - Structure with info about the relevant device. Type: KY_DEVICE_INFO

```
class KY_DEVICE_INFO:
    def __init__(self):
        self.szDeviceDisplayName          = ""
        self.nBus                         = 0
        self.nSlot                        = 0
        self.nFunction                    = 0
        self.DevicePID                    = 0
        self.isVirtual                    = False
```

```
class KYFGCAMERA_INFO:
    def __init__(self):
        self.master_link                  = 0
        self.link_mask                    = 0
        self.link_speed                   = 0
        self.stream_id                    = 0
        self.deviceVersion                = ""
        self.deviceVendorName             = ""
        self.deviceManufacturerInfo       = ""
        self.deviceModelName              = ""
        self.deviceID                     = ""
        self.deviceUserID                 = ""
        self.outputCamera                 = False
        self.virtualCamera                = False
```

```
class KY_CAM_PROPERTY_TYPE:
    PROPERTY_TYPE_UNKNOWN             = -1
    PROPERTY_TYPE_INT                 = 0x00
    PROPERTY_TYPE_BOOL                = 0x01
    PROPERTY_TYPE_STRING              = 0x02
    PROPERTY_TYPE_FLOAT               = 0x03
    PROPERTY_TYPE_ENUM                = 0x04
    PROPERTY_TYPE_COMMAND             = 0x05
    PROPERTY_TYPE_REGISTER            = 0x06
```

```
class KY_STREAM_BUFFER_INFO_CMD:
    KY_STREAM_BUFFER_INFO_BASE                  = 0
    KY_STREAM_BUFFER_INFO_SIZE                  = 1
    KY_STREAM_BUFFER_INFO_USER_PTR              = 2
    KY_STREAM_BUFFER_INFO_TIMESTAMP             = 3
    KY_STREAM_BUFFER_INFO_ INSTANTFPS           = 4
    KY_STREAM_BUFFER_INFO_ID                    = 1000
```

```
class KYFG_AUX_DATA:
    messageID           = 0
    reserved            = False
    dataSize            = 0
```

```
class KYFG_FRAME_AUX_DATA(KYFG_AUX_DATA):
    sequence_number     = 0
    timestamp           = 0
    reserved            = 0
```

```
class KYFG_IO_AUX_DATA(KYFG_AUX_DATA):
    masked_data          = 0
    timestamp            = 0
```

```
class KYFG_FRAME_RAW(KYFG_AUX_DATA):
    data          = [0 for i in range(AUX_DATA_MAX_SIZE)]
```

```
class KY_ACQ_QUEUE_TYPE:
    KY_ACQ_QUEUE_INPUT                    =   0
    KY_ACQ_QUEUE_OUTPUT                   =   1
    KY_ACQ_QUEUE_UNQUEUED                 =   2
    KY_ACQ_QUEUE_AUTO                     =   3
```

```
class KY_AuthKey:
    secret               = [0 for i in range(KY_AUTHKEY_SIZE)]
```

```
class KY_DATA_TYPE:
    KY_DATATYPE_UNKNOWN                   = 0
    KY_DATATYPE_STRING                    = 1
    KY_DATATYPE_STRINGLIST                = 2
    KY_DATATYPE_INT16                     = 3
    KY_DATATYPE_UINT16                    = 4
    KY_DATATYPE_INT32                     = 5
    KY_DATATYPE_UINT32                    = 6
    KY_DATATYPE_INT64                     = 7
    KY_DATATYPE_UINT64                    = 8
    KY_DATATYPE_FLOAT64                   = 9
    KY_DATATYPE_PTR                       = 10
    KY_DATATYPE_BOOL8                     = 11
    KY_DATATYPE_SIZET                     = 12
    KY_DATATYPE_BUFFER                    = 13
```

```
class KY_STREAM_INFO_CMD:
    KY_STREAM_INFO_PAYLOAD_SIZE                              =  7
    KY_STREAM_INFO_BUF_ALIGNMENT                             = 13
    KY_STREAM_INFO_PAYLOAD_SIZE_INCREMENT_FACTOR             = 1000
    KY_STREAM_INFO_BUF_COUNT                                 = 2000
```

```
class KYFG_FRAME_AUX_DATA_RAW:
    sequence_number           = 0
    timestamp                 = 0
    reserved                  = 0
```

20.2 Camera Configurations

20.2.1 KYFG_CameraScan()

The Frame Grabber scans for connected cameras, establishes connection and defines the default speed for each camera, on every connected channel.

```
def KYFG_CameraScan( handle)
```

Parameter name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber

Return value

camHandleList - List of API camera handles of detected cameras. Type: List

FGSTATUS - Status and error report.

FGSTATUS_EXCEEDED_MAX_CAMERA_CONNECTIONS - value will indicate that number of connected cameras exceeds the maximum allowed connected cameras.

20.2.2 KYFG_CameraOpen2()

Opens a connection to chosen camera, retrieves native XML file or uses external XML file provided to override the native one.

```
def  KYFG_CameraOpen2( camHandle, xml_file_path)
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to connected camera
xml_file_path	str	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. ***Please see remarks!***

Return value

FGSTATUS - Status and error report.

Remarks

An XML file can be loaded to override the native XML of the camera. Otherwise *None* should be passed in order to retrieve camera’s native XML file.

20.2.1 KYFG_CameraInfo()

Retrieves current information about the chosen camera. The camera info includes general device information and connectivity topology. This function can be called before KYFG_CameraOpen2().

```
def KYFG_CameraInfo( camHandle, cam_info):
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to connected camera

Return value

FGSTATUS - Status and error report.

cam_info - Chosen camera information. Type: KYFGCAMERA_INFO

Example Code

```
(Status, camInfo) = KYFG_CameraInfo(camHandle)
print("master_link: ", str(camInfo.master_link))
print("link_mask: ", str(camInfo.link_mask))
print("link_speed: ", str(camInfo.link_speed))
print("stream_id: ", str(camInfo.stream_id))
print("deviceVersion: ", str(camInfo.deviceVersion))
print("deviceVendorName: ", str(camInfo.deviceVendorName))
print("deviceManufacturerInfo: ", str(camInfo.deviceManufacturerInfo))
print("deviceModelName: ", str(camInfo.deviceModelName))
print("deviceID: ", str(camInfo.deviceID))
print("deviceUserID: ", str(camInfo.deviceUserID))
print("outputCamera: ", str(camInfo.outputCamera))
print("virtualCamera: ", str(camInfo.virtualCamera))
```

20.2.1 KYFG_CameraGetXML()

Extracts native XML file from chosen camera and fills user allocated buffer. The size (in bytes) and file type (.xml or .zip) are also retrieved even if buffer isn’t large enough to hold all file data.

```
def  KYFG_CameraGetXML( camHandle )
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera

Return value

FGSTATUS - Status and error report.

isZipFile - Indicator whether the camera’s XML file is in ZIP or XML format. Type: *KYBOOL* buffer – Bytearray, that includes xml string or zip binary file

Example code

```
(KYFG_CameraGetXML_status, isZipped, buffer) =
KYFG_CameraGetXML(camHandleArray[grabberIndex][0])
print("Is Zipped: " + str(isZipped.get()))
print("KYFG_CameraGetXML_status: " + str(format(KYFG_CameraGetXML_status, '02x')))
if (isZipped == False):
    print("Writing buffer to xml file...")
    newFile = open("camera_xml.xml","w")
    newFile.write(''.join(buffer))
    newFile.close()
else:
    print("Writing buffer to zip file...")
    newFile = open("camera_xml.zip","wb")
    newFile.write(bytes(buffer))
    newFile.close()
```

20.3 Callback functions

20.3.1 KYFG_CallbackRegister()

Register a general runtime acquisition callback function. The callback (userFunc) will be called upon each new received frame of a valid stream, with appropriate BUFFHANDLE. Callback call is not necessarily serialized, which means different streams might generate concurrent calls before end of previous callback execution.

```
def  KYFG_CallbackRegister( handle,  userFunc,  userContext )
```

Parameter name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber
userFunc	Func Name	Name of callback function. See remarks
userContext	int – currently 0	(optional) User context. Afterwards this value is retrieved when the callback is issued. Helps to determine the origin of stream in host application.

Return value

FGSTATUS - Status and error report.

Remarks

The callback function should be defined by 2 parameters:

def Stream_callback_func(buffHandle, userContext):

buffHandle – is a stream handle.

userContext – is a user context

Example Code

```
def Stream_callback_func(buffHandle, userContext):
    totalFrames = 0
    buffSize = 0
    buffIndex = 0
    buffData = 0

    if (buffHandle == 0 ):
        Stream_callback_func.copyingDataFlag = 0
        return
    (status, totalFrames) = KYFG_GetGrabberValueInt(buffHandle, "RXFrameCounter")
    (buffSize,) = KYFG_StreamGetSize(buffHandle)
    (status, buffIndex)= KYFG_StreamGetFrameIndex(buffHandle)
    (buffData,) = KYFG_StreamGetPtr(buffHandle, buffIndex)
    if ( Stream_callback_func.copyingDataFlag == 0):
        Stream_callback_func.copyingDataFlag = 1

    print('Good callback buffer handle: ' + str(format(buffHandle, '02x')) + ", current index: " +
str(buffIndex) + ", total frames: " + str(totalFrames) + "          ", end='\r')
    sys.stdout.flush()

    Stream_callback_func.copyingDataFlag = 0
    return

Stream_callback_func.data = 0
Stream_callback_func.copyingDataFlag = 0
```

20.3.1 KYFG_StreamBufferCallbackRegister()

Register a stream runtime acquisition callback function. The callback (userFunc) will be called upon new received frame, of a valid stream, with appropriate STREAM_BUFFER_HANDLE. Each stream’s callback is serialized and will be held until end of callback execution. The different stream callbacks are working concurrently. Use the Stream interface functions to handle received data. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

```
def  KYFG_StreamBufferCallbackRegister (streamHandle, userFunc, userContext):
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle of a stream

userFunc		Pointer to callback function
userContext		User context. Afterwards this pointer is retrieved when the callback is issued. Helps to determine the origin of stream in host application.

Return value

FGSTATUS - Status and error report.

20.4 Camera/Frame Grabber Values

20.4.1 KYFG_GetCameraValue()

Get camera configuration field value.

```
def  KYFG_GetCameraValue (camHandle, paramName):
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to chosen camera
paramName	Str	Name of configuration parameter

Return value

FGSTATUS - Status and error report.

paramVal– The value of the required parameter. Type: According to a request

Remarks

In case of PROPERTY_TYPE_ENUM, the tuple includes 3 elements: status, paramValueStr and paramValueInt, where paramValueStr and paramValueInt represent the required enum entry.

20.4.2 KYFG_GetGrabberValue()

Get Frame Grabber configuration field value.

```
def  KYFG_GetGrabberValue (handle, paramName):
```

Parameter name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber
paramName	Str	Name of configuration parameter

Return value

FGSTATUS - Status and error report.

paramVal - The value of the required parameter. Type: According to a request

Remarks

In case of PROPERTY_TYPE_ENUM, the tuple includes 3 elements: status, paramValueStr and paramValueInt, where paramValueStr and paramValueInt represent the required enum entry.

20.4.3 KYFG_SetCameraValue()

Set camera configuration value of Integer type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
def  KYFG_SetCameraValue( camHandle,  paramName, paramValue)
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to chosen camera
paramName	str	Name of configuration parameter
paramValue	According to parameter type	The value of the parameter to set

Return value

Status – FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

20.4.4 KYFG_SetGrabberValue()

Set camera configuration value of Enumeration type field. According to Gen<i>Cam standard naming and xml field definition and type.

```
def  KYFG_SetGrabberValue (handle,  paramName, paramValue)
```

Parameter name	Type	Description
handle	FGHANDLE or int	API handle to chosen frame grabber
paramName	str	Name of configuration parameter
paramValue	According to parameter type	The value of the parameter to set

Return value

Status – FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

20.4.5 KYFG_CameraExecuteCommand()

Execute camera command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

```
def  KYFG_CameraExecuteCommand (camHandle,  paramName)
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to chosen Camera
paramName	str	Name of command

Return value

Status – FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

20.4.6 KYFG_GrabberExecuteCommand()

Execute camera command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

```
def  KYFG_GrabberExecuteCommand (handle,  paramName)
```

Parameter name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber
paramName	str	Name of command

Return value

Status – FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

20.5 Python API

20.5.1 KYFG_StreamGetFrameIndex()

Retrieves the index of the last acquired frame acquired from specified stream.

```
def  KYFG_StreamGetFrameIndex( streamHandle )
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle to a stream

Return value

Status - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

StreamGetSize - Index of the last acquired frame from specified stream. Type: int

In case of an error (FGSTATUS_OK, -1) will be returned

20.5.2 KYFG_StreamCreateAndAlloc()

A new stream will be allocated for specified camera. The created stream buffers will hold the data of acquired frames. Stream buffer acquisition mechanism and buffer size calculations are handled internally. Buffer frame size is calculated with consideration of specified number of frames, in addition to camera and grabber configuration parameters set previously to this function call. Changing certain camera/grabber parameters, after successfully stream allocation, might result in unstable software operation, memory leaks and even total system crash.

```
def  KYFG_StreamCreateAndAlloc ( camHandle, pStreamHandle, frames, streamIndex);
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to connected camera
pStreamHandle	STREAM_HANDLE	Output parameter - pointer to STREAM_HANDLE variable that will hold handle of newly created stream
frames	int	Number of frames that should be allocated for this stream.
streamIndex	int	Index of stream. Currently unused and must be 0.

Return value

FGSTATUS - Status and error report.

20.5.1 KYFG_StreamCreate()

A new stream will be created for the chosen camera. The stream will manage frame buffers allocated either by user or by library. Frame buffers will be organized in queues – input, output, automatic – and in a set of unqueued frame buffers.

```
def  KYFG_StreamCreate( camHandle, streamIndex )
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to connected camera
streamIndex	int	Index of stream. Currently unused and must be 0.

Return value

FGSTATUS - Status and error report.

pStreamHandle - handle of newly created stream. Type: *STREAM_HANDLE*

20.5.2 KYFG_StreamGetInfo()

Retrieves information about specified stream.

```
def  KYFG_StreamGetInfo (streamHandle, cmdStreamInfo):
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle of a stream
cmdStreamInfo	KY_STREAM_INFO_CMD	Specifies what information is being requested. Possible values are: • KY_STREAM_INFO_PAYLOAD_SIZE – The function will return size of memory required for single frame buffer. ‘pInfoBuffer’ must be NULL or point to size_t variable. • KY_STREAM_INFO_BUF_ALIGNMENT – The function will return required alignment of memory allocated for a buffer. ‘pInfoBuffer’ must be NULL or point to size_t variable.

Return value

FGSTATUS - Status and error report.

pInfoBuffer - user variable that will be filled with required information. Can be NULL.

pInfoSize - Pointer to size of provided pInfoBuffer. Can be NULL.

In: size of the provided pInfoBuffer in bytes.

Out: minimum required size of pInfoBuffer to hold requested information.

pInfoType - Pointer to data type of pInfoBuffer content. Can be NULL.

Out: data type of pInfoBuffer for requested information.

20.5.3 KYFG_BufferAnnounce()

This function is used to announce a buffer allocated by user and bind it to a stream.

The memory size should correspond to a single acquisition frame. This size can be retrieved using function KYFG_StreamGetInfo() with KY_STREAM_INFO_PAYLOAD_SIZE info command. Also, any virtual memory allocated by user should be aligned to the value retrieved using function KYFG_StreamGetInfo() with KY_STREAM_INFO_BUF_ALIGNMENT info command.

The user remains the owner of memory – the memory will NOT be freed by library and MUST stay valid until stream is deleted.

Initially, the buffer will be added to the set of unqueued buffers of that stream. To make the buffer available for incoming data, user should add it to incoming queue using function KYFG_BufferToQueue(). Alternatively, function KYFG_BufferQueueAll() can be used after all desired user buffers are announced.

def KYFG_BufferAnnounce (streamHandle, pBuffer, pPrivate):

Parameter name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle of a stream
pBuffer		Input parameter - pointer to memory allocated by user
pPrivate	int	This parameter is currently ignored

Return value

FGSTATUS - Status and error report.

pBufferHandle - variable that will hold handle of newly announced frame buffer.

Type: STREAM_BUFFER_HANDLE

20.5.4 KYFG_BufferGetInfo()

Retrieves information about previously announced buffer.

def KYFG_BufferGetInfo (streamBufferHandle, cmdStreamBufferInfo):

Parameter name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE or int	Handle of a stream buffer
cmdStreamBufferInfo	int	One of the values stored in KY_STREAM_BUFFER_INFO_CMD Specifies what information is being requested. Possible values are: • KY_STREAM_BUFFER_INFO_BASE. The function will return Base address of the buffer memory. ‘pInfoBuffer’ must be NULL or point to a pointer variable. • KY_STREAM_BUFFER_INFO_SIZE – reserved for future enhancements

		• KY_STREAM_BUFFER_INFO_USER_PTR – reserved for future enhancements • KY_STREAM_BUFFER_INFO_TIMESTAMP – reserved for future enhancements • KY_STREAM_BUFFER_INFO_INSTANTFPS - instant FPS calculated from this and previous timestamp • KY_STREAM_BUFFER_INFO_ID - unique id of buffer in the stream

Return value

FGSTATUS - Status and error report.

pInfoBuffer - User variable that will be filled with required information.

pInfoSize - Size of provided pInfoBuffer.

In: size of the provided pInfoBuffer in bytes.

Out: minimum required size of pInfoBuffer to hold requested information.

pInfoType - Data type of pInfoBuffer content.

Out: data type of pInfoBuffer for requested information.

20.5.5 KYFG_BufferToQueue()

Moves a previously announced buffer to specified queue.

```
def  KYFG_BufferToQueue (streamBufferHandle, dstQueue):
```

Parameter name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE or int	Handle of a stream buffer
cmdStreamBufferInfo	int	One of the values stored in KY_ACQ_QUEUE_TYPE Destination queue: • KY_ACQ_QUEUE_INPUT – buffers in this queue are ready to be filled with data. • KY_ACQ_QUEUE_OUTPUT - buffers in this queue have been filled and awaiting user processing. This queue is filled

		internally by library. An ability for application to move buffers to this queue is reserved for future library enhancements and currently will result in error code FGSTATUS_DESTINATION_QUEUE_NOT_SUPPORTED • KY_ACQ_QUEUE_UNQUEUED – reserved for future enhancements • KY_ACQ_QUEUE_AUTO – reserved for future enhancements

Return value

FGSTATUS - Status and error report.

20.5.6 KYFG_BufferQueueAll()

Moves all frame buffers bound to specified stream from one queue to another queue.

```
def  KYFG_BufferQueueAll (streamHandle, srcQueue, dstQueue):
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE or int	Handle of a stream
srcQueue	int	One of values stored in KY_ACQ_QUEUE_TYPE Source queue. See KYFG_BufferToQueue() description for possible values
dstQueue	int	One of values stored in KY_ACQ_QUEUE_TYPE Destination queue. See KYFG_BufferToQueue() description for possible values

Return value

FGSTATUS - Status and error report.

20.6 Data acquisition

20.6.1 KYFG_CameraStart()

Starts transmission for the chosen camera. The chosen stream would be filled with data from the camera. Only 1 stream can be active at a time, per camera. Number of frames to be acquired may be set, while 0 frames indicate continues acquisition mode.

```
def  KYFG_CameraStart(camHandle, streamHandle, frames);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
streamHandle	STREAM_HANDLE	API handle to data stream for selected camera
frames	Int	Number of frames to be acquired. After the specified number of frames were acquired, the camera would be stopped. 0 for continues acquisition mode.

Return value

FGSTATUS - Status and error report.

20.6.2 KYFG_CameraStop()

Stops transmission for the chosen camera.

```
def  KYFG_CameraStop( camHandle )
```

Parameter name	Type	Description
camHandle	CAMHANDLE or int	API handle to connected camera

Return value

FGSTATUS - Status and error report.

20.6.3 KYFG_StreamGetPtr()

Retrieves a pointer to data memory space of 1 frame in the chosen buffer.

```
def KYFG_StreamGetPtr (streamHandle, frame);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream
frame	int	Frame index of data pointer to be retrieved

Return value

FramePtr - Pointer to data of specified frame. NULL will be retrieved if frame index is out of range or other operation failure.

21.1 public ref class Lib

21.1.1 Lib()

Standard constructor.

21.1.2 int Lib::Scan();

Scans for KAYA devices currently connected to the PC PCIe slots and available virtual devices and optionally fills array with devise IDs.

```
int Lib.Scan();
```

Return value

Returns the number of connected hardware and virtual devices.

21.1.3 System::String Lib.DeviceDisplayName() (DEPRECATED)

Retrieve device name for the specified index.

```
System::String Lib.DeviceDisplayName(int dev_id);
```

Parameter name	Type	Description
dev_id	int	Discovered device index

Return value

The name of Frame Grabber issued by the specified index.

21.1.4 DEVICE_INFO Lib.DeviceInfo ()

Returnes info about the relevant device.

```
DEVICE_INFO DeviceInfo (int index);
```

Parameter name	Type	Description
index	int	Discovered device index

Return value

DEVICE_INFO – Info about the device

```
public ref class DEVICE_INFO
```

```
{
        System::String          DeviceName;
        System::Int32           Bus;
        System::Int32           Slot;
        System::Int32           Function;
        System::UInt32          DevicePID;
        System::Boolean         isVirtual;
}
```

21.1.5 IGrabber Lib.Open();

Connect to a specific Frame Grabber and initializes all required components.

```
IGrabber Lib.Open( int index);
```

Parameter name	Type	Description
index	int	The index, from scan result array acquired with Scan() function, of the Frame Grabber device to open.

Return value

Instance of Grabber class, that implements IGrabber interface.

21.1.6 IGrabber Lib.OpenEx()

Connect to a specific Frame Grabber and initializes all required components. Project file may be passed here in order to initialize Frame Grabber and Camera parameters with previously saved values.

```
IGrabber Lib.OpenEx(int index, System.String projectFile);
```

Overloads:

```
IGrabber Lib.OpenEx(int index);
```

Parameter name	Type	Description
index	int	The index, from scan result array acquired with Lib.Scan() function, of the Frame Grabber device to open.
projectFile	String	(optional) Full path of a project file with saved values. Input value can be NULL.

21.2 public interface class IDevice

21.2.1 System.Collections.Generic.List<ICamera> IDevice.CameraScan();

The Frame Grabber scans for connected cameras, establishes connection and defines the default speed for each camera, on every connected channel.

Return value

List of all found cameras connected to the current Grabber device. Number of cameras could be retrieved by getting size of the list.

21.2.2 void IDevice.Close()

Close the current Frame Grabber. Stops data acquisition of all opened cameras, disconnects from all connected cameras and deletes previously created buffers associated with these cameras.

21.2.3 void IDevice.SetValue(System.String paramName, System.Object paramValue)

Set camera/Frame Grabber configuration field value.

Parameter name	Type	Description
paramName	String	Name of configuration parameter
paramValue	int, Boolean, String, Double	Object, that represents a param value corresponding to the param name.

21.2.4 System.Object IDevice.GetValue(System.String paramName);

Get Frame Grabber configuration field value.

Parameter name	Type	Description
paramName	String	Name of configuration parameter

Return value

Object, that represents a param value corresponding to the param name. Could be one of the following types: Int32, Boolean, String, Double or Tuple (see remarks).

Remark

In case the requested parameter of ENUM type, the returned Object will be an instance of:

```
System.Tuple<System.String, System.Int64>
```

Where the first parameter will represent the name of the ENUM, and the second will represent the enumeration value of the ENUM

21.2.5 void IDevice.ExecuteCommand(System.String paramName)

Execute Frame Grabber command.

Parameter name	Type	Description
paramName	String	Name of configuration parameter

21.2.6 void IDevice.WritePortBlock(int port, System.UInt64 address, array<System.Byte> buffer)

Write buffer of specified size to specific port. This function access the link directly disregarding the camera connection topology.

Parameter name	Type	Description
port	int	Frame Grabber port index
address	System.UInt64	Start address of the data to write
buffer	array<System.Byte>	Buffer data to write

21.2.7 void IDevice.WritePortReg(int port, System.UInt64 address, System.UInt32 data)

Write bootstrap registers from specific port, 32bit value each time. This function access the link directly disregarding the camera connection topology.

Parameter name	Type	Description
port	int	Frame Grabber port index
address	System.UInt64	Start address of the data to write
data	System.UInt32	Bootstrap registers value

21.2.8 array<System.Byte> IDevice.ReadPortBlock(int port, System.UInt64 address, System.UInt32 size)

Read buffer of specified size from specific port. This function access the link directly disregarding the camera connection topology.

Parameter name	Type	Description
port	int	Frame Grabber port index
address	System.UInt64	Start address of the data to read
size	System.UInt32	Size in bytes of buffer to read

Return value

The received data from the port.

21.2.9 System::UInt32 IDevice.ReadPortReg(int port, System.UInt64 address)

Read bootstrap registers from specific port, 32bit value each time. This function access the link directly disregarding the camera connection topology.

Parameter name	Type	Description
port	int	Frame Grabber port index
address	System.UInt64	Address of the register

Return value

The received data from the register.

21.2.10 void IDevice.AuthProgramKey(array<System.Byte> key, int lock)

Program provided key to the grabber.

Parameter name	Type	Description
key	array<System.Byte>	A key to be programmed into Frame Grabber
lock	int	If this parameter is 0 the grabber can be re-programmed with a different key later. If this parameter is 1 then provided key is locked in the Frame Grabber and following call of this function will fail.

21.2.11 int IDevice.AuthVerify(array<System.Byte> key)

Verify provided key against one already programmed to the grabber.

Parameter name	Type	Description
key	array<System.Byte>	A key to be verified with Frame Grabber

Return value

1 – if the key accepted, 0 – otherwise.

21.2.12 void IDevice.AuxDataCallbackRegister(FGAuxDataCallback delegator, Object userContext)

Overloads:

void IGrabber.AuxDataCallbackRegister(FGAuxDataCallback delegator)

Register run-time callback for receiving auxiliary data. The callback will be called when various auxiliary data is generated.

Parameter name	Type	Description
delegator	FGAuxDataCallback	Callback delegate function
userContext	Object	(optional) User context. Afterwards this pointer is retrieved when the callback is issued. Helps to determine the origin of function call in host application.

21.2.13 void IDevice.AuxDataCallbackUnregister(FGAuxDataCallback delegator)

Unregister run-time auxiliary data callback.

Parameter name	Type	Description
delegator	FGAuxDataCallback	Callback delegate function

21.3 public interface class ICamera

21.3.1 CAMERA_INFO^ KYFG_CameraInfo()

Retrieves current information about the chosen camera. The camera info includes general device information and connectivity topology.

```
CAMERA_INFO KYFG_CameraInfo();
```

Return value

CAMERA_INFO - Info about the camera

```
public ref class CAMERA_INFO
{
        const System::Byte^          master_link;
        const System::Byte^          link_mask;
        System::Int32^               link_speed;
        System::UInt32^              stream_id;
        System::String^              deviceVersion;
        System::String^              deviceVendorName;
        System::String^              deviceManufacturerInfo;
        System::String^              deviceModelName;
        System::String^              deviceID;
        System::String^              deviceUserID;
        System::Boolean^             outputCamera;
        System::Boolean^             virtualCamera;
}
```

21.3.2 void ICamera.Open(System.String xml_file_path)

Opens a connection to chosen camera, retrieves native XML file or uses external XML file provided to override the native one.

Parameter name	Type	Description
xml_file_path	System.String	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. **Please see remarks!**

Remarks

An XML file can be loaded to override the native XML of the camera. Otherwise NULL should be passed in order to retrieve camera's native XML file.

21.3.3 void ICamera.Close()

Close a connection to the selected camera. Stops data acquisition and deletes previously created buffers associated with the camera. The connection information is preserved, so a new connection can be established later.

21.3.4 void ICamera.Start(IStream streamHandle, int frames)

Starts transmission for the chosen camera. The chosen stream would be filled with data from the camera. Only 1 stream can be active at a time, per camera. Number of frames to be acquired may be set, while 0 frames indicate continues acquisition mode.

Parameter name	Type	Description
streamHandle	IStream	API handle to data stream for selected camera
frames	int	Number of frames to be acquired. After the specified number of frames were acquired, the camera would be stopped. 0 for continues acquisition mode.

21.3.5 void ICamera.Stop()

Stops transmission for the chosen camera.

21.3.6 void ICamera.SetValue(System.String paramName, Object paramValue)

Set camera configuration field value. According to Gen<i>Cam standard naming and xml field definition and type.

Parameter name	Type	Description
paramName	System::String	Name of configuration parameter
param	Object, that represents a param value corresponding to the param name. Can be int, Boolean, String, Float	Camera configuration value

21.3.7 Object ICamera::GetValue(System.String paramName)

Get camera configuration field value.

Parameter name	Type	Description
paramName	System.String	Name of configuration parameter

Return value

Object, that represents a param value corresponding to the param name. Can be int, Boolean, String, Float, String or Tuple (see remark)

Remark

In case the requested parameter of ENUM type, the returned Object will be the instance of:

```
System.Tuple<System.String, System.Int64>
```

Where the first parameter will represent the name of the ENUM, and the second will represent the enumeration value of the ENUM

21.3.8 void ICamera.ExecuteCommand(System.String paramName)

Execute camera command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

Parameter name	Type	Description
paramName	System.String	Name of configuration parameter

21.3.9 IStream ICamera.StreamCreateAndAlloc(int frames)

A new stream will be allocated for specified camera. The created stream buffers will hold the data of acquired frames. Stream buffer acquisition mechanism and buffer size calculations are handled internally. Buffer frame size is calculated with consideration of specified number of frames, in addition to camera and grabber configuration parameters set previously to this function call.

Changing certain camera/grabber parameters, after successfully stream allocation, might result in unstable software operation, memory leaks and even total system crash.

Parameter name	Type	Description
frames	int	Number of frames that should be allocated for this stream.

Return value

IStream handle of newly created stream.

21.3.10 IStream ICamera.StreamCreate()

A new stream will be created for the chosen camera. The stream will manage frame buffers allocated either by user or by library. Frame buffers will be organized in queues – input, output, automatic – and in a set of unqueued frame buffers.

Return value

IStream handle of newly created stream.

21.3.11 System.Tuple<byte[], KYBOOL> ICamera.GetXML()

Extracts native XML file from chosen camera and fills user allocated buffer. The size (in bytes) and file type (.xml or .zip) are also retrieved even if buffer isn't large enough to hold all file data.

Return value

Tuple(xml_managed_string, isZip_managed), where:

- xml_managed_string - Byte array, which contains the required XML
- isZip_managed - KY_TRUE if the XML archived within Zip, KY_FALSE otherwise

Example

```
Tuple<byte[], KAYA.KYBOOL> xml_string_tuple = camera.GetXML();
if (xml_string_tuple.Item2 == KAYA.KYBOOL.KY_TRUE)
   {
      System.IO.File.WriteAllBytes("C:\\Users\\PC-01\\Desktop\\cameras_xml.zip",
                                          xml_string_tuple.Item1);
   }
   else
   {
            System.IO.File.WriteAllBytes("C:\\Users\\PC-01\\Desktop\\ cameras_xml.xml",
                                                    xml_string_tuple.Item1);
```

```
        }
```

21.3.12 System.UInt32 ICamera.WriteReg(System.UInt64 address, array<System.Byte> buffer)

Direct write data buffer to the selected camera.

Parameter name	Type	Description
address	System::UInt64	Start address of the data to write
buffer	array<System::Byte>	Buffer data to write

Return value

Size of written bytes.

21.3.13 array<System.Byte> ICamera.ReadReg(System.UInt64 address, System.UInt32 size)

Direct read data buffer from the selected camera.

Parameter name	Type	Description
address	System::UInt64	Start address of the data to write
buffer	System::UInt32	Buffer data to write

Return value

Buffer that holds read data

21.3.14 void ICamera.CameraCallbackRegister(CameraCallback delegator, Object userContext)

Overloads:

```
void ICamera.CameraCallbackRegister(CameraCallback delegator))
```

Register a camera runtime acquisition callback function.

Parameter name	Type	Description
delegator	CameraCallback	Delegator to callback function
userContext	Object	User defined context to identify the received callback

21.3.15 void ICamera.CameraCallbackUnregister(CameraCallback delegator)

Unregister a camera runtime acquisition callback function.

Parameter name	Type	Description
delegator	CameraCallback	Delegator to callback function

21.4 public interface class IStream

21.4.1 void IStream.BufferCallbackRegister(StreamBufferCallback^ delegator, Object^ userContext)

Overloads:

void IStream.BufferCallbackRegister(StreamBufferCallback^ delegator)

Register a stream runtime acquisition callback function. The callback (userFunc) will be called upon new received frame, of a valid stream.

Parameter name	Type	Description
delegator	StreamBufferCallback	Delegator to callback function
userContext	Object	User defined context to identify the received callback

21.4.2 void IStream.BufferCallbackUnregister(StreamBufferCallback^ delegator)

Unregister a camera runtime acquisition callback function.

Parameter name	Type	Description
delegator	StreamBufferCallback	Delegator to callback function

21.4.3 Object^ IStream.GetPtr(int buffIndex)

Retrieves a pointer to data memory space of 1 frame in the chosen buffer.

Parameter name	Type	Description
buffIndex	System::UInt32	Frame index of data pointer to be retrieved

Return value

Pointer to required buffer.

21.4.4 void IStream.Delete()

Deletes a stream. Any memory allocated by user is NOT freed by this function. All memory allocated by library is freed and all API handles bound to the stream became invalid.

21.4.5 Object IStream.GetInfo(KY_STREAM_INFO_CMD info)

Retrieves information about specified stream.

Parameter name	Type	Description
info	KY_STREAM_INFO_CMD	Specifies what information is being requested. Possible values are: KY_STREAM_INFO_PAYLOAD_SIZE KY_STREAM_INFO_BUF_ALIGNMENT KY_STREAM_INFO_PAYLOAD_SIZE_INCREMENT_FACTOR KY_STREAM_INFO_BUF_COUNT

Return value

The required info.

21.4.6 AuxData IStream.GetAux(int frame)

Retrieves Auxiliary data of specified frame.

Parameter name	Type	Description
frame	int	Frame index to be retrieved

Return value

AuxData of specified frame.

21.4.7 IStreamBuffer IStream.BufferAllocAndAnnounce(System.UInt64 nBufferSize)

This function is used to allocate and announce a buffer and bind it to a stream.

Parameter name	Type	Description
nBufferSize	System::UInt64	The size of allocated memory

Return value

Allocated Stream Buffer.

21.4.8 IStreamBuffer IStream.BufferAnnounce(array<System.Byte> pBuffer)

This function is used to announce a buffer allocated by user and bind it to a stream.

Parameter name	Type	Description
pBuffer	array<System.Byte>	User allocated byte array, which will be used to hold frame data

Return value

Stream Buffer.

21.4.9 void IStream.BufferQueueAll(KY_ACQ_QUEUE_TYPE srcQueue, KY_ACQ_QUEUE_TYPE dstQueue)

Moves all frame buffers bound to specified stream from one queue to another queue.

Parameter name	Type	Description
srcQueue	KY_ACQ_QUEUE_TYPE	Source queue
dstQueue	KY_ACQ_QUEUE_TYPE	Destination queue

21.5 public interface class IStreamBuffer

21.5.1 int IStreamBuffer.GetFrameIndex();

Retrieves the index of the current buffer (frame).

Return value

Index of the current buffer.

21.5.2 Object IStreamBuffer.GetPtr();

Retrieves a pointer to data memory space of the current buffer (frame).

Return value

Pointer to current buffer (frame).

21.5.3 Object IStreamBuffer.GetInfo(KY_STREAM_BUFFER_INFO_CMD info);

Retrieves information about previously announced buffer (frame).

Parameter name	Type	Description
info	KY_STREAM_BUFFER_INFO_CMD	Specifies what information is being requested. Possible values are: KY_STREAM_BUFFER_INFO_BASE: The function will return Base address of the buffer memory. KY_STREAM_BUFFER_INFO_SIZE:

		Size of the buffer KY_STREAM_BUFFER_INFO_USER_PTR Pointer to the buffer KY_STREAM_BUFFER_INFO_TIMESTAMP Buffer timestamp KY_STREAM_BUFFER_INFO_ID Unique ID of buffer in the stream

Return value

The relevant info regarding the buffer (frame).

```
public enum class KY_STREAM_BUFFER_INFO_CMD
{
        KY_STREAM_BUFFER_INFO_BASE =::KY_STREAM_BUFFER_INFO_BASE,
        KY_STREAM_BUFFER_INFO_SIZE =::KY_STREAM_BUFFER_INFO_SIZE,
        KY_STREAM_BUFFER_INFO_USER_PTR
        =::KY_STREAM_BUFFER_INFO_USER_PTR,
        KY_STREAM_BUFFER_INFO_TIMESTAMP=::KY_STREAM_BUFFER_INFO_TIMES
        AMP,
        KY_STREAM_BUFFER_INFO_ID = ::KY_STREAM_BUFFER_INFO_ID
};
```

21.5.4 System.UInt64 IStreamBuffer.getSize()

Retrieves the size of the current buffer (frame).

Return value

The size of the current buffer.

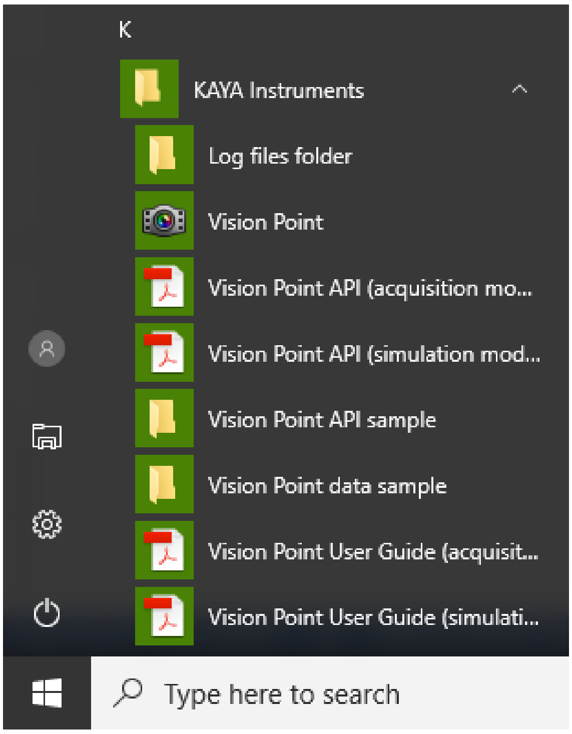
21.5.5 void IStreamBuffer.BufferToQueue(KY_ACQ_QUEUE_TYPE dstQueue)

Moves a previously announced buffer to specified queue.

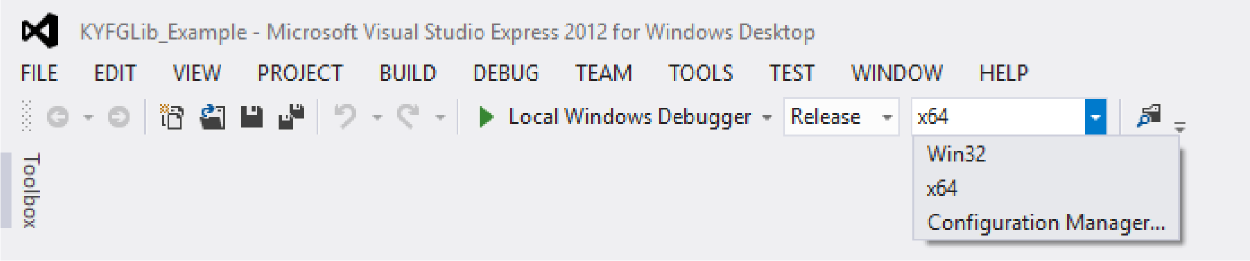
Parameter name	Type	Description
dstQueue	KY_ACQ_QUEUE_TYPE	Destination queue: KY_ACQ_QUEUE_INPUT, KY_ACQ_QUEUE_OUTPUT, KY_ACQ_QUEUE_UNQUEUED, KY_ACQ_QUEUE_AUTO

22.1 API example for Windows

1. Open an API example project KYFGLib_Example.vcxproj for Microsoft Visual Studio, provided in the download directory. The "API Samples" directory can be easily found using the quick search, as shown in the image below.


K
KAYA Instruments
Log files folder
Vision Point
Vision Point API (acquisition mo...
Vision Point API (simulation mod...
Vision Point API sample
Vision Point data sample
Vision Point User Guide (acquisit...
Vision Point User Guide (simulati...
Type here to search

2. Choose solution platform according to your operation system, as shown in the image below.
   Note: The Vision Point software stack does not support Win32 platform with OS x64.

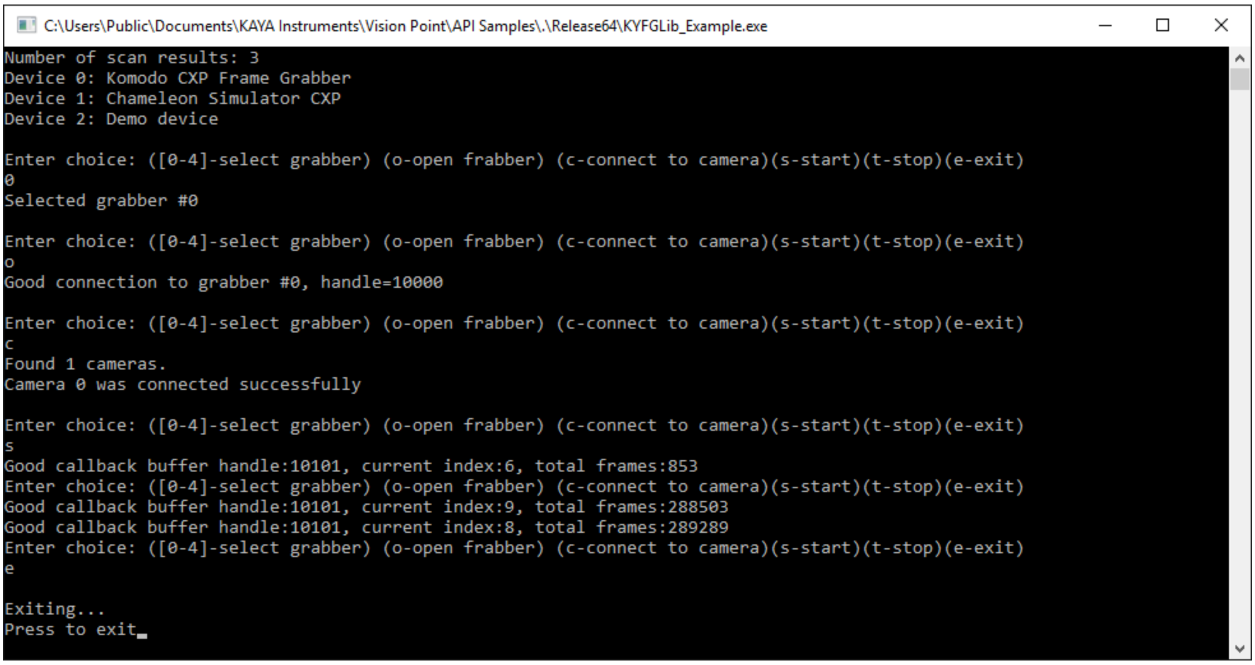


3. Build a solution.
4. Run the application.
5. Enter a device, or Demo mode, from the list.

6. Enter a command. The following table describes the commands options.

Command	Description
[0-4]	Device selection
o	Open Frame Grabber
c	Connect to camera
s	Start the frame acquisition
t	Stop the frame acquisition
e	Exit the Example

An example of this operation is shown in the image below.



C:\Users\Public\Documents\KAYA Instruments\Vision Point\API Samples\.\Release64\KYFGLib_Example.exe

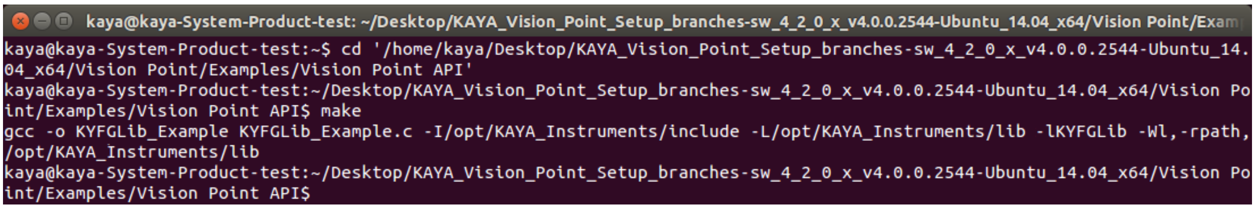
```
Number of scan results: 3
Device 0: Komodo CXP Frame Grabber
Device 1: Chameleon Simulator CXP
Device 2: Demo device

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
0
Selected grabber #0

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
o
Good connection to grabber #0, handle=10000

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
c
Found 1 cameras.
Camera 0 was connected successfully

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
s
Good callback buffer handle:10101, current index:6, total frames:853
Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
Good callback buffer handle:10101, current index:9, total frames:288503
Good callback buffer handle:10101, current index:8, total frames:289289
Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
e

Exiting...
Press to exit_
```

NOTE:

By default, the KYFGLib_Example.vcxproj project contains KYFGLib_Example.c which uses **Cyclic Buffer**. In order to use **Queued Buffer**, in the loaded project, change the KYFGLib_Example.c file to KYFGLib_Example_QueuedBuffers.c located in the same directory, and rebuild example.

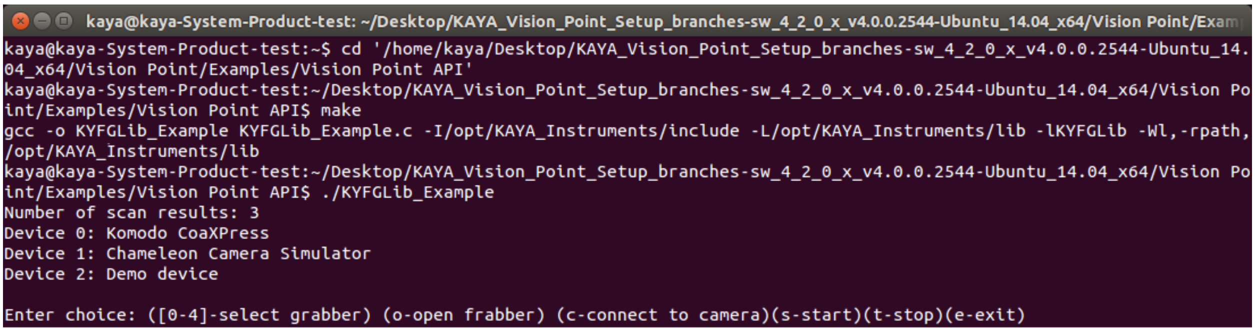
22.2 API example for Linux

1. Open the Terminal and enter the directory path of the API Example. The API Example is stored under Vision Point/Examples/Vision Point API directory.
2. Type “make” and make sure the KYFGLib_Example executable file was created, in the same directory. The image below shows the 1st and the 2nd step.


kaya@kaya-System-Product-test: ~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Point/Exam

```
kaya@kaya-System-Product-test:~$ cd '/home/kaya/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.
04_x64/Vision Point/Examples/Vision Point API'
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ make
gcc -o KYFGLib_Example KYFGLib_Example.c -I/opt/KAYA_Instruments/include -L/opt/KAYA_Instruments/lib -lKYFGLib -Wl,-rpath,
/opt/KAYA_Instruments/lib
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$
```

3. To run the API Example, simply type “./KYFGLib _Example” followed by “Enter”, as shown in the image below.

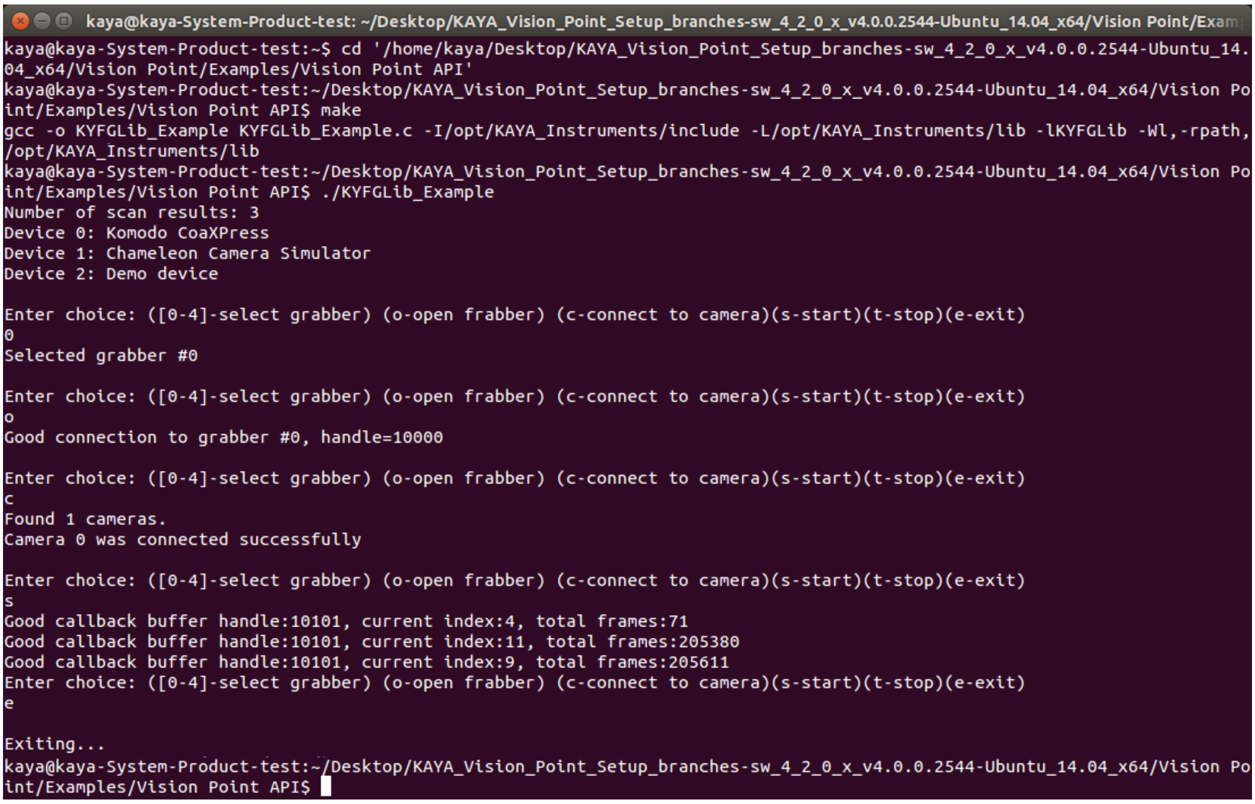

kaya@kaya-System-Product-test: ~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Point/Exam

```
kaya@kaya-System-Product-test:~$ cd '/home/kaya/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.
04_x64/Vision Point/Examples/Vision Point API'
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ make
gcc -o KYFGLib_Example KYFGLib_Example.c -I/opt/KAYA_Instruments/include -L/opt/KAYA_Instruments/lib -lKYFGLib -Wl,-rpath,
/opt/KAYA_Instruments/lib
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ ./KYFGLib_Example
Number of scan results: 3
Device 0: Komodo CoaXPress
Device 1: Chameleon Camera Simulator
Device 2: Demo device

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
```

4. Enter a device, or Demo mode, from the list.
5. Enter a command. The following table describes the command options.

Command	Description
[0-4]	Device selection
o	Open Frame Grabber
c	Connect to camera
s	Start the frame acquisition
t	Stop the frame acquisition
e	Exit the Example

An example of this operation is shown in the image below.


kaya@kaya-System-Product-test: ~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Point/Exam

```
kaya@kaya-System-Product-test:~$ cd '/home/kaya/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.
04_x64/Vision Point/Examples/Vision Point API'
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ make
gcc -o KYFGLib_Example KYFGLib_Example.c -I/opt/KAYA_Instruments/include -L/opt/KAYA_Instruments/lib -lKYFGLib -Wl,-rpath,
/opt/KAYA_Instruments/lib
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ ./KYFGLib_Example
Number of scan results: 3
Device 0: Komodo CoaXPress
Device 1: Chameleon Camera Simulator
Device 2: Demo device

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
0
Selected grabber #0

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
o
Good connection to grabber #0, handle=10000

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
c
Found 1 cameras.
Camera 0 was connected successfully

Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
s
Good callback buffer handle:10101, current index:4, total frames:71
Good callback buffer handle:10101, current index:11, total frames:205380
Good callback buffer handle:10101, current index:9, total frames:205611
Enter choice: ([0-4]-select grabber) (o-open frabber) (c-connect to camera)(s-start)(t-stop)(e-exit)
e

Exiting...
kaya@kaya-System-Product-test:~/Desktop/KAYA_Vision_Point_Setup_branches-sw_4_2_0_x_v4.0.0.2544-Ubuntu_14.04_x64/Vision Po
int/Examples/Vision Point API$ █
```

NOTE:

By default, the make file includes KYFGLib_Example.c which uses **Cyclic Buffer example**.

In order to use **Queued Buffer**, change the KYFGLib_Example.c file to KYFGLib_Example_QueuedBuffers.c located in the same directory, and rebuild example.

23.1 Firmware version 1.xx line selector enumeration

The Line selection enumeration was changed at firmware version 2.xx. If you intend on using firmware version 1.xx please refer to the following table for correct Line selection enumerations.

Value	Output	Gen<i>Cam parameter name
0	OptoCoupled Input	KY_OPTO_IN_0
1	OptoCoupled Input	KY_OPTO_IN_1
2	OptoCoupled Input	KY_OPTO_IN_2
3	OptoCoupled Input	KY_OPTO_IN_3
4	LVDS Input 0	KY_LVDS_IN_0
5	LVDS Input 1	KY_LVDS_IN_1
6	LVDS Input 2	KY_LVDS_IN_2
7	LVDS Input 3	KY_LVDS_IN_3
8	TTL 0	KY_TTL_0
9	TTL 1	KY_TTL_1
10	TTL 2	KY_TTL_2
11	TTL 3	KY_TTL_3
12	TTL 4	KY_TTL_4
13	TTL 5	KY_TTL_5
14	TTL 6	KY_TTL_6
15	TTL 7	KY_TTL_7
16	LVTTL 0	KY_LVTTL_0
17	LVTTL 1	KY_LVTTL_1
18	LVTTL 2	KY_LVTTL_2
19	LVTTL 3	KY_LVTTL_3
20	OptoCoupled	KY_OPTO_OUT_0
21	OptoCoupled	KY_OPTO_OUT_1
22	OptoCoupled	KY_OPTO_OUT_2
23	OptoCoupled	KY_OPTO_OUT_3
24	LVDS Output 0	KY_LVDS_OUT_0
25	LVDS Output 1	KY_LVDS_OUT_1
26	LVDS Output 2	KY_LVDS_OUT_2
27	LVDS Output 3	KY_LVDS_OUT_3
28	Camera 0 Trigger	KY_CAM_TRIG

Table 5 : Line selection options (Firmware version 1.xx)

23.2 Firmware version 1.xx I/O source enumeration

The I/O source enumeration was changed at firmware version 2.xx. If you intend on using firmware version 1.xx please refer to the following table for correct I/O source enumerations.

Valu	Source	Gen<i>Cam	I/O	Tim	Trigg	Enco
0	Disabled	KY_DISABLED	✓	✓	✓	✓

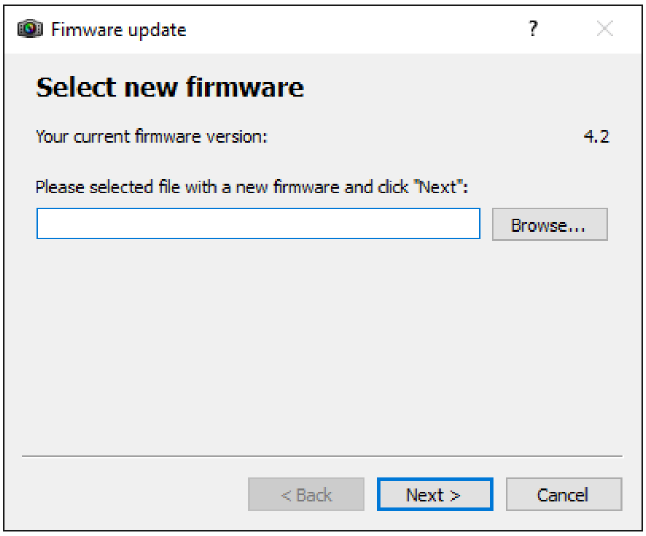
1	OptoCoupled	KY_OPTO_IN_0	✓	✓	✓	✓
2	OptoCoupled	KY_OPTO_IN_1	✓	✓	✓	✓
3	OptoCoupled	KY_OPTO_IN_2	✓	✓	✓	✓
4	OptoCoupled	KY_OPTO_IN_3	✓	✓	✓	✓
5	LVDS Input 0	KY_LVDS_IN_0	✓	✓	✓	✓
6	LVDS Input 1	KY_LVDS_IN_1	✓	✓	✓	✓
7	LVDS Input 2	KY_LVDS_IN_2	✓	✓	✓	✓
8	LVDS Input 3	KY_LVDS_IN_3	✓	✓	✓	✓
9	TTL 0	KY_TTL_0	✓	✓	✓	✓
10	TTL 1	KY_TTL_1	✓	✓	✓	✓
11	TTL 2	KY_TTL_2	✓	✓	✓	✓
12	TTL 3	KY_TTL_3	✓	✓	✓	✓
13	TTL 4	KY_TTL_4	✓	✓	✓	✓
14	TTL 5	KY_TTL_5	✓	✓	✓	✓
15	TTL 6	KY_TTL_6	✓	✓	✓	✓
16	TTL 7	KY_TTL_7	✓	✓	✓	✓
17	LVTTL 0	KY_LVTTL_0	✓	✓	✓	✓
18	LVTTL 1	KY_LVTTL_1	✓	✓	✓	✓
19	LVTTL 2	KY_LVTTL_2	✓	✓	✓	✓
20	LVTTL 3	KY_LVTTL_3	✓	✓	✓	✓
21	OptoCoupled					
22	OptoCoupled					
23	OptoCoupled					
24	OptoCoupled					
25	LVDS Output 0					
26	LVDS Output 1					
27	LVDS Output 2					
28	LVDS Output 3					
29	Camera_Trigger	KY_CAM_TRIG	✓	✓	✓	
30	Reserved					
31	Reserved					
32	Reserved					
33	Continuous	KY_CONTINUOUS		✓		
34	Software	KY_SOFTWARE		✓	✓	
35	Reserved					
36	Encoder 0	KY_ENCODER_0		✓	✓	
37	Encoder 1	KY_ENCODER_1		✓	✓	
38	Encoder 2	KY_ENCODER_2		✓	✓	
39	Encoder 3	KY_ENCODER_3		✓	✓	
40	Timer0Active	KY_TIMER_ACTIVE	✓	✓	✓	
41	Timer1Active	KY_TIMER_ACTIVE	✓	✓	✓	
42	Timer2Active	KY_TIMER_ACTIVE	✓	✓	✓	
43	Timer3Active	KY_TIMER_ACTIVE	✓	✓	✓	
44	Timer4Active	KY_TIMER_ACTIVE	✓	✓	✓	
45	Timer5Active	KY_TIMER_ACTIVE	✓	✓	✓	
46	Timer6Active	KY_TIMER_ACTIVE	✓	✓	✓	
47	Timer7Active	KY_TIMER_ACTIVE	✓	✓	✓	
48	User Output 0	KY_USER_OUT_0	✓			
49	User Output 1	KY_USER_OUT_1	✓			
50	User Output 2	KY_USER_OUT_2	✓			
51	User Output 3	KY_USER_OUT_3	✓			
52	User Output 4	KY_USER_OUT_4	✓			
53	User Output 5	KY_USER_OUT_5	✓			
54	User Output 6	KY_USER_OUT_6	✓			
55	User Output 7	KY_USER_OUT_7	✓			

Table 6 : Frame Grabber I/O source (Firmware version 1.xx)

24.1 Updating the device firmware using Vision Point Application

In order to update the firmware of a KAYA Instrument's device, an "XXX_XX.bin" file is needed, when the XXX is the board name and XX is the desired firmware number.

1. In the Toolbar Menu, under Device Control tab, chose the "Firmware update" option. A new window will open displaying the current device firmware version.
2. Click "Browse…" button, as shown in Figure 3, and select the desired firmware update file, in accordance with the device chosen (.bin file extension), and Click "Next >" button.



Fimware update

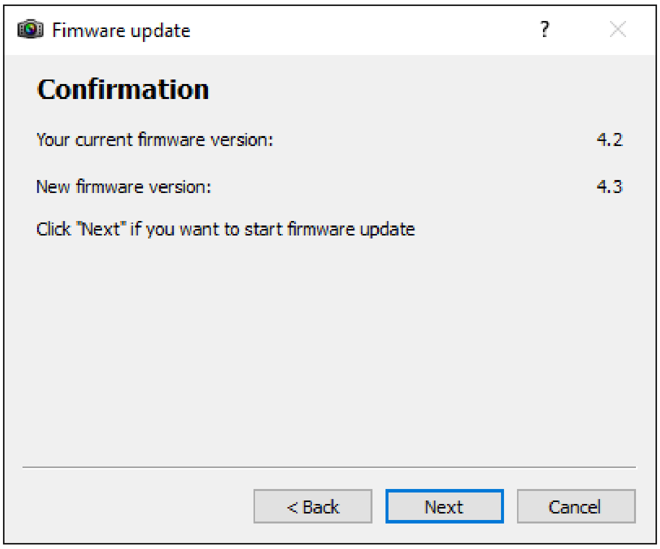
Select new firmware

Your current firmware version: 4.2

Please selected file with a new firmware and click "Next":

Figure 3 : Firmware Update selection window

3. The next window will display both, current and new firmware, as shown in the Figure 4. By clicking the "Next >" button, the conformation is made and the firmware update will start immediately.



Fimware update

Confirmation

Your current firmware version: 4.2

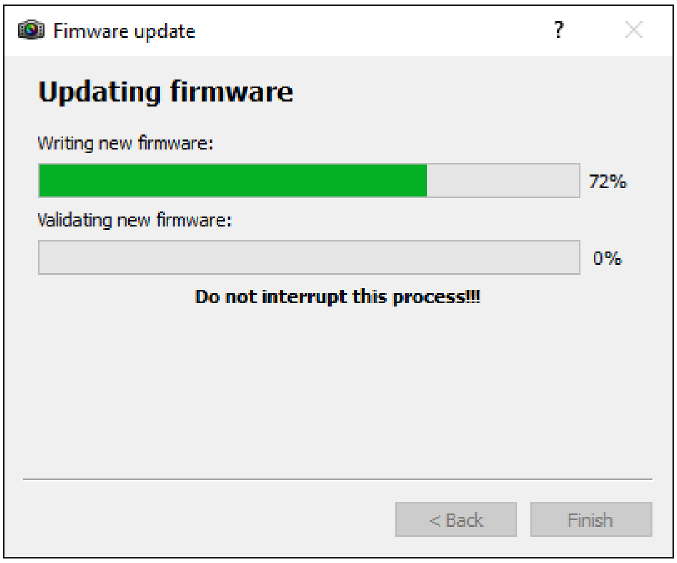
New firmware version: 4.3

Click "Next" if you want to start firmware update

Figure 4 : Firmare Update Confirmation window

4. The next window displays the initiated firmware update. The firmware update process displayed in the first progress bar and the firmware validation displayed in the second, as shown in Figure 5.
5. **Do not interrupt the process!**

   In case of an error, the firmware update will fail and return to previous operation mode.
6. A successful update will result in reaching 100% on both progress bars.
7. **A full PC power off cycle is required to activate the new firmware**.
8. **Turn** on the PC and check the firmware version by opening the Vision Point application, Frame Grabber tab. The firmware version located under Hardware information. Make sure that the firmware version matches the version supplied. That would insure the success of the firmware update operation.


Firmware update

Updating firmware

Writing new firmware: 72%

Validating new firmware: 0%

Do not interrupt this process!!!

Figure 5 : Firmare Update process window

24.2 Updating the device firmware using pre-built utility for Linux

In order to update the firmware of a KAYA Instrument's device, an "XXX_XX.bin" file is needed, when the XXX is the board name and XX is the desired firmware number.

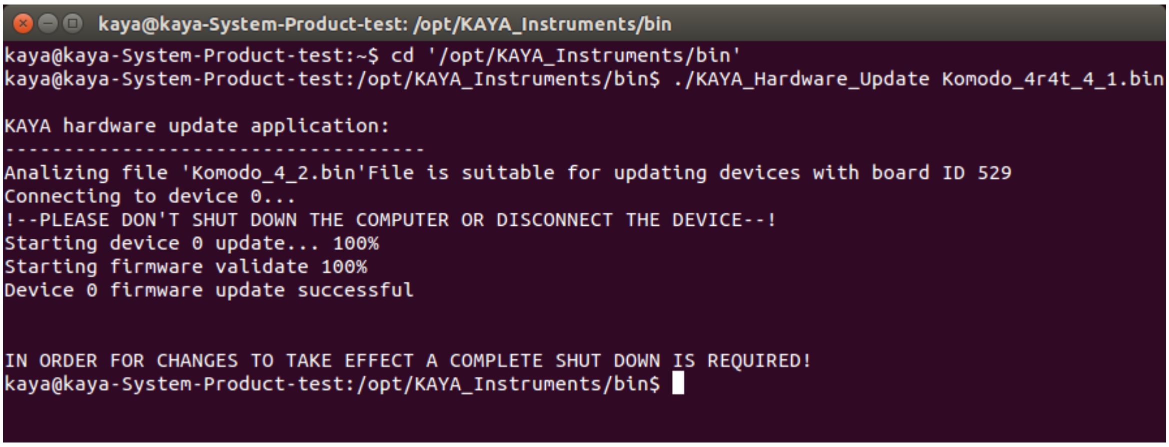
WARNING: Currently this method is not suitable in setups where more than one board with the same product ID installed on the same machine. Please apply to KAYA's support if you need to update firmware in such setup.

1. Make sure the .bin file is present in a local directory.
2. Open the Terminal and enter the directory path of KAYA Hardware Update executable file: "cd 'opt/KAYA_Instruments/bin' ".

3. Execute the KAYA Hardware Update using full path to the firmware update file as a parameter.

   Example: "./KAYA_Hardware_Update <*path_to_folder_with_bin_file*>/Komodo_4_3.bin ".
4. Press Enter and wait for a message that indicates the end of process.
5. **Do not interrupt the process!**
6. **A full PC power off cycle is required to activate the new firmware**.
7. The sequence of the steps is illustrated in the screenshot below.

Please, Contact KAYA Instruments' representative for any question.



```
kaya@kaya-System-Product-test:~$ cd '/opt/KAYA_Instruments/bin'
kaya@kaya-System-Product-test:/opt/KAYA_Instruments/bin$ ./KAYA_Hardware_Update Komodo_4r4t_4_1.bin

KAYA hardware update application:
------------------------------------
Analizing file 'Komodo_4_2.bin'File is suitable for updating devices with board ID 529
Connecting to device 0...
!--PLEASE DON'T SHUT DOWN THE COMPUTER OR DISCONNECT THE DEVICE--!
Starting device 0 update... 100%
Starting firmware validate 100%
Device 0 firmware update successful


IN ORDER FOR CHANGES TO TAKE EFFECT A COMPLETE SHUT DOWN IS REQUIRED!
kaya@kaya-System-Product-test:/opt/KAYA_Instruments/bin$ █
```

Figure 6 : Firmare Update via Terminal process window

24.3 Updating the device firmware using pre-built utility for Windows

In order to update the firmware of a KAYA Instrument's device, an "XXX_XX.bin" file is needed, when the XXX is the board name and XX is the desired firmware number.

WARNING: Currently this method is not suitable in setups where more than one board with the same product ID installed on the same machine. Please apply to KAYA's support if you need to update firmware in such setup.

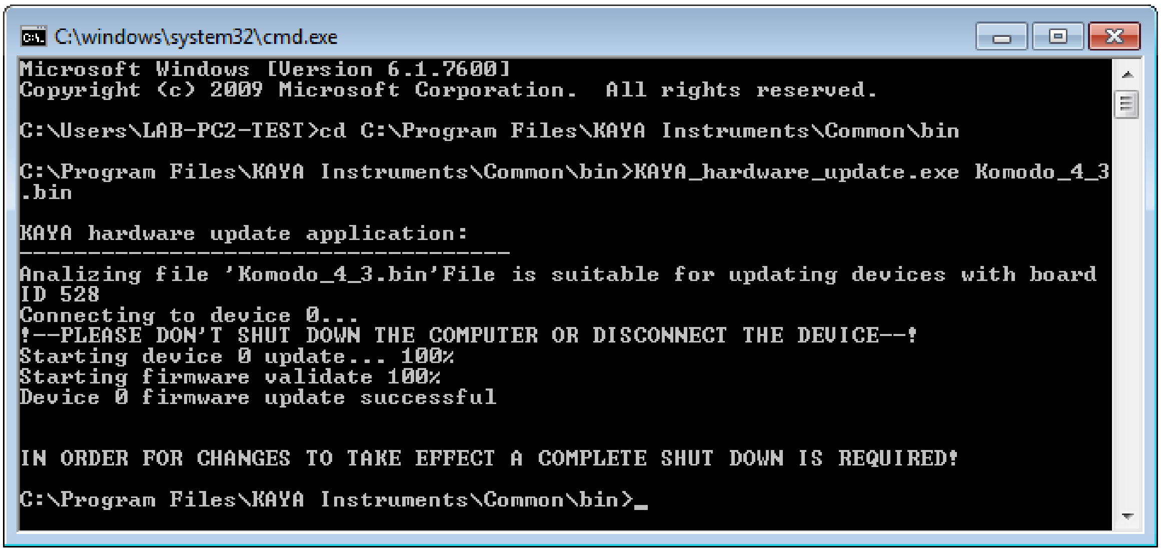
1. Make sure the .bin file is present in a local directory.
2. Open the Command line and enter the directory path of KAYA Hardware Update executable file:

   "cd '\Program Files\KAYA_Instruments\Common\bin' ".
3. Execute the KAYA Hardware Update using full path to the firmware update file as a parameter.

   Example: "KAYA_Hardware_Update <*path_to_folder_with_bin_file*>/Komodo_4_3.bin ".
4. Press Enter and wait for a message that indicates the end of process.
5. **Do not interrupt the process!**

6. **A full PC power off cycle is required to activate the new firmware**.
7. The sequence of the steps is illustrated in the screenshot below.

Please, Contact KAYA Instruments' representative for any question.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\LAB-PC2-TEST>cd C:\Program Files\KAYA Instruments\Common\bin

C:\Program Files\KAYA Instruments\Common\bin>KAYA_hardware_update.exe Komodo_4_3
.bin

KAYA hardware update application:
------------------------------------
Analizing file 'Komodo_4_3.bin'File is suitable for updating devices with board
ID 528
Connecting to device 0...
!--PLEASE DON'T SHUT DOWN THE COMPUTER OR DISCONNECT THE DEVICE--!
Starting device 0 update... 100%
Starting firmware validate 100%
Device 0 firmware update successful


IN ORDER FOR CHANGES TO TAKE EFFECT A COMPLETE SHUT DOWN IS REQUIRED!

C:\Program Files\KAYA Instruments\Common\bin>_
```

Figure 7 : Firmare Update via Command line process window

24.4 Collecting log files

The log files created and override each time the application is launched.

Windows operating system:

KAYA's log folder can be easily opened using one of the two ways, listed below:

1. Choose Log files folder under KAYA Instruments from the quick start:

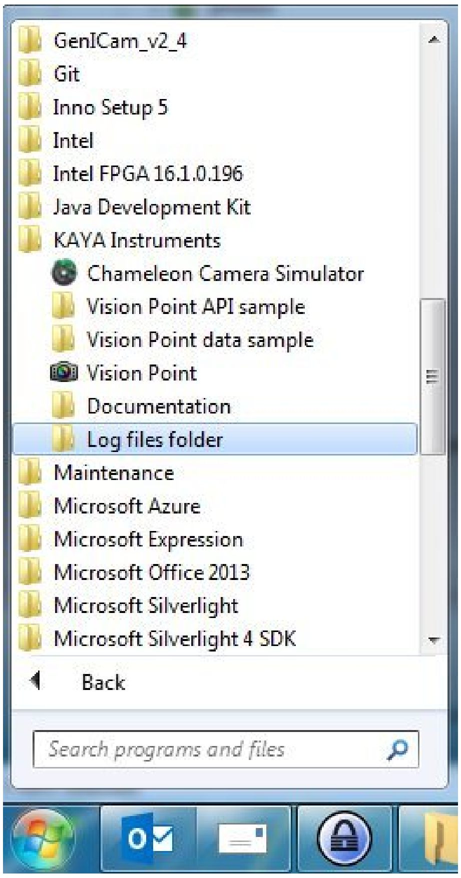


Figure 8 : Log files folder from the quick start menu path

2. Using Vision Point application. Enter “Help” tab and click on "Open logs folder" option.

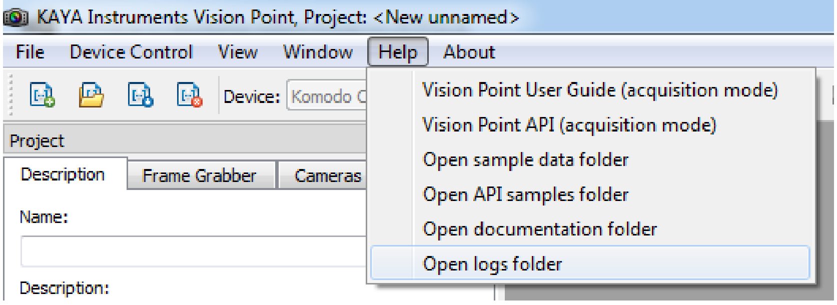


Figure 9 : Log files folder from Vision Point Help menu path

NOTES:

1. A separate log file is created for each application, which uses KAYA API, with a display name of the main executable with addition of process ID and timestamp.
2. The Vision Point application installation log files folder can be found under user’s main driver: C:\Program Files\KAYA Instruments\Log\Installer folder.

Linux operating system:

KAYA’s log files folder can be easily opened following the path:
/var/log/KAYA_Instruments

24.5 Technical Support and Professional Services

If you searched Vision Point API Data Book document and could not find the answers you need, contact KAYA Instruments support service. Phone numbers for our office are listed at the front of this document.

24.6 Submitting a support request

Before opening support request, one should prepare the following information:

- PC configuration
- Operation System
- Card part number or full name
- Firmware in use
- Software in use