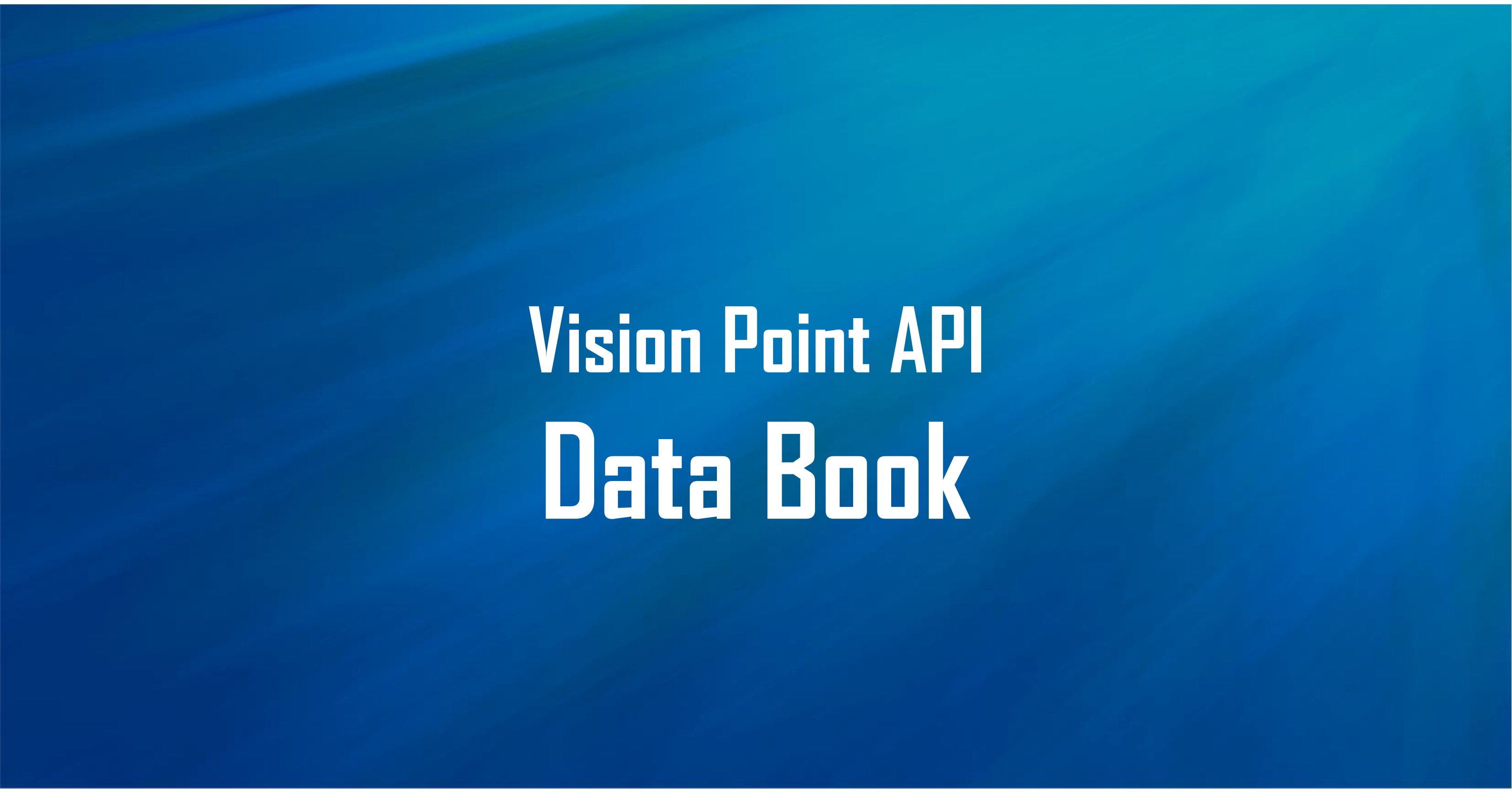


Vision Point API
Data Book

January 2023 - Rev 6.3.0

KAYA INSTRUMENTS

www.kayainstruments.com

20 HaMesila St., Nesher 3688520, Israel
POB 25004, Haifa 3125001, Israel
Tel:(+972)-72-2723500 Fax:(+972)-72-2723511

Table of Contents

1 Figures and Tables

1.1 List of Figures

1.2 List of Tables

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 the saving of Frame Grabber configurations before camera discovery. - KYFG_CameraOpen() has been deprecated and replaced with KYFG_CameraOpen2() - KYFG_AuxDataCallbackRegister() for Auxiliary data retrieval, from various 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 the description of the new function.
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 the system's PATH. Instead, will be installed into a sub-folder for internal use only. If these libraries are needed by the user's application, they should be installed separately. - Visual Studio 2017 flavor support. Users 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 sizes of ControlPacketDataSize via Grabber configurations - Virtual Grabber: Add new versioning register to support new suppression 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 .fgprj file
5.1	09/2019	Visio Point API release 5.1 - Added option to specify Gen<I>Cam library source path - Chameleon: Fixed issue with first stream start doesn't output image - Chameleon: Fixed issue with loaded user xml file, parameters doesn't change from remote source - Chameleon: Fixed configuration of "TimerControl", "SimulationTriggerDelay" and "SimulationTriggerFilter" parameters - Added support for new Frame Grabber devices (e.g Predator II) - Added support for Frame Grabber devices with 12.5Gbit connection speed - Extended usage of KYFGLib_Initialize() function. - Added KYFGLib_InitParameters structure - Added KYFGLIB_CONCURRENCY_FLAGS enumeration - Removed KYFG_Pid2Name() function. This function is no longer implemented. User should use KY_DeviceDisplayName() function instead.
5.2	06/2020	Visio Point API release 5.2 - **Windows 7 is no longer supported.** - **VS2012 run-time is no longer supported, KYFGLib.dll/lib are identical copies of KYFGLib_vc141.dll/lib for projects that link with "KYFGLib.lib"** - "Simulation example" renamed "Chameleon example" - Function 'KYFG_Scan()' is deprecated in favor of 'KY_DeviceScan()' - KY_DEVICE_INFO: - support value 2 of "version" filed: added field "m_Flags" - support value 3 of "version" filed: added field "m_Protocol" - Added descriptiin of FGSTATUS_STREAM_IS_LOCKED and FGSTATUS_STREAM_CANNOT_LOCK error codes - Added temperature events - Added KY_SOFTWARE_VERSION structure and KY_GetSoftwareVersion() function - Added KYFG_CameraInfo2() and KYFGCAMERA_INFO2
5.3	08/2020	Visio Point API release 5.3 - Added max "FifoThreshold" Grabber parameter value, calculated using available device memory. - "DeviceMemorySize" Grabber parameter is deprecated. - Support for 2nd generation of KAYA's Frame Grabbers (Linux). - Initial support for 2nd generation of KAYA's Frame Grabbers (Xavier).

		- Fixed camera communication command Futex error in Ubuntu 18.04 for CLHS and 10GigE Frame Grabbers (Linux). - Specific stream unaligned resolution allocation for CoaXPress Frame Grabber minor fix. - Corrected reflection of fan control hysteresis on/off parameters for 2nd generation KAYA Frame Grabbers. - Vision Point API data book merged with Chameleon API data book document.
5.4	12/2020	Visio Point API release 5.4 - Ubuntu 20.04 is now supported. - Xavier JetPack 4.4.1 SDK support. - Added configurable log retaining policy. - Automatic Power over CoaXPress management for Predator. NOTE: Latest firmware for the Predator card requires the latest version of Vision Point (**2020.3** or later). - Add "DevicePhysicalLinksMax" Frame Grabber parameter indicating maximum available physical links. - Fix "KYFG Get Camera Value String.vi" and "KYFG Get Grabber Value String.vi" in LabView adaptor. - Corrected behavior of BUFFER_INFO_PIXELFORMAT_NAMESPACE and BUFFER_INFO_DELIVERED_IMAGEHEIGHT commands used in GenTL's DSGetBufferInfo API function. - Limit the number of PoCXP monitoring channels for Predator II.
5.4 (patch)	04/2021	Visio Point API release 5.4 (Service pack 1) - Fix functionality of KYFG_GetCameraValueFloatMaxMin() and KYFG_GetCameraValueIntMaxMin() - Fix buffer allocation problem for certain unaligned resolutions used with "SegmentsPerBuffer" grabber configuration parameter. - Fix temperature indication for Komodo II CXP - Improved stability of serial port enumeration and communication used with external clserkyi.dll - FGSTATUS error codes table was updated according to KYFGLib_defines. file
6.0	09/2021	Visio Point API release 6.0 - CoaXPress 2 support was introduced in this software release, including CXP2 tagged command packets, generate and receive CXP2 HeartBeats and Events - GenTL example code can be found under “<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/GenTL_simple_test.c”. - New events added to the enumeration "KYDEVICE_EVENT_ID": - KYDEVICE_EVENT_CXP2_HEARTBEAT_ID - KYDEVICE_EVENT_CXP2_EVENT_ID - KYDEVICE_EVENT_GENCP_EVENT_ID KYDEVICE_EVENT_GIGE_EVENTDATA_ID With corresponding structures added: - KYDEVICE_EVENT_CXP2_HEARTBEAT - KYDEVICE_EVENT_CXP2_EVENT - KYDEVICE_EVENT_GENCP_EVENT - KYDEVICE_EVENT_GIGE_EVENTDATA - New functions added: - KYCS_GenerateCxpEvent() - KYFG_CameraScanEx() - KYFG_DevicePortSendEventMessage() - KYFG_CameraSendEventMessage() - API C examples now show how a user can allocate their own acquisition buffers in their software and submit them to the KYFGLib library. - GenTL example code was added to the installation package. - Python API binding was updated with missing and new functionality. - Added new device IDs (see Vision_Point_API_Release_Notes .pdf for details).
6.0.1	10/2021	Visio Point API release 6.0.1 - API function call Sequence was updated

6.1	04/2022	Visio Point API release 6.1 - Vision Point API data book was rearranged, combining native API and both C# and Python bindings under the same function - Python examples were added to support multiple camera flow - C# binding new functions were added to the API: - KYFG_GetGrabberValueIntMaxMin() - KYFG_GetGrabberValueFloatMaxMin() - KYFG_GetCameraValueIntMaxMin() - KYFG_GetCameraValueFloatMaxMin() - KYFG_StreamLinkFramesContinuously() - KYFG_DevicePortSendEventMessage() - KYFG_CameraSendEventMessage() - KYCS_GetPortStatus() - Python binding new and fixed functions: - KYFG_BufferSubmit() - KYFG_StreamBufferCallbackRegister() - KYFG_StreamBufferCallbackUnregister() - KYFG_CameraCallbackRegister() - KYFG_CameraCallbackUnregister()
6.2	09/2022	Visio Point API release 6.2 - Working with the debugger section was added to the document - KYFG_BufferRevoke() new function was added to public API - Update KY_STREAM_INFO_CMD structure Function KYFG_CameraGetXML() returns data from the camera as-is, therefore, client software should not rely on terminating 0 in returned value, such code will now fail and must be changed, i.e. the function KYFG_CameraGetXML() fills output buffer WITHOUT terminating 0 char, even if the result is not zipped.
6.2 (patch)	11/2022	Visio Point API release 6.2 (Service pack 1) - Added Get/Set Camera/Grabber parameters usage limitation in section 5.3
6.3.0	01/2023	Visio Point API release 6.3.0 - KYFG_BufferAnnounceChunks() new function was added to public API - Updated Komodo 4R4T system configuration section - Updated KY_STREAM_BUFFER_INFO_CMD class

Table 1 – Revision History

3 Introduction

3.1 Safety precautions

With your KAYA’s board in hand, please take the time to read through the precautions listed below to prevent preventable and unnecessary injuries and damage to you, other personnel, or property. Read these safety instructions carefully before your first use of the product, as these precautions contain safety instructions that must be observed. Be sure to follow this manual to prevent misuse of the product.

⚠ Caution! Read Carefully and do not disregard these instructions.
In the event of a failure, disconnect the power supply Disconnect the power supply immediately and contact our sales personnel for repair. Continuing to use the product in this state may result in a fire or electric shock.
If an unpleasant smell or smoking occurs, disconnect the power supply. Disconnect the power supply immediately! Continuing to use the product in this state may result in a fire or electric shock. After verifying that no smoking is observed, contact our sales personnel for repair.
Do not disassemble, repair or modify the product. This may result in a fire or electric shock due to a circuit shortage or heat generation. Contact our sales personnel before inspection, modification or repair.
Do not place the product on unstable surfaces. Otherwise, it may drop or fall, resulting in injury to persons or the camera.
Do not use the product if dropped or damaged. Otherwise, a fire or electric shock may occur.
Do not touch the product with metallic objects. Otherwise, a fire or electric shock may occur.
Do not place the product in dusty or humid environments, nor where water may splash. Otherwise, a fire or electric shock may occur.
Do not wet the product or touch it with wet hands. Otherwise, the product may fail or it may cause a fire, smoking or electric shock.
Do not touch the gold-plated sections of the connectors on the product. Otherwise, the surface of the connector may be contaminated by sweat or skin-oil, resulting in contact failure of a connector, malfunction, fire or electric shock due to static electricity discharge.
Do not use or place the product in the following locations. ▪ Unventilated areas such as closets or bookshelves. ▪ Near oils, smoke or steam. ▪ Next to heat sources. ▪ A closed (and not running) car where the temperature becomes high. ▪ Static electricity replete locations ▪ Near 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 objects on the product. Otherwise, the product may be damaged.
Be sure to discharge static electricity from the body before touching any sensitive electronic components. The electronic circuits in your computer and the circuits on the board are sensitive to static electricity and surges. Improper handling may seriously damage the circuits. In addition, do not let your clothing come in contact with the circuit boards or components. Otherwise, the product may be damaged.

3.2 Disclaimer

KAYA Instruments will assume no responsibility for any damage that may ensue by the use of this product for any purpose other than intended, as previously stated. Without detracting from what was previously written, please be advised that the company will take no responsibility for any damages caused by:

- Earthquake, thunderstrike, natural disasters, fire caused by use beyond our control, wilful and/or accidental misuse and/or use under other abnormal and/or unreasonable conditions.
- Secondary damages caused by the use of this product or its unusable state (business interruption or others).
- Use of this product in any manner that contradicts this manual or malfunctions that may occur due to connection to other devices. Damage to this product that is out of our control or failure due to modification
- Accidents and/or third parties that may be involved.

Additionally, **KAYA Instruments** assumes no responsibility or liability for:

- Erasure or corruption of data caused by the use of this product.
- Any consequences or other abnormalities following the use of this product

4 Overview

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

This API is to be used with KAYA's Frame Grabbers and Chameleon camera simulator hardware provided by KAYA Instruments. This is a high-level API for connecting, configuring and capturing data streaming over 1, 2, 4 or 8 channels. KAYA's Frame Grabbers are capable of connecting to various cameras at various speeds and topologies. KAYA's Chameleon camera simulator is capable of generating streams at various speeds and pixel formats.

4.1 Document Structure

This API guide is divided into few major topics each related to different functionalities:

- Connection and Info: Connect/disconnect to a specific hardware device.
- Camera Configurations:
  Acquisition mode:
  - Scan, connect and get camera information.
  - Use camera native XML or override with another XML file.

  Generation mode:
  - Simulating and configuring different cameras.
  - Use camera Simulator built-in XML or override with another camera XML file.
- Callback Functions
  - Callback functions for data acquisition/generation.
- Camera/Frame Grabber values
  - Use XML fields to configure the camera, camera simulator and Frame Grabber parameters, according to Gen<i>Cam standard naming and XML field definition and type.
- Stream interface
  - Access and handle of each allocated buffer in memory.
- Data acquisition/generation
  - Acquisition/Generation of data streams.
- 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 parameter types and definitions 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.

4.2 Function Call Sequence

For the API to carry out the desired results, one of the following sequences of function calls should be followed, depending on the utilization mode:

4.2.1 Acquisition mode

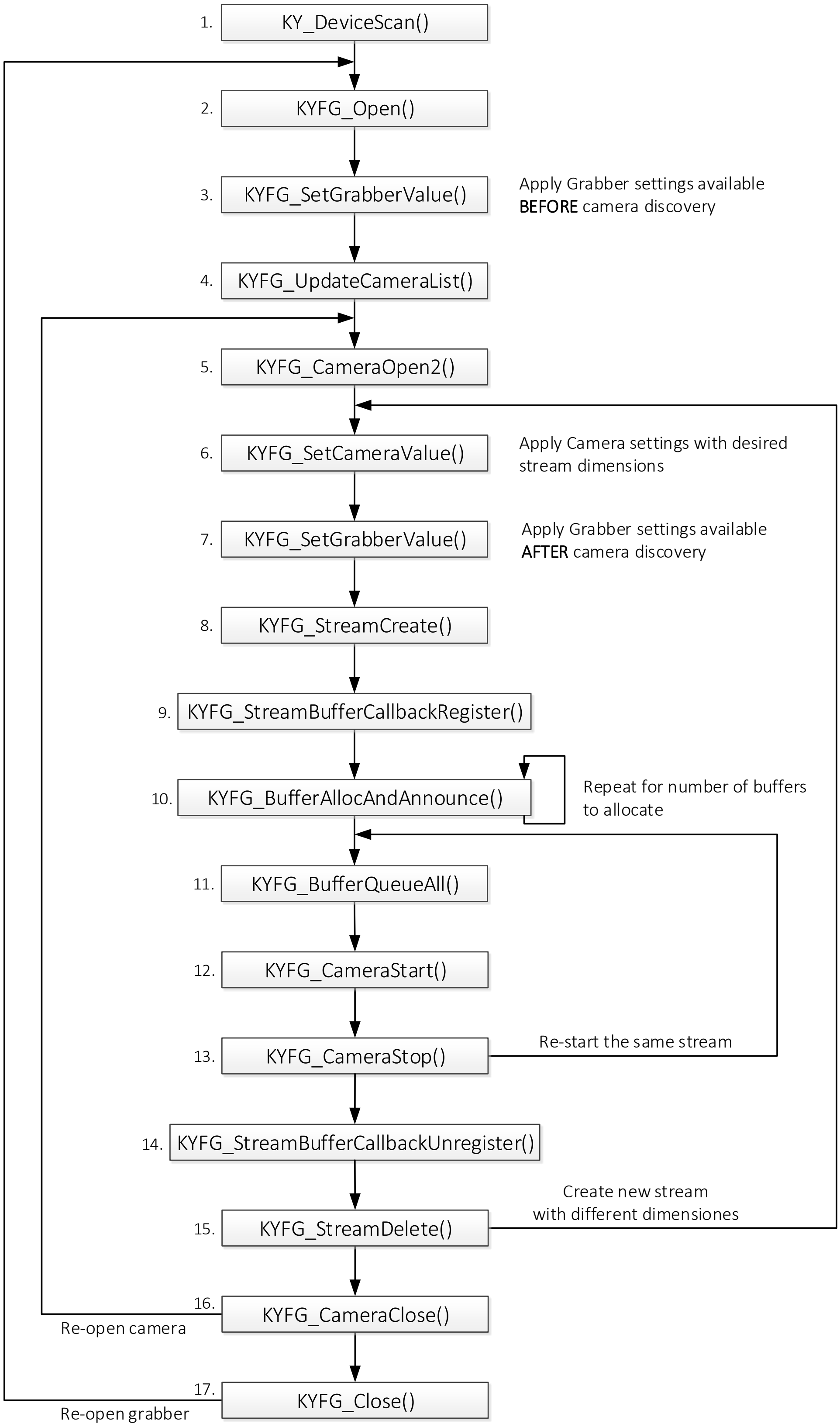

1. KY_DeviceScan()
2. KYFG_Open()
3. KYFG_SetGrabberValue()
Apply Grabber settings available BEFORE camera discovery
4. KYFG_UpdateCameraList()
5. KYFG_CameraOpen2()
6. KYFG_SetCameraValue()
Apply Camera settings with desired stream dimensions
7. KYFG_SetGrabberValue()
Apply Grabber settings available AFTER camera discovery
8. KYFG_StreamCreate()
9. KYFG_StreamBufferCallbackRegister()
10. KYFG_BufferAllocAndAnnounce()
Repeat for number of buffers to allocate
11. KYFG_BufferQueueAll()
12. KYFG_CameraStart()
13. KYFG_CameraStop()
Re-start the same stream
14. KYFG_StreamBufferCallbackUnregister()
15. KYFG_StreamDelete()
Create new stream with different dimensions
16. KYFG_CameraClose()
Re-open camera
17. KYFG_Close()
Re-open grabber

Figure 1 – Acquisition mode function call sequence

1. Scan for devices currently connected to the PC. This will return number of found devices, both physical and virtual.
2. Open a control to a selected device by index.
3. (Optional) function call to set the desired values to determine the Grabber parameters using different available methods, before camera discovery.
4. Scan for available cameras and updates the list of cameras currently connected to the device. There is no restriction on connection topology, connection supported speed is according to device and protocol specification.
5. Open a connection to the selected camera and load its native or external XML file.
6. (Optional) function-call to set the desired values to determine the camera parameters using different available methods.
7. (Optional) function call to set the desired values to determine the Grabber parameters using different available methods, after camera discovery.
8. Creates a video stream object, based on Grabber and Camera configurations at creation state. The stream provides required frame allocation information and manages frame acquisition.
9. Register a stream runtime acquisition callback function. To work properly, the callback (userFunc) 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 Stream Interface functions to retrieve currently acquired frame.
10. Allocate and announce a buffer and bind it to a specific stream. In most cases, it is advised to allocate the buffer memory size corresponding to a full frame. For continues acquisition, several buffers should be allocated in order to prevent from hardware dropping any incoming data frames.
11. Move all announced buffers to input queue and prepare them for acquisition.
12. Start the acquisition for the specified camera.
13. Stop the acquisition for the specified camera.
14. Unregister a stream runtime acquisition callback function.
15. Deletes a stream. Any memory allocated by the user is NOT freed by this function. All memory allocated by the library is freed and all API handles bound to the stream became invalid.
16. If an acquisition is no longer required from an active camera, close the connection to the chosen device. KYFG_CameraOpen2() can be used to re-connect to the same camera, unless it was physically disconnected, without the need to re-scan for cameras.
17. Close the connection to the Frame Grabber, this will clear all relevant resources.

4.2.2 Generation mode

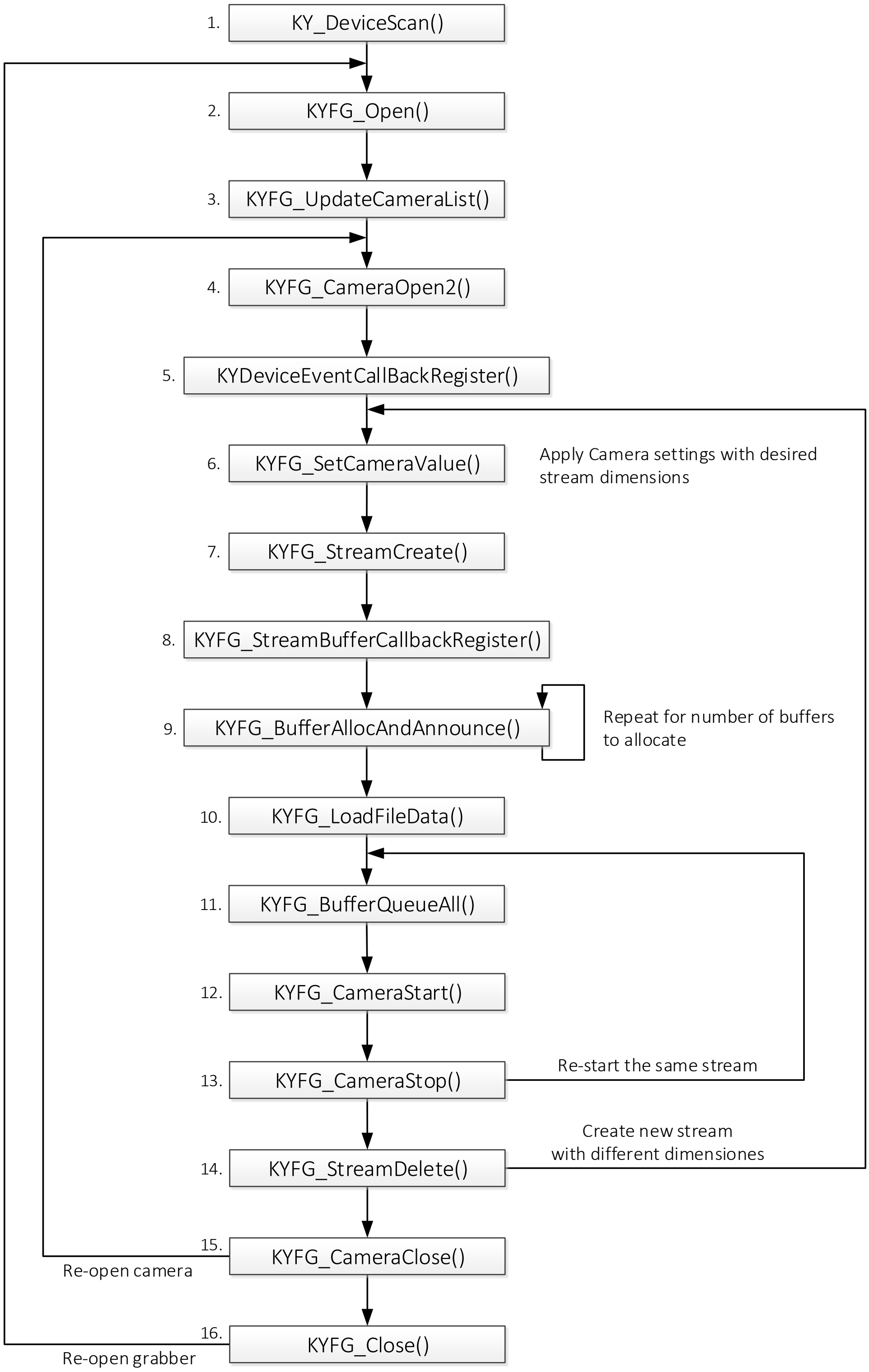

1. KY_DeviceScan()
2. KYFG_Open()
3. KYFG_UpdateCameraList()
4. KYFG_CameraOpen2()
5. KYDeviceEventCallBackRegister()
6. KYFG_SetCameraValue()
Apply Camera settings with desired stream dimensions
7. KYFG_StreamCreate()
8. KYFG_StreamBufferCallbackRegister()
9. KYFG_BufferAllocAndAnnounce()
Repeat for number of buffers to allocate
10. KYFG_LoadFileData()
11. KYFG_BufferQueueAll()
12. KYFG_CameraStart()
13. KYFG_CameraStop()
Re-start the same stream
14. KYFG_StreamDelete()
Create new stream with different dimensiones
15. KYFG_CameraClose()
Re-open camera
16. KYFG_Close()
Re-open grabber

Figure 2 – Generation mode function call sequence

1. Scan for devices 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 device. Use the index corresponding to the device from the array acquired in the previous step.
3. Scan for cameras and updates the list of currently determined cameras. There is no restriction on connection topology or CoaXPress camera speed.

4. Open a connection to the specific camera and load its native or external XML file. If no external XML file is provided, then only internal XML will be used providing a minimal set of mandatory camera parameters.
5. Register a callback function for the device events. The handler will generate a KYDEVICE_EVENT_CAMERA_START_REQUEST event which indicates a request for start generation from the host. Following this event, the Chameleon should at least start the stream generation, or even perform a complete stream allocation and fill buffers with data.
6. (Optional) function-call to set the desired values to determine the camera parameters using different available methods.
7. Allocate the memory required for acquiring a video stream from a chosen camera. The stream will manage frame buffers allocated either by the user or by the library. Frame buffers will be organized in queues. Several frames should be allocated in order not to immediately run over previously received data.
8. Register a stream runtime generation callback function. To work properly, the callback (userFunc) 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 Stream Interface functions to retrieve currently acquired frame. KYFG_CameraOpen2() and KYFG_CameraClose() doesn't invalidate the callback registration.
9. 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 and should be repeated according to the desired amount of frame buffers.
10. Load image file(s) and fill the allocated buffers with its data, according to specified image dimensions and bitness.
11. Move all announced frame buffers from one queue to another input queue.
12. Start generation of a video stream from a specified camera.
13. Stop generation of a video stream from a specified camera.
14. Deletes a stream. Any memory allocated by the user is NOT freed by this function. All memory allocated by the library is freed and all API handles bound to the stream became invalid.
15. If stream generation from a specific camera is no longer required, the connection may be closed. To connect back to the camera and only if this camera wasn't physically disconnected, no camera scan is needed, and KYFG_CameraOpen2() can be called.
16. Close the connection to a Chameleon device, using the corresponding index from the array acquired in step 1.

5 API Notes and Limitations

5.1 API Usage in Multi-Threaded Applications

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 accessed via an instance of STREAM_HANDLE
- A frame buffer accessed via an instance of STREAM_BUFFER_HANDLE

5.2 Linux blocking calls and signals in the streaming thread

In Linux, user should not use blocking calls and blocking operations (such as reading std::cin) in the same thread, which is subscribed to signals, i.e. in the thread where KYFG_Open() function was called.
There are two possible solutions to overcome this:

1. Use different threads for KYFG_Open() and std::in, as well as for any other blocking operations such as reading/writing files, etc.
2. Use KYFGLib_Initialize() function before opening any Frame Grabber, cameras and starting streams, and add KYFGLIB_CONCURRENCY_SA_RESTART to the "concurrency_mode" field of the passed KYFGLib_InitParameters instance.

The 1st approach is preferred over the 2nd one because using KYFGLIB_CONCURRENCY_SA_RESTART may impact stream performance in certain circumstances. This is because what this flag usually means is that your blocking thread will still be interrupted by a signal, but the operating system will just save that thread state, switch context to signal handling procedure and then restore the context to proceed with the blocking call.

⚠ 5.3 Important notes

1. **Async-signal-safe:**
   In Linux, KAYA's API functions are NOT async-signal-safe, i.e. they can NOT be safely called from within a signal handler. Read more about "signal-safety".

2. **DllMain function:**
   KAYA's API should **NOT** be used from DllMain function on Windows OS.
   There are significant limits on what you can safely do at a DLL entry point. See General Best Practices for specific Windows APIs that are unsafe to call in DllMain. If anything other than the simplest initialization is required, it is recommended to do that in an initialization function for the DLL. You can require applications to call the initialization function after DllMain has run and before they call any other functions in the DLL.

3. **Get/Set parameters:**
   KYFG_Get/Set Grabber/Camera parameters functions are inherently slow because they utilize the GenICam reference implementation. Therefore, we do not suggest using them in performance-critical parts of the code, such as the stream callback function, etc. Instead, we highly recommend using KYFG_BufferGetInfo() with a relevant command to retrieve the required information. You can still use those functions in non performance-critical parts for example at the system initialization, before or after acquisition sessions etc.

6 Library Exiting

During its operation, our library allocates certain resources that must be freed before the library is unloaded from the process:

6.1 Memory buffers

When you use KYFG_StreamCreateAndAlloc(), and memory buffers are allocated by our library, they are marked as such (to distinguish with a scenario when buffers were allocated by the user program, announced using KYFG_BufferAnnounce() and thus should also be released by the user program). These buffers are released by our library when the user calls KYFG_StreamDelete(), which should be done by the user's application. If you do not call KYFG_StreamDelete(), it will be internally called by the KYFG_CameraClose(), which also should be done by the user's application. If, for some reason, you also do not call KYFG_CameraClose(), it will be internally called by the KYFG_Close(), which your application **must** call at some point.

6.2 Background monitoring thread

THere is a background monitoring thread started by our library when an application calls KYFG_Open() (or KYFG_OpenEx()). This thread is stopped when KYFG_Close() is called. Unloading the library without closing all grabber handles may lead to undefined behaviour (uncaught exceptions, etc.). You application must call KYFG_Close() before unloading the library.

6.3 Callback thread

This is the thread that is waked up by the DMA interrupts and calls stream callback functions in the application. Since this operation is asynchronous, there is no guarantee that application's stream callback will not be called after other ..Close() calls are performed. You application should make sure that all stream callback invocations are ignored after it has closed all handles described above.

NOTE about DllMain's DLL_PROCESS_DETACH in Windows

One could expect that when our DLL is unloaded from the process memory, it can automatically release all relevant resources (stop threads, unpin and release memory, etc). Unfortunately, as it is stated in the "DllMain entry point" documentation, "There are significant limits on what you can safely do in a DLL entry point". Our library does implement some cleanup steps in its DllMain function, but it is still responsibility of an application to perform full cleanup before library is unloaded.

7 Connection and Info

Before starting, please read section 6 on how to perform library exiting correctly.

7.1 KY_GetSoftwareVersion()

Retrieves information about currently running software version.

C

```
FGSTATUS KYFGLib_KY_GetSoftwareVersion(KY_SOFTWARE_VERSION* pVersion);
```

Parameter name	Type	Description
pVersion	KY_SOFTWARE_VERSION *	Pointer to KY_SOFTWARE_VERSION structure.

Return value:

FGSTATUS - Status and error report.

C#

```
KY_SOFTWARE_VERSION Lib.GetSoftwareVersion();
```

Return value:

KY_SOFTWARE_VERSION - Structure with info about KAYA Vision Point software.

Python

```
def KY_GetSoftwareVersion()
```

Return value:

FGSTATUS - Status and error report.
KY_SOFTWARE_VERSION - Structure with info about KAYA Vision Point software.

7.2 KYFGLib_Initialize()

An optional call before KY_DeviceScan(). Initializes KYFGLib library and fills KYFGLib_InitParameters structure with various parameters about KYFGLib library.

C

```
FGSTATUS KYFGLib_Initialize(KYFGLib_InitParameters* pKYFGLib_InitParameters);
```

Parameter name	Type	Description
pKYFGLib_InitParameters	KYFGLib_InitParameters*	Pointer to KYFGLib_InitParameters structure. [1]

Return value:

FGSTATUS - Status and error report.

C#

```
void Lib.Initialize(InitParameters^ initParameters);
```

Parameter name	Type	Description
initParameters	InitParameters^ [1]	Pointer to InitParameters structure [2].

Python

```
def KYFGLib_Initialize(initParams)
```

Parameter name	Type	Description
initParams	KYFGLib_InitParameters [1]	Pointer to InitParameters structure [2].

Return value:

FGSTATUS - Status and error report.

Remarks:

1. For details please refer to the KYFGLib_InitParameters structure description.
2. Users who want this member to be considered by KYFGLib_Initialize() call must set the version value to 2 or higher.

7.3 KYFG_Scan() (DEPRECATED)

A deprecated function and is no longer supported. New applications should use KY_DeviceScan().
Scans for KAYA devices currently connected to the PC PCIe slots and available virtual devices and optionally fills the array with device IDs. [1]

C

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

Parameter name	Type	Description
pids_info	unsigned int*	Pointer to <pid> array of scanned devices. [2]
count	int	The number of devices to assign to pids_info array (assume pids_info array is valid).

Return value:
Number of connected hardware and virtual devices. Type: int.

Example code:

```
unsigned int * info = 0;
unsigned int infosize = 0;
infosize = KYFG_Scan(0, 0); //First scan to retrieve the number of devices connected to PC
info = (unsigned int *) malloc(sizeof(unsigned int) * infosize );
if(info != 0)
{
    KYFG_Scan(info, infosize);  // Scans for currently connected devices. Returns array with each ones pid
}
```

C#

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

Return value:
Number of connected hardware and virtual devices. Type: int.

Python

```
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) [2]

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / KYFGLIB_DLL_NOT_FOUND / Other errpr.
Number of connected hardware and virtual devices. Type: int.
pids_info - List of scanned devices. Type: list.

Remarks:

1. The software stack requires “KYService” to be running, otherwise, KYFG_Scan() will return 0.
2. If the pids_info parameter is called with NULL, the pids_info array will not be filled and the function will only return the number of connected and virtual Frame Grabbers, otherwise the array is filled with each Device Product ID (pid).

7.4 KY_DeviceScan()

Scans for KAYA devices currently connected to the PC PCIe slots and available virtual devices. [1]

C

```
FGSTATUS KY_DeviceScan(int *pDetectedDevices);
```

Parameter name	Type	Description
pDetectedDevices	int*	Pointer to int that will receive the number of available devices.

Return value:

FGSTATUS - Status and error report.

Example code:

```
        int nDetectedDevices;
        KY_DeviceScan(&nDetectedDevices);
```

C#

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

Return value:

Number of connected hardware and virtual devices.

Python

```
def KY_DeviceScan()
```

Return value:

FGSTATUS - Status and error report.

Number of connected hardware and virtual devices. Type: int

Remarks:

1. The software stack requires “KYService” to be running, otherwise, KY_DeviceScan() will return 0.

7.5 KYFG_Open()

Connects to a specific Frame Grabber and initializes all required components. [1]

C

FGHANDLE KYFG_Open(int index);

Parameter name	Type	Description
index	int	The index, from scan result array acquired by the KY_DeviceScan() function, of the Frame Grabber device to open. [2]

Return value:

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

C#

IGrabber Lib.Open(int index);

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

Return value:

Instance of Grabber class that implements IGrabber interface.

Python

def KYFG_Open(deviceIndex)

Parameter Name	Type	Description
deviceIndex	int	The index, from the scan result list acquired with the KY_DeviceScan() function, of the device to open. [2]

Return value:

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

Remarks:

1. The software stack requires “KYService” to be running, otherwise, KYFG_Open() will return INVALID_FGHANDLE.
2. When calling the function with an index of -1, a connection to the first found Frame Grabber will be established, such function call eliminates the need for KY_DeviceScan() function call.

7.6 KYFG_OpenEx()

Connects to a specific device and initializes all required components with previously saved values [1]. Project file may be passed here in order to initialize Frame Grabber and Camera parameters with previously saved values.

C

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

Parameter name	Type	Description
index	int	The index, from scan result array, acquired with KY_DeviceScan() function, of the Frame Grabber device to open. [2]
projectFile	const char*	(optional) The full path of a project file with saved values. The input value can be NULL.

Return value:
Returns an API handle to the device. INVALID_FGHANDLE will indicate a wrong, impossible or unsupported connection.

C#

`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.DeviceScan() 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.

Return value:
Instance of Grabber class that implements IGrabber interface.

Python

`def KYFG_OpenEx(deviceIndex)`

Parameter Name	Type	Description
deviceIndex	int	The index, from the scan result list acquired with the KY_DeviceScan() function, of the device to open. [1]

Return value:
Returns an API handle to the device. INVALID_FGHANDLE will indicate a wrong, impossible or unsupported connection.

Remarks:

1. A project file can be generated using our Vision Point application. See section "Creating a New Project File" of the “Vision_Point_App_User_Guide” document.
2. A project file with previously saved values can be passed in order to initialize camera parameters. For additional information regarding the project file please refer to the Vision Point application user guide: “Vision_Point_App_User_Guide”.’
3. When calling the function with an index of -1, a connection to the first found Frame Grabber will be established, such function call eliminates the need for KY_DeviceScan() function call.

7.7 KY_DeviceInfo()

Fills KY_DEVICE_INFO structure with info about the relevant device. Before calling this function you must set the field “version” to a value between 0 and the maximum number supported at the time of writing your code. See struct KY_DEVICE_INFO for details.

C

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

Parameter name	Type	Description
index	int	Device index
pInfo	KY_DEVICE_INFO*	pointer to an empty struct

Return value:

FGSTATUS - Status and error report.

C#

`DEVICE_INFO Lib.DeviceInfo(int index);`

Parameter Name	Type	Description
index	int	Discovered device index

Return value:

KY_DEVICE_INFO - Returns structure with info about the device.

Python

`def KY_DeviceInfo(index)`

Parameter Name	Type	Description
index	int	Discovered device index

Return value:

FGSTATUS - Status and error report.
KY_DEVICE_INFO - Returns structure with info about the device.

7.8 KYFG_Close()

Close the device specified by its handle. Stops data acquisition/ generation of all opened cameras, disconnects from all connected cameras and deletes previously created buffers associated with these cameras or camera simulator.

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.Close();
```

Python

```
def KYFG_Close(handle)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to a chosen device.

Return value:

FGSTATUS - Status and error report.

7.9 KYFG_Pid2Name() (DEPRECATED)

A deprecated function and is no longer supported. New applications should use KY_DeviceDisplayName().
Converts Product ID (PID) to its designated device name.

C

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

Parameter name	Type	Description
pid	unsigned int	Product id.

Return value:
The name of the Frame Grabber was issued by the specified PID.

C#

Not applicable.

Python

Not applicable.

7.10 KY_DeviceDisplayName() (DEPRECATED)

This function is deprecated. New applications should use function KY_DeviceInfo() and use pInfo.szDeviceDisplayName to retrieve device name.
Retrieve the device name for the specified index.

C

```
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.

C#

```
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.

Python

```
def KY_DeviceDisplayName(index)
```

Parameter Name	Type	Description
index	int	Discovered device index

Return value:

FGSTATUS - Status and error report.
The name of Frame Grabber issued by the specified index.

7.11 KYFG_SetGrabberConfigurationParameterCallback() (C++ only)

Registers a parameter callback function. This function will be called during execution of KYFG_GetGrabberConfigurationParameterDefinitions() with pointer to NodeDescriptor. Additionally, a registered user context pointer is retrieved which consequently can be interpreted by the 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. Afterward, this pointer is retrieved when the callback is issued.

Return value:

FGSTATUS - Status and error report.

7.12 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 newly acquired frame and copying it to a local buffer:

```
FGHANDLE handle[MAXBOARDS];

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();
        }
}
```

8 Camera Configurations

8.1 KYFG_CameraScan() (DEPRECATED)

This function is deprecated. New applications should use function KYFG_UpdateCameraList(). Projects should be loaded using function KYFG_OpenEx().
The Frame Grabber scans for connected cameras, establishes a connection and defines the default speed for each camera, on every connected channel.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
camHandleArray	CAMHANDLE*	An 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[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
int detectedCameras[MAXBOARDS];
…
KYFG_CameraScan(handle[grabberIndex], camHandleArray[grabberIndex],
&detectedCameras[grabberIndex]);
printf("Found %d cameras connected to Frame Grabber", detectedCameras [grabberIndex]);
…
```

C#

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

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

Python

```
def KYFG_CameraScan(handle)
```

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

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.

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

8.2 KYFG_CameraScanEx

The Frame Grabber scans for connected cameras, or performs partial re-detection depending on previously defined 'bRetainOpenCameras' parameter, allowing to skip currently active links and detect only new connections, establishing a connection and defining the default speed for each camera, on every connected channel. In the case of generation mode, this function is used to retrieve the number of cameras implemented by a given Chameleon Simulator, and fill the array with their API handles.

NOTE: *Currently only one camera is implemented by Chameleon Simulator.*

C

```
FGSTATUS KYFG_CameraScanEx(FGHANDLE handle, KYFGLib_CameraScanParameters* pScanParams)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
pScanParams	KYFGLib_CameraScanParameters*	Pointer to camera scan parameters structure

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.

C#

Not applicable.

Python

```
def KYFG_CameraScanEx(handle, bRetainOpenCameras)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber.
bRetainOpenCameras	bool	TRUE- if scan should skip currently active links and detect only new connections. FALSE- if all currently open camera handles should be reset and full re-scan should be performed.

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.
Camera scan parameters structure. Type: KYFGLib_CameraScanParameters.

Example code:

```
(status, ScanParameters) = KYFG_CameraScanEx(handle, True)
print("KYFG_CameraScanEx version: ", ScanParameters.version)
print("KYFG_CameraScanEx nCameraCount: ", ScanParameters.nCameraCount)
print("KYFG_CameraScanEx bRetainOpenCameras: ", ScanParameters.bRetainOpenCameras)
for i in range(0, ScanParameters.nCameraCount):
        print("KYFG_CameraScanEx pCamHandleArray: ", ScanParameters.pCamHandleArray[i])
```

8.3 KYFG_UpdateCameraList()

The behavior of this function is similar to the ‘KYFG_CameraScanEx’ function, with an addition of setting the ‘‘bRetainOpenCameras’’ parameter to KYTRUE. This means that open camera handles will not be affected by this call and will be retained at the same places of the array where they were returned by the previous call, except for camera(s) that were closed between calls.

C

```
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 an integer. Must be set to the number of elements allocated in the 'pCamHandleArray'. After successful function return indicates the number of elements that were filled

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.

C#

```
System.Collections.Generic.List<ICamera> IDevice.UpdateCameraList();
```

Return value:
List of API camera handles of detected cameras.

Python

```
def KYFG_UpdateCameraList(handle)
```

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

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.
List of API camera handles of detected cameras.

8.4 KYFG_CameraOpen() (DEPRECATED)

This function is deprecated. New applications should use the function KYFG_CameraOpen2(). The project should be loaded using the function KYFG_OpenEx().

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
xml_file_path	const char*	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. [1]
project_file_path	const char*	(optional) Path to previously saved values file. Value can be NULL. [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 to retrieve the 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 the project file please refer to the Vision Point application user guide: "Vision_Point_App_User_Guide".

C#

Not applicable.

Python

Not applicable.

8.5 KYFG_CameraOpen2()

Acquisition/Generation mode: Opens a connection to the chosen camera and retrieves native XML/loads built-in XML file or uses external XML file provided to override the native one. The project should be loaded using the function KYFG_OpenEx().

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
xml_file_path	const char*	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. [1]

Return value:

FGSTATUS - Status and error report. INPUT_ARGUMENT_TYPE_ERROR - value will indicate that the camHandle or xml_file_path are invalid.

C#

```
void ICamera.Open(System::String xml_file_path);
```

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. [1]

Python

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

Parameter Name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera.
xml_file_path	str	Path to override XML file. If NULL, the native XML file from the camera will be retrieved. [1]

Return value:

FGSTATUS - Status and error report. INPUT_ARGUMENT_TYPE_ERROR - value will indicate that the camHandle or xml_file_path are invalid.

Remarks:

1. An XML file can be loaded to override the native XML of the camera. Otherwise, NULL should be passed to retrieve the camera's native XML file. If 'xml_file_path' is 0 then only internal XML will be used providing a minimal set of mandatory camera parameters.

8.6 KYFG_CameraInfo() (DEPRECATED)

This function is deprecated. New applications should use the function KYFG_CameraInfo2().
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().

C

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

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

Return value:
FGSTATUS - Status and error report.

Example code:

```
CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
KYFGCAMERA_INFO cameraInfo;
KYFG_CameraInfo(camHandleArray[grabberIndex], &cameraInfo);
printf(“Camera model: %s, its manufacturer is %s”,
        cameraInfo.deviceModelName, cameraInfo.deviceVendorName);
```

Python

```
def KYFG_CameraInfo(camHandle)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera.

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))
```

Return value:
FGSTATUS - Status and error report.
Information about the chosen camera. Type: KYFGCAMERA_INFO

8.7 KYFG_CameraInfo2()

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().

C

```
FGSTATUS KYFG_CameraInfo2(CAMHANDLE camHandle, KYFGCAMERA_INFO2 *info);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
info	KYFGCAMERA_INFO2*	Pointer to camera information structure

Example code:

```
CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
KYFGCAMERA_INFO2 cameraInfo;
…
cameraInfo.version = 1;
KYFG_CameraInfo2(camHandleArray[grabberIndex], &cameraInfo);
printf(“Camera model: %s, its manufacturer is %s”,
        cameraInfo.deviceModelName, cameraInfo.deviceVendorName);
…
```

Return value:

FGSTATUS - Status and error report.

C#

```
CAMERA_INFO ICamera.KYFG_CameraInfo();
```

Return value:

Information about the chosen camera. Type: KYFGCAMERA_INFO2 [1]

Python

```
def KYFG_CameraInfo2(camHandle)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera

Example code:

```
(Status, camInfo) = KYFG_CameraInfo2(camHandle)
print("version: ", str(camInfo.version))
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))
```

Return value:

FGSTATUS - Status and error report.

Information about the chosen camera. Type: KYFGCAMERA_INFO2 [1]

Remarks:

1. Users who want this member to be considered by KYFG_CameraInfo2 call, must set the version value to 0.

8.8 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 the buffer. The size (in bytes) and file type (.xml or .zip) are also retrieved.

C

```
FGSTATUS KYFG_GetXML(CAMHANDLE camHandle,
                     char** buffer,
                     uint64_t*bufferSize,
                     KYBOOL *isZipFile);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
buffer	char**	Pointer to pointer of the buffer that will hold the camera's native XML file. [1]
bufferSize	uint64_t*	A 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:

1. This function allocates memory to hold the content of the camera native XML file, therefore the caller is responsible for releasing this memory.
2. Note for Visual Studio users: If your code is built with a compiler other than VS 2017, there could be a runtime library conflict issue. The pointer might become corrupted and the free() function might cause a crash. To avoid this issue please use KYFG_CameraGetXML().

Example code:

```
    CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS] // max KY_MAX_CAMERAS connected
    char* buffer;
    uint64_t bufferSize = 0;
    KYBOOL isZip = KYFALSE;
    FILE* fileOut = NULL;
    If(FGSTATUS_OK == KYFG_GetXML(camHandleArray[grabberIndex], &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
   }
```

C#

Not applicable.

Python

Not applicable.

8.9 KYFG_CameraGetXML()

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

C

```
FGSTATUS KYFG_CameraGetXML(CAMHANDLE camHandle,
                            char* buffer,
                           KYBOOL *isZipFile,
                           uint64_t *bufferSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
buffer	char*	Pointer to user allocated buffer. [1]
isZipFile	KYBOOL*	Pointer to indicator whether the camera's XML file is in ZIP or XML format.
bufferSize	[in,out] uint64_t*	Pointer to the size of the buffer. [2]

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. On entry, bufferSize [in] should be set to the size of the allocated buffer. After the function returns, bufferSize [out] will hold the actual size of the content extracted.
2. The bufferSize of the ZIP file will indicate the required buffer size and type of camera XML file. If the value is smaller than the needed size, or the buffer value is NULL, the buffer will not be filled.

Example code:

```
CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
char* buffer;
uint64_t 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[grabberIndex], NULL,
&isZip, &bufferSize))
{
        buffer = (char*)malloc(bufferSize);   // allocate memory for buffer
        // extract camera's native XML file
        if(FGSTATUS_OK == KYFG_CameraGetXML(camHandleArray[grabberIndex], 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
}
```

C#

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

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 code:

```
        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);
        }
```

Python

```
def KYFG_CameraGetXML(camHandle)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE	API handle to the 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()
```

8.10 KYFG_CameraClose()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera

Return value:

FGSTATUS - Status and error report.

C#

```
void ICamera.Close()
```

Python

```
def KYFG_CameraClose(camHandle)
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera

Return value:

FGSTATUS - Status and error report.

8.11 KYFG_SetCameraConfigurationParameterCallback() (C++ only)

Registers a parameter callback function. This function will be called during execution of KYFG_GetCameraConfigurationParameterDefinitions() with pointer to NodeDescriptor. Additionally, a registered user context pointer is retrieved which consequently can be interpreted by the 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. Afterward, this pointer is retrieved when a callback is issued.

Return value:
FGSTATUS - Status and error report.

8.12 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_GetCameraConfigurationParameterDefinitions().

9 Callback Functions

9.1 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 newly received frame of a valid stream, with appropriate BUFFHANDLE. Callback call is not necessarily serialized, which means different streams might generate concurrent calls before the end of the previous callback execution. Use the Buffer Interface functions to handle received data. Additionally, a registered user context pointer is retrieved which consequently can be interpreted by the host application for internal use.

C

```
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 the callback function
userContext	void*	(optional) Pointer to user context. Afterward, this pointer is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

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

Parameter Name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	*see remarks	Pointer to callback function
userContext	int – currently 0	(Optional) User context. This value is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(buffHandle, userContext)
```

buffHandle - API handle to a stream. Type: BUFFHANDLE
userContext - User context registered using KYFG_CallbackRegister

Example code:

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

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

print('Good callback buffer handle: ' + str(format(buffHandle, '02x')) + ", current index: " + str(buffIndex)
+ ", total frames: " + str(totalFrames) + "         ", end='\r')
sys.stdout.flush()
return
# register stream callback function “Stream_callback_func” for all streams related to selected Frame
Grabber
KYFG_CallbackRegister(fgHandle, Stream_callback_func, 0)
```

9.2 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.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	FGCallback	Pointer to the callback function

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def KYFG_CallbackUnregister(handle, userFunc)
```

Parameter Name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	*see remarks	Pointer to callback function

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(buffHandle, userContext)
```

buffHandle - API handle to a stream. Type: BUFFHANDLE
userContext - User context registered using KYFG_CallbackRegister

9.3 KYFG_CameraCallbackRegister()

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

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

```
void ICamera.CameraCallbackRegister(CameraCallback delegator, Object userContext);
```

Overloads:

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

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

Python

```
def KYFG_CameraCallbackRegister(camHandle, userFunc, userContext)
```

Parameter Name	Type	Description
camHandle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	*see remarks	Pointer to callback function
userContext	int – currently 0	(Optional) User context. Afterward, this value is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYFG_CameraCallbackRegister

9.4 KYFG_CameraCallbackUnregister()

Unregisters a previously registered camera runtime acquisition callback function.

C

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

Parameter name	Type	Description
handle	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 newly acquired frame and copying it to local buffer:

```
void Stream_callback_func(void* userContext, BUFFHANDLE buffHandle)
{
        static void* data = NULL;
        static KYBOOL copyingDataFlag = KYFALSE;
        uint64_t 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[MAXBOARDS][KY_MAX_CAMERAS]
        // maximum KY_MAX_CAMERAS cameras can be connected
        int nDetectedCameras = 0;
        KYFG_CameraScan(handle, camHandleArray, &nDetectedCameras);
        if (nDetectedCameras > 0 )
        {
            KYFG_CameraCallbackRegister(camHandleArray[grabberIndex],
        Stream_callback_func, NULL);
        }
        while(1){}
        return 0;
    }
```

C#

```
void ICamera.CameraCallbackUnregister(CameraCallback delegator);
```

Parameter Name	Type	Description
delegator	CameraCallback	Delegator to callback function

Python

```
def KYFG_CameraCallbackUnregister(camHandle, userFunc)
```

Parameter Name	Type	Description
camHandle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	*see remarks	Pointer to callback function

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYFG_CameraCallbackRegister

9.5 KYFG_StreamBufferCallbackRegister()

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

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

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

Overloads:

```
void IStream.BufferCallbackRegister(StreamBufferCallback^ delegator)
```

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

Python

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

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream
userFunc	*see remarks	Pointer to callback function
userContext	*see example code	User context. Afterward, this pointer is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYFG_StreamBufferCallbackRegister

Example code:

```
# User context class
class StreamInfoStruct:
def __init__(self):
self.width = 0
self.height = 0
self.callbackCount = 0
return

# User callback function according to the prototype
def Stream_callback_func(buffHandle, userContext):
streamInfo = cast(userContext, py_object).value
print('buffer' + str(format(buffHandle, '02x')) + ': height=' + str(streamInfo.height) + ', width=' +
str(streamInfo.width) + ', callback count=' + str(streamInfo.callbackCount))
streamInfo.callbackCount = streamInfo.callbackCount + 1
return

# Create and init user context class and register callback function
streamInfoStruct = StreamInfoStruct()
streamInfoStruct.width = KYFG_GetCameraValueInt(camHandleArray[grabberIndex][0], "Width")
streamInfoStruct.height = KYFG_GetCameraValueInt(camHandleArray[grabberIndex][0], "Height")

(KYFG_StreamBufferCallbackRegister_status,) =
KYFG_StreamBufferCallbackRegister(cameraStreamHandle,Stream_callback_func,
py_object(streamInfoStruct))
```

9.6 KYFG_StreamBufferCallbackUnregister()

Unregisters a previously registered stream callback function.

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

```
void IStream.BufferCallbackUnregister(StreamBufferCallback delegator)
```

Parameter Name	Type	Description
delegator	StreamBufferCallback	Delegator to callback function

Python

```
def KYFG_StreamBufferCallbackUnregister(streamHandle, userFunc)
```

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream.
userFunc	*see remarks	Pointer to the callback function.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYFG_StreamBufferCallbackRegister

9.7 KYFG_AuxDataCallbackRegister()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to the chosen device.[1]
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 the function call in the host application

Return value:

FGSTATUS - Status and error report.

C#

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

Overloads:

```
void IGrabber.AuxDataCallbackRegister(FGAuxDataCallback delegator)
```

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.

Python

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

Parameter Name	Type	Description
handle	FGHANDLE	API handle to the connected device.
userFunc	*see remarks	Pointer to callback function implementation.
userContext	* User context object	User context. Afterward, this pointer is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYFG_AuxDataCallbackRegister

9.8 KYFG_AuxDataCallbackUnregister()

Unregister previously registered run-time auxiliary data callback function. [2]

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	FGAuxDataCallback	Callback function prototype[1]

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.AuxDataCallbackUnregister(FGAuxDataCallback delegator)
```

Parameter Name	Type	Description
delegator	FGAuxDataCallback	Callback delegate function

Python

```
def KYFG_AuxDataCallbackUnregister(handle, userFunc)
```

Parameter Name	Type	Description
handle	FGHANDLE	API handle to the connected device.
userFunc	*see remarks	Pointer to the callback function to unregister.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

   ```
   def  <user_stream_callback>(streamHandle, userContext)
   ```

   streamHandle - API handle to a stream. Type: STREAM_HANDLE
   userContext - User context registered using KYFG_AuxDataCallbackRegister
2. Since several Auxiliary data retrieval functions may be registered, the userFunc parameter should be passed to the un-registering function to determine which specific function to un-register.

9.9 KYDeviceEventCallBackRegister()

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

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.EventCallBackRegister(DeviceEventCallBack^ deviceEventDelegator);
```

Python

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

Parameter Name	Type	Description
handle	FGHANDLE	API handle to the connected device.
userFunc	*see remarks	Pointer to callback function implementation.
userContext	* User context object	User context. Afterward, this pointer is retrieved when the callback is issued. Helps to determine the origin of the stream in the host application.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The callback function prototype should be as follows:

```
def  <user_stream_callback>(streamHandle, userContext)
```

streamHandle - API handle to a stream. Type: STREAM_HANDLE
userContext - User context registered using KYDeviceEventCallBackRegister

9.10 KYDeviceEventCallBackUnregister()

Unregisters a previously registered user runtime callback function.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
userFunc	FGAuxDataCallback	Callback function prototype.[1],[2]

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.EventCallBackUnregister(DeviceEventCallBack^ deviceEventDelegator);
```

Python

```
def KYDeviceEventCallBackUnregister(handle, userFunc)
```

Parameter Name	Type	Description
handle	FGHANDLE	API handle to the connected device.
userFunc	*see remarks	Pointer to the callback function to unregister. See remarks.

Return value:

FGSTATUS - Status and error report.

Remarks:

1. Since several callback functions may be registered, the userFunc parameter should be passed to the un-registering function to determine which specific function to un-register.
2. Any of the callback functions described in this section should perform minimally necessary tasks and return as soon as possible, avoiding long-running I/O operations. For example, KYFG_CameraWriteReg is an I/O operation that involves signal round-trip to the camera and waiting for the camera's acknowledgment. It is advised to move the operations to a separated thread using a function, such as KYFG_CameraWriteReg, for a direct write data buffer to the selected camera.
3. The callback function prototype should be as follows:

   ```
   def  <user_stream_callback>(streamHandle, userContext)
   ```

   streamHandle - API handle to a stream. Type: STREAM_HANDLE
   userContext - User context registered using KYDeviceEventCallBackRegister

10 Camera Parameters - Setters

General remarks:

1. KYFG_SetCameraValue() and all of its sub functions are used to handle camera stream specific parameters (e.g camera stream RX packets). For setting camera stream-specific parameters, the CameraSelector should be first chosen.
2. Camera specific parameters can be extracted from internal or external camera xml file.

10.1 KYFG_SetCameraValue()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value:

FGSTATUS - Status and error report.

C#

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

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

Python

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

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

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.1 KYFG_SetCameraValueInt()

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

C

```
FGSTATUS KYFG_SetCameraValueInt(CAMHANDLE camHandle, const char *paramName, int64_t value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
value	Int64_t	Enumeration value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueInt(camHandle, paramName, value)
```

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

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.2 KYFG_SetCameraValueFloat()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
value	double	The floating-point value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueFloat(camHandle, paramName, value)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.
value	double	The floating-point value of the chosen parameter configuration.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.3 KYFG_SetCameraValueBool()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
value	KYBOOL	The boolean value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueBool(camHandle, paramName, value)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.
value	boolean	The boolean value of the chosen parameter configuration.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.4 KYFG_SetCameraValueEnum()

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

C

```
FGSTATUS KYFG_SetCameraValueEnum(CAMHANDLE camHandle, const char *paramName, int64_t value);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
value	int64_t	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[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
…
int64_t pixel_format_value = 0x101;                    // mono 8bit format
KYFG_SetCameraValueEnum(camHandleArray[grabberIndex],
“PixelFormat”,  pixel_format_value);
```

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueEnum(camHandle, paramName, value)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.
value	int	The enumeration value of chosen parameter configuration.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.5 KYFG_SetCameraValueRegister()

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

C

Use KYFG_SetCameraValue() function.

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueRegister(camHandle, paramName, value)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera
paramName	str	Name of the configuration parameter
value	bytearray	Data to write to the chosen register.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.6 KYFG_ExecuteCommand() (DEPRECATED)

This function is deprecated. New applications should use KYFG_CameraExecuteCommand().
Execute camera command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
FGSTATUS - Status and error report.

C#

Not applicable.

Python

Not applicable.

10.1.7 KYFG_CameraExecuteCommand()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
FGSTATUS - Status and error report.

C#

`void ICamera.ExecuteCommand(System.String paramName);`

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

Python

`def KYFG_CameraExecuteCommand(camHandle, paramName)`

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of command.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.8 KYFG_SetCameraValueString()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
value	const char*	The string value of chosen camera configuration

Return value:
FGSTATUS - Status and error report.

C#

Use ICamera.SetValue() function.

Python

```
def KYFG_SetCameraValueString(camHandle, paramName, value)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.
value	str	The string value of the chosen parameter configuration.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

10.1.9 KYFG_SetCameraValueEnum_ByValueName()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
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[MAXBOARDS][KY_MAX_CAMERAS]
        // maximum KY_MAX_CAMERAS cameras can be connected

        KYFG_SetCameraValueEnum_ByValueName(camHandleArray[grabberIndex], “AcquisitionMode”,
                                            “Continuous”);
```

C#

```
void ICamera.SetValueEnum_ByValueName(const System::String^ paramName, const System::String^ param);
```

Parameter Name	Type	Description
paramName	System.String	Name of configuration parameter
paramValue	System.String	Name of parameter enumeration choice

Python

```
def KYFG_SetCameraValueEnum_ByValueName(camHandle, paramName, paramValueName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter
paramValueName	str	Name of parameter enumeration choice

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

11 Camera Parameters - Getters

General remarks:

1. KYFG_GetCameraValue() and all of its sub functions are used to handle camera stream specific parameters (e.g camera stream RX packets). For getting camera stream-specific parameters, the CameraSelector should be first chosen.
2. Camera specific parameters can be extracted from internal or external camera xml file.

11.1 KYFG_GetCameraValueType()

Get camera configuration field type.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
KY_CAM_PROPERTY_TYPE - Parameter type information.

C#

Not applicable.

Python

```
def KYFG_GetCameraValueType(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - Status and error report. INPUT_ARGUMENT_TYPE_ERROR - value will indicate that the camHandle or paramName is invalid.
value_type - Parameter type information. Type: KY_CAM_PROPERTY_TYPE

11.2 KYFG_GetCameraValue()

Get camera configuration field value.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value:

FGSTATUS - Status and error report.

C#

`System.Object ICamera.GetValue(System.String paramName);`

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 be one of the following types: int, Boolean, String, Float, String or Tuple (see remarks). [1]

Python

`def KYFG_GetCameraValue(camHandle, paramName)`

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - Status and error report.

paramValue - The value of the required parameter. Can be be one of the following types: int, Boolean, String, Float, String or Tuple (see remarks). [2]

Remarks:

1. 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.
2. In case of PROPERTY_TYPE_ENUM, the tuple includes 3 elements: status, paramValueStr and paramValueInt, where paramValueStr and paramValueInt represent the required enum entry.

11.2.1 KYFG_GetCameraValueInt()

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

C

```
int64_t KYFG_GetCameraValueInt(CAMHANDLE camHandle, const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
An integer value of camera configuration field of integer type. INVALID_INT_PARAMETER_VALUE will be returned In case of an error.

C#

Use ICamera.GetValue() function.

Python

```
def KYFG_GetCameraValueInt(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
paramValue - Integer value of chosen parameter configuration field of integer type.

11.2.2 KYFG_GetCameraValueEnum()

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

C

```
int64_t KYFG_GetCameraValueEnum(CAMHANDLE camHandle, const char *paramName);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

An integer value of camera configuration field of integer type. INVALID_INT_PARAMETER_VALUE will be returned In case of an error.

Example code:

```
        CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS]
        // maximum KY_MAX_CAMERAS cameras can be connected
        int64_t linkconfig = 0;
        linkconfig = KYFG_GetCameraValueEnum(camHandleArray[grabberIndex], “LinkConfig”);
```

C#

Use ICamera.GetValue() function.

Python

```
def KYFG_GetCameraValueEnum(camHandle, paramName, value)
```

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

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error. The integer value of camera configuration field of integer type.

11.2.3 KYFG_GetCameraValueRegister()

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

C

```
FGSTATUS KYFG_GetCameraValueRegister(
        CAMHANDLE camHandle,
        const char *paramName,
        uint8_t *bufferPtr,
        uint32_t *bufferSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
bufferPtr	uint8_t*	A buffer that will be filled with the value of the chosen parameter
bufferSize	uint32_t*	Size of buffer to fill

Return value:

FGSTATUS - FGSTATUS_OK / FGSTATUS_BUFFER_TOO_SMALL- size of provided buffer is too small to hold the value completely.

C#

Use ICamera.GetValue() function.

Python

```
def KYFG_GetCameraValueRegister(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
buffer_size - the size of the buffer holding the data
buffer - The data from the register of chosen parameter configuration. Type: bytearray

11.2.4 KYFG_GetCameraValueFloat()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
The floating-point value of camera configuration field of Float type. INVALID_FLOAT_PARAMETER_VALUE will be returned in case of an error.

C#

Use ICamera.GetValue() function.

Python

```
def KYFG_GetCameraValueFloat(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / INVALID_FLOAT_PARAMETER_VALUE / Other errors will be returned in case of an error.
paramValue - Floating point value of chosen parameter configuration field of Float type.

11.2.5 KYFG_GetCameraValueBool()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

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

C#

Use ICamera.GetValue() function.

Python

```
def KYFG_GetCameraValueBool(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
paramValue - The boolean value of the chosen parameter configuration field of Boolean type.

11.2.6 KYFG_GetCameraValueString() (DEPRECATED)

This function is deprecated. KYFG_GetCameraValueStringCopy() should be used by new applications.
Get camera configuration field value of String type. According to Gen<i>Cam standard naming and xml field definition and type. [1]

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
ptr	char**	Pointer to the size of user allocated memory for the chosen parameter

Return value:
FGSTATUS - Status and error report.

Remarks:
1. This function allocates memory for the char array and the caller is responsible for releasing this memory using the free() function.
2. Note for Visual Studio users:
   In case your code is built with a compiler other than VS 2017, there could be a runtime library conflict issue. The pointer might become corrupted and the free() function might cause a crash. To avoid this issue please use KYFG_GetCameraValueStringCopy() .

Example code:

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

C#

Not applicable.

Python

```
def KYFG_GetCameraValueString(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
paramStr - String value of chosen parameter configuration field of String type.

11.2.7 KYFG_GetCameraValueStringCopy()

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

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
StringPtr	Char*	Pointer user char array that will be filled with the value of the chosen parameter
stringSize [in,out]	unsigned int*	Pointer to size value of the file to be extracted. [1],[2],[3]

Return value:

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

Remarks:

1. Value is copied to a user-allocated char array.
2. stringSize [in] value will determine the size of the provided char array. stringSize [out] will hold the actual size of the string to extract.
3. If the 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.
4. stringSize should reflect the actual size of the provided char array otherwise it might cause a severe crash.

Example code:

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

C#

`System.String ICamera.GetValueString(System.String paramName);`

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

String value of chosen parameter configuration field of String type. Type: System::String

Python

`def KYFG_GetCameraValueStringCopy(camHandle, paramName)`

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. FGSTATUS_BUFFER_TOO_SMALL – value will indicate that in function calculated buffer size is too small to hold the requested string value. (The size is determined by additional function call and the required buffer is created according to this size). INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of another error.
paramStr - String value of chosen parameter configuration field of String type.

11.3 KYFG_GetCameraValueIntMaxMin()

Get camera maximum and minimum configuration field values of Integer type. According to Gen<i>Cam standard naming and xml field definition and type.

C

```
FGSTATUS KYFG_GetCameraValueIntMaxMin(
        CAMHANDLE camHandle,
        const char *paramName,
        int64_t* pIntMax,
        int64_t* pIntMin);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
pIntMax	int64_t*	A pointer that will be filled with the value of the chosen parameter
pIntMin	int64_t*	A pointer that will be filled with the value of the chosen parameter

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.

C#

```
System.Object ICamera.GetValueIntMaxMin(System.String paramName);
```

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

Object, that represents a param value corresponding to the param name. Type: Tuple (see remarks). [1]

Python

```
def KYFG_GetCameraValueIntMaxMin(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
value_int_max - Maximum configuration value of Integer type field. Type: int
value_int_min - Minimum configuration value of Integer type field. Type: int

Remarks:

1. Object will be the instance of:

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

11.4 KYFG_GetCameraValueFloatMaxMin()

Get camera maximum and minimum configuration field values of Float type. According to Gen<i>Cam standard naming and xml field definition and type.

C

```
FGSTATUS KYFG_GetCameraValueFloatMaxMin(
        CAMHANDLE camHandle,
        const char *paramName,
        double* pDoubleMax,
        double* pDoubleMin);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the chosen camera. See general remarks.
paramName	const char*	Name of configuration parameter
pDoubleMax	double*	A pointer that will be filled with the value of the chosen parameter
pDoubleMin	double*	A pointer that will be filled with the value of the chosen parameter

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.

C#

```
System.Object ICamera.GetValueFloatMaxMin(System.String paramName);
```

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

Object, that represents a param value corresponding to the param name. Type: Tuple (see remarks). [1]

Python

```
def KYFG_GetCameraValueFloatMaxMin(camHandle, paramName)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
value_float_max - Maximum configuration value of Float type field. Type: Double
value_float_min - Minimum configuration value of Float type field. Type: Double

Remarks:

1. Object will be the instance of:

```
System.Tuple< System.Double, System.Double>
```

12 Frame Grabber Parameters - Setters

General remarks:

1. KYFG_SetGrabberValue() and all of its sub functions are used to handle general Frame Grabber configurations (e.g GPIO configurations). For setting camera stream-specific parameters, the CameraSelector should be first chosen.
2. Please refer to the “KAYA Frame Grabber Feature Guide” and “Chameleon Simulator Feature Guide” documents for the full parameters list and examples.

12.1 KYFG_SetGrabberValue()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value:

FGSTATUS - Status and error report.

C#

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

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

Python

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

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	According to the parameter type	The value of the parameter to set.

Return value:

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

12.1.1 KYFG_SetGrabberValueInt()

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

C

```
FGSTATUS KYFG_SetGrabberValueInt(FGHANDLE handle, const char *paramName, int64_t value);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
value	Int64_t	Enumeration value of chosen camera configuration

Return value:
FGSTATUS - Status and error report.

C#

Use IDevice.SetValue() function.

Python

```
def KYFG_SetGrabberValueInt(handle, paramName, value)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	int	The value of the parameter to set.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.2 KYFG_SetGrabberValueFloat()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
value	double	The floating-point value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use IDevice.SetValue() function.

Python

```
def KYFG_SetGrabberValueFloat(handle, paramName, value)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	double	The floating-point value of the chosen parameter configuration.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.3 KYFG_SetGrabberValueBool()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
value	KYBOOL	The boolean value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use IDevice.SetValue() function.

Python

```
def KYFG_SetGrabberValueBool(handle, paramName, value)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	boolean	The boolean value of the chosen parameter configuration.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.4 KYFG_SetGrabberValueEnum()

Set 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.

C

```
FGSTATUS KYFG_SetGrabberValueEnum(FGHANDLE handle, const char *paramName, int64_t value);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
value	int64_t	Enumeration value of chosen camera configuration

Return value:
FGSTATUS - Status and error report.

C#

Use IDevice.SetValue() function.

Python

```
def KYFG_SetGrabberValueEnum(handle, paramName, value)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	int	The enumeration value of chosen parameter configuration.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.5 KYFG_ExecuteGrabberCommand() (DEPRECATED)

This function is deprecated. New applications should use KYFG_GrabberExecuteCommand().
Execute Frame Grabber command; applicable for values of Command type. According to Gen<i>Cam standard naming and xml field definition and type.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
FGSTATUS - Status and error report.

C#

Not applicable.

Python

Not applicable.

12.1.6 KYFG_GrabberExecuteCommand()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.ExecuteCommand(System.String paramName);
```

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

Python

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

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

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.7 KYFG_SetGrabberValueString()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
value	const char*	The string value of chosen camera configuration

Return value:

FGSTATUS - Status and error report.

C#

Use IDevice.SetValue() function.

Python

`def KYFG_SetGrabberValueString(handle, paramName, value)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
value	str	The string value of the chosen parameter configuration.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

12.1.8 KYFG_SetGrabberValueEnum_ByValueName()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
paramValueName	const char*	Name of parameter enumeration choice

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.SetValueEnum_ByValueName(const System::String^ paramName, const System::String^ param);
```

Parameter Name	Type	Description
paramName	System.String	Name of configuration parameter
paramValue	System.String	Name of parameter enumeration choice

Python

```
def KYFG_SetGrabberValueEnum_ByValueName(handle, paramName, paramValueName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber.
paramName	str	Name of the configuration parameter.
paramValueName	str	Name of parameter enumeration choice.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

13 Frame Grabber Parameters - Getters

General remarks:

1. KYFG_GetGrabberValue() and all of its sub functions are used to handle general Frame Grabber configurations (e.g GPIO configurations). For getting camera stream-specific parameters, the CameraSelector should be first chosen.
2. Please refer to the “KAYA Frame Grabber Feature Guide” and “Chameleon Simulator Feature Guide” documents for the full parameters list and examples.

13.1 KYFG_GetGrabberValueType()

Get Frame Grabber configuration field type.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
KY_CAM_PROPERTY_TYPE - Parameter type information.

C#

Not applicable.

Python

`def KYFG_GetGrabberValueType(handle, paramName)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - Status and error report. INPUT_ARGUMENT_TYPE_ERROR - value will indicate that the handle or paramName is invalid.
value_type - Parameter type information. Type: KY_CAM_PROPERTY_TYPE

13.2 KYFG_GetGrabberValue()

Get Frame Grabber configuration field value.

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
paramValue	void*	Pointer to camera configuration value

Return value:

FGSTATUS - Status and error report.

C#

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

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 be one of the following types: int, Boolean, String, Float, String or Tuple (see remarks). [1]

Python

`def KYFG_GetGrabberValue(handle, paramName)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - Status and error report.

paramValue - The value of the required parameter. Can be be one of the following types: int, Boolean, String, Float, String or Tuple (see remarks). [2]

Remarks:

1. 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.
2. In case of PROPERTY_TYPE_ENUM, the tuple includes 3 elements: status, paramValueStr and paramValueInt, where paramValueStr and paramValueInt represent the required enum entry.

13.2.1 KYFG_GetGrabberValueInt()

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

C

`int64_t KYFG_GetGrabberValueInt(FGHANDLE handle, const char *paramName);`

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:
An integer value of camera configuration field of integer type. INVALID_INT_PARAMETER_VALUE will be returned In case of an error.

C#

Use IDevice.GetValue() function.

Python

`def KYFG_GetGrabberValueInt(handle, paramName)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
paramValue - Integer value of chosen parameter configuration field of integer type.

13.2.2 KYFG_GetGrabberValueEnum()

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

C

```
int64_t KYFG_GetGrabberValueEnum(FGHANDLE handle, const char *paramName);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

An integer value of camera configuration field of integer type. INVALID_INT_PARAMETER_VALUE will be returned In case of an error.

C#

Use IDevice.GetValue() function.

Python

```
def KYFG_GetGrabberValueEnum(handle, paramName, value)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber
paramName	str	Name of the configuration parameter

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
The integer value of camera configuration field of integer type.

13.2.3 KYFG_GetGrabberValueRegister()

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

C

```
FGSTATUS KYFG_GetGrabberValueRegister(
        FGHANDLE handle,
        const char *paramName,
        uint8_t *bufferPtr,
        uint32_t *bufferSize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
bufferPtr	uint8_t*	A buffer that will be filled with the value of the chosen parameter
bufferSize	uint32_t*	Size of buffer to fill

Return value:
FGSTATUS - FGSTATUS_OK / FGSTATUS_BUFFER_TOO_SMALL- size of provided buffer is too small to hold the value completely / Other error.

C#

Use IDevice.GetValue() function.

Python

```
def KYFG_GetGrabberValueRegister(handle, paramName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber.
paramName	str	Name of the configuration parameter.

Return value:
FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
buffer_size - the size of the buffer holding the data
buffer - The data from the register of chosen parameter configuration. Type: bytearray

13.2.4 KYFG_GetGrabberValueFloat()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

The floating-point value of camera configuration field of Float type. INVALID_FLOAT_PARAMETER_VALUE will be returned in case of an error.

C#

Use IDevice.GetValue() function.

Python

```
def KYFG_GetGrabberValueFloat(handle, paramName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / INVALID_FLOAT_PARAMETER_VALUE / Other error.

paramValue - Floating point value of chosen parameter configuration field of Float type.

13.2.5 KYFG_GetGrabberValueBool()

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

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter

Return value:

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

C#

Use IDevice.GetValue() function.

Python

`def KYFG_GetGrabberValueBool(handle, paramName)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

paramValue - The boolean value of the chosen parameter configuration field of Boolean type.

13.2.6 KYFG_GetGrabberValueString() (DEPRECATED)

This function is deprecated. KYFG_GetGrabberValueStringCopy() should be used by new applications.
Get Frame Grabber configuration field value of String type. According to Gen<i>Cam standard naming and xml field definition and type [1].

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
ptr	char**	Pointer to the size of user allocated memory for the chosen parameter

Return value:

FGSTATUS - Status and error report.

Remarks:

1. This function allocates memory for the char array and the caller is responsible for releasing this memory using the free() function.
2. Note for Visual Studio users:
   In case your code is built with a compiler other than VS 2017, there could be a runtime library conflict issue. The pointer might become corrupted and the free() function might cause a crash. To avoid this issue please use KYFG_GetGrabberValueStringCopy().

C#

Not applicable.

Python

`def KYFG_GetGrabberValueString(handle, paramName)`

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
paramStr - String value of chosen parameter configuration field of String type.

13.2.7 KYFG_GetGrabberValueStringCopy()

Get Frame Grabber configuration value of String type field. Please see the remarks!

C

```
FGSTATUS KYFG_GetGrabberValueStringCopy(
        FGHANDLE handle,
        const char *paramName,
        char *stringPtr,
        unsigned int *stringSize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
stringPtr	char*	Pointer user char array that will be filled with the value of the chosen parameter
stringSize [in,out]	unsigned int*	Pointer to size value of the file to be extracted. [1],[2],[3]

Return value:

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. (The size is determined by additional function call and the required buffer is created according to this size)
At function return, stringSize will hold the desired string length including NULL termination character.

Remarks:

1. Value is copied to a user-allocated char array.
2. stringSize [in] value will determine the size of the provided char array. stringSize [out] will hold the actual size of the string to extract.
3. If the 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.
4. stringSize should reflect the actual size of the provided char array otherwise it might cause a severe crash.

C#

```
System.String IDevice.GetValueString(System.String paramName);
```

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

String value of chosen parameter configuration field of String type. Type: System::String

Python

```
def KYFG_GetGrabberValueStringCopy(handle, paramName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen frame grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK / FGSTATUS_BUFFER_TOO_SMALL – value will indicate that in function calculated buffer size is too small to hold the requested string value / INPUT_ARGUMENT_TYPE_ERROR / Other error.
paramStr - String value of chosen parameter configuration field of String type.

13.3 KYFG_GetGrabberValueIntMaxMin()

Get Frame Grabber maximum and minimum configuration field values of Integer type. According to Gen<i>Cam standard naming and xml field definition and type.

C

```
FGSTATUS KYFG_GetGrabberValueIntMaxMin(
        FGHANDLE handle,
        const char *paramName,
        int64_t* pIntMax,
        int64_t* pIntMin);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
pIntMax	int64_t*	A pointer that will be filled with the value of the chosen parameter
pIntMin	int64_t*	A pointer that will be filled with the value of the chosen parameter

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.

C#

```
System.Object IDevice.GetValueIntMaxMin(System.String paramName);
```

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

Object, that represents a param value corresponding to the param name. Type: Tuple (see remarks). [1]

Python

```
def KYFG_GetGrabberValueIntMaxMin(handle, paramName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to chosen Frame Grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
value_int_max - Maximum configuration value of Integer type field. Type: int
value_int_min - Minimum configuration value of Integer type field. Type: int

Remarks:

1. Object will be the instance of:

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

13.4 KYFG_GetGrabberValueFloatMaxMin()

Get Frame Grabber maximum and minimum configuration field values of Float type. According to Gen<i>Cam standard naming and xml field definition and type.

C

```
FGSTATUS KYFG_GetGrabberValueFloatMaxMin(
	FGHANDLE handle,
	const char *paramName,
	double* pDoubleMax,
	double* pDoubleMin);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber. See general remarks.
paramName	const char*	Name of configuration parameter
pDoubleMax	double*	A pointer that will be filled with the value of the chosen parameter
pDoubleMin	double*	A pointer that will be filled with the value of the chosen parameter

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.

C#

```
System.Object IDevice.GetValueFloatMaxMin(System.String paramName);
```

Parameter Name	Type	Description
paramName	System::String	Name of configuration parameter

Return value:

Object, that represents a param value corresponding to the param name. Type: Tuple (see remarks) [1].

Python

```
def KYFG_GetGrabberValueFloatMaxMin(handle, paramName)
```

Parameter Name	Type	Description
handle	FGHANDLE or int	API handle to the chosen Frame Grabber.
paramName	str	Name of the configuration parameter.

Return value:

FGSTATUS - FGSTATUS_OK. INPUT_ARGUMENT_TYPE_ERROR / Other errors will be returned in case of an error.
value_float_max - Maximum configuration value of Float type field. Type: Double
value_float_min - Minimum configuration value of Float type field. Type: Double

Remarks:

1. Object will be the instance of:

```
System.Tuple< System.Double, System.Double>
```

14 Authentication interface

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

14.1 KY_AuthProgramKey()

C

```
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	Please see the remarks! [1]

Return value:

FGSTATUS - Status and error report.

C#

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

Parameter Name	Type	Description
key	array<System.Byte>	A key to be programmed into Frame Grabber
lock	int	Please see the remarks! [1]

Python

```
def KY_AuthProgramKey(handle, pKey, lock)
```

Parameter Name	Type	Description
handle	FGHANDLE	API handle to a Frame Grabber
key	array<System.Byte>	A key to be programmed into Frame Grabber
lock	int	Please see the remarks! [1]

Return value:

FGSTATUS - Status and error report.

Remarks:

1. 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 the following call of this function will fail.

14.2 KY_AuthVerify()

Verify provided key against one already programmed to the grabber.

C

`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 containing information to be programmed into 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);
}
```

C#

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

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.

Python

```
def KY_AuthVerify(handle, pKey)
```

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

Return value:

FGSTATUS - Status and error report.

15 Buffer Interface (DEPRECATED) (C only)

The API described in this section is deprecated. The API described in “Stream Interface” should be used in new applications.

15.1 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 the 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 crashes.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
buffHandle	BUFFHANDLE*	Pointer to API handle to a data buffer for the selected camera. [1]
frames	uint32_t	The number of frames that should be allocated for this buffer. [2]

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 a large frame rate.

Example code:

```
CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS]
// maximum KY_MAX_CAMERAS cameras can be connected
BUFFHANDLE buffHandle = 0;
…
if (FGSTATUS_OK == (KYFG_BufferAlloc(camHandleArray[grabberIndex], &buffHandle, 16))
{
    printf(“New buffer was allocated with handle %X”, buffHandle);
}
```

15.2 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 the camera.

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

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

Return value:

FGSTATUS - Status and error report.

15.3 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 a data buffer for selected camera

Return value:

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

15.4 KYFG_BufferGetFrameIndex() (DEPRECATED)

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

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

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

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

Return value:

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

15.5 KYFG_BufferGetPtr() (DEPRECATED)

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

Retrieves a pointer to a 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 a data buffer for selected camera
frame	uint32_t	Frame index of data pointer to be retrieved

Return value:

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

Example code:

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

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

15.6 KYFG_BufferGetAux() (DEPRECATED)

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

Retrieves a pointer to Auxiliary data of the specified frame.

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

Parameter name	Type	Description
buffHandle	BUFFHANDLE	API handle to a data buffer for selected camera
frame	int	Frame index of data pointer to be retrieved
pAuxData	KYFG_AUX_DATA*	Pointer to auxiliary data of the 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);
	}
```

16 Stream Interface

16.1 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, the 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 suited 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 the application, while simultaneously being filled with new data by Hardware. To avoid application data corruption, the stream callback function, implemented by the user application, should process frame memory as fast as possible and return control to the 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 the memory buffer of frames that 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 the user application is issued. This frame memory will not be affected until it is returned to *Input Queue* using KYFG_BufferToQueue() function call. The user application is responsible for putting frames to *Input Queue* for each frame supplied to the host application through Stream callback. If the host application fails to do so then *Input Queue* will eventually become empty and newly 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, the DEVICE_QUEUED_BUFFERS_SUPPORTED grabber parameter should be read using KYFG_GetGrabberValueInt() function call. If the returned value is 1 then queued buffers mode is supported, otherwise all API functions related to this mode will return the following error: FGSTATUS_QUEUED_BUFFERS_NOT_SUPPORTED

16.2 KYFG_StreamCreateAndAlloc()

A new stream will be allocated for the specified camera. The created stream buffers will hold the data of acquired frames/generated. Stream buffer mechanism and buffer size calculations are handled internally. Buffer frame size is calculated with consideration of the specified number of frames, 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 operation and memory leaks, therefore the stream should be recreated with the new frame dimensions. See remarks.

C

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

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

Return value:

FGSTATUS - Status and error report.

Example code:

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

C#

```
IStream ICamera.StreamCreateAndAlloc(int frames);
```

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

Return value:

IStream handle of newly created stream.

Python

def KYFG_StreamCreateAndAlloc(camHandle, frames, streamIndex)

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the connected camera
frames	int	The number of frames that should be allocated for this stream.
streamIndex	int	Index of the stream. Currently unused and must be 0.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
pStreamHandle - pointer to STREAM_HANDLE variable that will hold the handle of the newly created stream. Type: STREAM_HANDLE

Remarks:

1. Parameters which affect the frame dimensions:
    a. Grabber parameters: “Width”, “Height”, “PixelFormat”, “PackedDataMode”, “SegmentsPerBuffer”, “DecimationVertical”, “DebayerMode”.
    b. Camera parameters: “Width”, “Height”, “PixelFormat”.
2. In addition, other camera parameters might change the camera image dimension, which should be reflected in width/height/pixel format, after those parameters are applied, e.g. binning/decimation.

16.3 **KYFG_StreamCreate()**

A new stream will be created for the chosen camera. The stream will manage frame buffers allocated either by the user or by the library. Frame buffers will be organized in queues – input, output, automatic – and in a set of unqueued frame buffers. Changing certain camera/grabber parameters, after successfully stream allocation, might result in unstable operation and memory leaks, therefore the stream should be recreated with the new frame dimensions. See remarks.

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

```
IStream ICamera.StreamCreate();
```

Return value:

IStream handle of newly created stream.

Python

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

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

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

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

Remarks:

1. Parameters which affect the frame dimensions:
   a. Grabber parameters: “Width”, “Height”, “PixelFormat”, “PackedDataMode”, “SegmentsPerBuffer”, “DecimationVertical”, “DebayerMode”.
   b. Camera parameters: “Width”, “Height”, “PixelFormat”.
2. In addition, other camera parameters might change the camera image dimension, which should be reflected in width/height/pixel format, after those parameters are applied, e.g. binning/decimation.

16.4 KYFG_StreamLinkFramesContinuously()

Links all announced frame buffers into a stream to form a continuous buffer. This function can be called only after all frames have been announced, using either KYFG_BufferAllocAndAnnounce() or KYFG_BufferAnnounce(). When creating a stream using this method, user will not be responsible for the buffer management control and buffers will rotate automatically without the need to queue them. i.e KYFG_BufferToQueue() or KYFG_BufferQueueAll() should NOT be called before starting the stream.

C

FGSTATUS KYFG_StreamLinkFramesContinuously(STREAM_HANDLE streamHandle);

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a previously created stream

Return value:

FGSTATUS - Status and error report.

C#

void ICamera.StreamLinkFramesContinuously(IStream streamHandle);

Parameter Name	Type	Description
streamHandle	IStream	API handle to data stream for selected camera

Python

def KYFG_StreamLinkFramesContinuously(streamHandle)

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle to a previously created stream

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

16.5 KYFG_StreamGetInfo()

Retrieves information about the specified stream.

C

```
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	One of the values stored in KY_STREAM_INFO_CMD, specifies what information is being requested.
pInfoBuffer	void *	Pointer to user variable that will be filled with the required information. Can be NULL. [1]
pInfoSize	size_t *	Pointer to the size of provided pInfoBuffer. Can be NULL. In: the size of the provided pInfoBuffer in bytes. Out: minimum required size of pInfoBuffer to hold the requested information. [2]
pInfoType	KY_DATA_TYPE *	Pointer to the data type of pInfoBuffer content. Can be NULL. Out: data type of pInfoBuffer for requested information. [3]

Return value:

FGSTATUS - Status and error report.

Remarks:

1. If the information type is known, provide a valid buffer of the correct size to fill in the 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.

C#

```
System.Object IStream.GetInfo(KY_STREAM_INFO_CMD info);
```

Parameter Name	Type	Description
info	KY_STREAM_INFO_CMD	One of the values stored in KY_STREAM_INFO_CMD, specifies what information is being requested.

Return value:

The required info

Python

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

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle of a stream
cmdStreamInfo	KY_STREAM_INFO_CMD	One of the values stored in KY_STREAM_INFO_CMD, specifies what information is being requested.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

pInfoBuffer - Byte array memory block with the required information.

pInfoSize - Size of pInfoBuffer.

pInfoType - Data type of pInfoBuffer content.

16.6 KYFG_StreamGetSize()

Retrieves the size of the last acquired/generated frame from a specified stream.

C

`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 the specified stream. In case of an error, -1 will be returned.

C#

`System.UInt64 IStreamBuffer.getSize();`

Return value:
The size of the current buffer.

Python

`def KYFG_StreamGetSize(streamHandle)`

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value:
FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
StreamGetSize - Size of the last acquired frame acquired from the specified stream. Type: int

16.7 KYFG_StreamGetFrameIndex()

Retrieves the index of the last acquired/generated frame from a specified stream.

C

```
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/generated frame from the specified stream. In case of an error, -1 will be returned.

C#

```
int IStreamBuffer.GetFrameIndex();
```

Return value:

Index of the current buffer (frame).

Python

```
def KYFG_StreamGetFrameIndex(streamHandle)
```

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

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

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

16.8 KYFG_StreamGetPtr()

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

C

```
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 the specified frame. NULL will be retrieved if the frame index is out of range or another 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);
        }
```

C#

```
System.Object IStream.GetPtr(int buffIndex);
```

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

Return value:

Pointer to required buffer.

Python

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

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream.
buffIndex	int	Frame index of data pointer to be retrieved.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

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

16.9 KYFG_StreamGetAux()

Retrieves Auxiliary data of the specified frame.

C

```
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 the specified frame. The payload should be interpreted as KYFG_FRAME_AUX_DATA

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.u_data.frame_data.sequence_number, auxData.u_data.frame_data.timestamp);
}
```

C#

```
AuxData IStream.GetAux(int frame);
```

Parameter Name	Type	Description
frame	int	Frame index to be retrieved

Return value:

Auxilary data of specified frame. Type: KYFG_AUX_DATA

Python

```
def KYFG_StreamGetAux(streamHandle, frame, 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	Struct of auxiliary data to be filled. The payload should be interpreted as KYFG_FRAME_AUX_DATA

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

16.10 KYFG_StreamDelete()

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

C

```
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.

C#

```
void IStream.Delete();
```

Python

```
def KYFG_StreamDelete(streamHandle)
```

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

Remarks:

1. Starting from API 5.2 this function will return FGSTATUS_STREAM_IS_LOCKED if the acquisition is in process at the time of calling this function. The application must stop acquisition before deleting a stream.

16.11 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 the function KYFG_StreamGetInfo() with the 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, the user should add it to the incoming queue using the function KYFG_BufferToQueue(). Alternatively, the function KYFG_BufferQueueAll() can be used after all desired user buffers are announced.

C

```
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 the connected camera
nBufferSize	size_t	The size of allocated memory. Reserved for future enhancements and currently, this parameter MUST be equal to the 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 the handle of newly announced frame buffer

Return value:
FGSTATUS - Status and error report.

C#

```
IStreamBuffer IStream.BufferAllocAndAnnounce(System.UInt64 nBufferSize);
```

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

Return value:
Allocated Stream Buffer

Python

```
def KYFG_BufferAllocAndAnnounce(streamHandle, nBufferSize, pPrivate)
```

Parameter Name	Type	Description

streamHandle	STREAM_HANDLE or int	API handle of a stream
nBufferSize	int	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	int	This parameter is currently ignored

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
pBufferHandle - STREAM_BUFFER_HANDLE variable that will hold handle of newly announced frame buffer.
Type: STREAM_BUFFER_HANDLE.

16.12 KYFG_BufferAnnounce()

This function is used to announce a buffer allocated by the user and bind it to a stream. The memory size should correspond to a single acquisition frame. This size can be retrieved using the function KYFG_StreamGetInfo() with the KY_STREAM_INFO_PAYLOAD_SIZE info command. Also, any virtual memory allocated by the 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 the library and MUST stay valid until the 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, the user should add it to an incoming queue using the function KYFG_BufferToQueue(). Alternatively, the function KYFG_BufferQueueAll() can be used after all desired user buffers are announced.

C

```
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 of a stream
pBuffer	void*	Input parameter - pointer to memory allocated by the user
nBufferSize	size_t	The size of allocated memory. Reserved for future enhancements and currently, this parameter MUST be equal to the 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 the handle of newly announced frame buffer

Return value:

FGSTATUS - Status and error report.

C#

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

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

Return value:

Instance of Stream Buffer class that implements IStreamBuffer interface.

Python

```
def KYFG_BufferAnnounce(streamHandle, pBuffer, pPrivate)
```

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE or int	API handle of a stream
pBuffer	bytearray	Pointer to an empty list (memory allocated by user) of nBufferSize size
pPrivate	int	This parameter is currently ignored

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.
pBufferHandle - STREAM_BUFFER_HANDLE variable that will hold handle of newly announced frame buffer.
Type: STREAM_BUFFER_HANDLE.

16.13 KYFG_BufferAnnounceChunks()

This function is used to announce a buffer per chunk to selected stream.

C

```
FGSTATUS KYFG_BufferAnnounceChunks(
        STREAM_HANDLE streamHandle,
        const BUFFER_DATA_CHUNK* pDataChunkArr,
        size_t nChunks,
        void * pPrivate,
        STREAM_BUFFER_HANDLE * pBufferHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream
pDataChunkArr	BUFFER_DATA_CHUNK*	Array of data chunks
nChunks	size_t	Number of chunks in pDataChunkArr
pPrivate	void*	This parameter is currently ignored
pBufferHandle	STREAM_BUFFER_HANDLE *	Output parameter - pointer to STREAM_BUFFER_HANDLE variable that will hold the handle of previously announced frame buffer

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

Not applicable.

16.14 KYFG_BufferGetInfo()

Retrieves information about previously announced buffer.

C

```
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	One of the values stored in KY_STREAM_BUFFER_INFO_CMD, specifies what information is being requested.
pInfoBuffer	void *	Pointer to user variable that will be filled with the required information. Can be NULL. [1]
pInfoSize	size_t *	Pointer to the size of provided pInfoBuffer. Can be NULL. In: the size of the provided pInfoBuffer in bytes. Out: minimum required size of pInfoBuffer to hold the requested information. [2]
pInfoType	KY_DATA_TYPE *	Pointer to the data type of pInfoBuffer content. Can be NULL. Out: data type of pInfoBuffer for requested information. [3]

Return value:

FGSTATUS - Status and error report.

Remarks:

1. If the information type is known, provide a valid buffer of the correct size to fill in the 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.

C#

```
System.Object IStreamBuffer.GetInfo(KY_STREAM_BUFFER_INFO_CMD info);
```

Parameter Name	Type	Description
info	KY_STREAM_BUFFER_INFO_CMD	One of the values stored in KY_STREAM_BUFFER_INFO_CMD, specifies what information is being requested.

Return value:

The required info regarding the buffer (frame).

Python

`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.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

pInfoBuffer - Byte array memory block with required information.

pInfoSize - Minimum required size of provided pInfoBuffer in bytes.

pInfoType - Data type of pInfoBuffer content for requested information. Type: One of KY_DATA_TYPE values

16.15 KYFG_BufferToQueue()

Moves a previously announced buffer to a specified queue.

C

```
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	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Return value:

FGSTATUS - Status and error report.

C#

```
void IStreamBuffer.BufferToQueue(KY_ACQ_QUEUE_TYPE dstQueue);
```

Parameter Name	Type	Description
dstQueue	KY_ACQ_QUEUE_TYPE	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Python

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

Parameter Name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE or int	Handle of a stream buffer
dstQueue	int	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error.

16.16 KYFG_BufferQueueAll()

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

C

```
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	One of the values stored in KY_ACQ_QUEUE_TYPE source queue.
dstQueue	KY_ACQ_QUEUE_TYPE	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Return value:

FGSTATUS - Status and error report.

C#

```
void IStream.BufferQueueAll(KY_ACQ_QUEUE_TYPE srcQueue, KY_ACQ_QUEUE_TYPE dstQueue);
```

Parameter Name	Type	Description
srcQueue	KY_ACQ_QUEUE_TYPE	One of the values stored in KY_ACQ_QUEUE_TYPE source queue.
dstQueue	KY_ACQ_QUEUE_TYPE	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Python

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

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE or int	Handle of a stream
srcQueue	int	One of the values stored in KY_ACQ_QUEUE_TYPE source queue.
dstQueue	int	One of the values stored in KY_ACQ_QUEUE_TYPE destination queue.

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

Remarks:

1. This function doesn't guarantee any buffer order queue. To get a specific buffer queue order the user should use individual BufferToQueue().

16.17 KYFG_BufferRevoke()

Revoke and delete resources of previously announced buffer. The buffer should be announced to 'streamHandle' stream and acquisition should be stopped.

C

```
FGSTATUS KYFG_BufferRevoke(
        STREAM_HANDLE streamHandle,
        STREAM_BUFFER_HANDLE streamBufferHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	Handle of a stream
streamBufferHandle	STREAM_BUFFER_HANDLE	Handle of previously announced buffer

Return value:

FGSTATUS - Status and error report.

C#

```
void IStream.BufferRevoke(STREAM_BUFFER_HANDLE streamBufferHandle);
```

Parameter Name	Type	Description
streamBufferHandle	STREAM_BUFFER_HANDLE	Handle of previously announced buffer

Python

```
def  KYFG_BufferRevoke(streamHandle, streamBufferHandle)
```

Parameter Name	Type	Description
streamHandle	STREAM_HANDLE	Handle of a stream
streamBufferHandle	STREAM_BUFFER_HANDLE	Handle of previously announced buffer

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int

16.18 KYFG_BufferSubmit()

Allocation of a stream for a list of buffers consisting of contiguous physical memory, these should be accessed via provided direct physical address.

C

```
FGSTATUS KYFG_BufferSubmit(
        CAMHANDLE camHandle,
        STREAM_HANDLE *streamHandle,
        void** bufferPtrArray,
        unsigned int frames,
        unsigned int frameSize,
        unsigned int flags);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
streamHandle	STREAM_HANDLE*	Pointer to the stream handle that will be created
bufferPtrArray	void**	Array of buffers' base physical memory addresses, when flags indicate SUBMIT_BUFF_PHYSICAL_ADDRESS. e.g uint64_t bufferPtrArray[nFrames];
frames	unsigned int	Number of announced frame memory pointers (i.e buffer base physical memory address) provided by bufferPtrArray. If 0 is passed, number of frames per stream will be retrieved by the function
frameSize	unsigned int	Size of a single frame
flags	unsigned int	Flags parameter value should be SUBMIT_BUFF_PHYSICAL_ADDRESS

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def  KYFG_BufferSubmit(camHandle, bufferPhysicalAddrArr, frameSize, flags)
```

Parameter Name	Type	Description
camHandle	CAMHANDLE or int	API handle to the chosen camera.
bufferPhysicalAddrArr	List	List of buffers' base physical memory addresses, .e.g uint64_t bufferPtrArray[nFrames];
frameSize	int	Size of a single frame (height * width * bitness in Bytes)
flags	int	Flags parameter value should always be SUBMIT_BUFF_PHYSICAL_ADDRESS

Return value:

FGSTATUS - Status and error report. FGSTATUS_OK / INPUT_ARGUMENT_TYPE_ERROR / Other error. Type: int
cameraStreamHandle - API handle to data stream for selected camera Type: STREAM_HANDLE

16.19 Example of code using Cyclic buffers

```
void Stream_callback_func(void* userContext, STREAM_HANDLE streamHandle)
{
        if (!streamBufferHandle)
        {
                // this callback indicates that acquisition has stopped
                return;
        }
        // process data associated with given stream buffer
        unsigned char* pFrameMemory = 0;
        KYFG_BufferGetInfo(streamBufferHandle,
                                        KY_STREAM_BUFFER_INFO_BASE,
                                        &pFrameMemory,
                                        NULL,
                                        NULL);
        printf("\rGood callback stream's buffer handle:%" PRISTREAM_BUFFER_HANDLE ", ID:%02" PRIu32,
                streamBufferHandle, frameId);
}

int main(int argc, char* argv[])
{
        FGHANDLE handle;
        CAMHANDLE camHandleArray[MAXBOARDS][KY_MAX_CAMERAS];
        // maximum KY_MAX_CAMERAS cameras can be connected
        int nDetectedCameras = 0, grabberIndex = 0, cameraIndex = 0;
        STREAM_HANDLE streamHandle = INVALID_STREAMHANDLE;
         …
        KYFG_CameraScan(handle, camHandleArray, &nDetectedCameras);
        if (nDetectedCameras > 0 )
        {
                KYFG_CameraCallbackRegister(camHandleArray[grabberIndex], Stream_callback_func, NULL);
        }
        if(FGSTATUS_OK != KYFG_StreamCreateAndAlloc(camHandleArray[grabberIndex][cameraIndex],
        &streamHandle , 16, 0))
        {
                 printf("Failed to allocate buffer.\n");
        }
        if(streamHandle != 0)
        {
                KYFG_CameraStart(camHandleArray[grabberIndex][cameraIndex], streamHandle, 0);
        }
        while(1){}
        return 0;
}
```

16.20 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[MAXBOARDS][KY_MAX_CAMERAS]
        // maximum KY_MAX_CAMERAS cameras can be connected
        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[grabberIndex], &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
        KYFG_CameraStart(camHandleArray[grabberIndex], cameraStreamHandle, 0)
         ...
        while(1){}
        return 0;
    }
```

Remarks:

1. 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 the queue, for example: (ID: 15, total frames: 29).

17 Data acquisition

17.1 KYFG_CameraStart()

Acquisition mode: Starts stream acquisition for the specified stream and camera. A single stream can be active at a time for each camera. The buffers of the selected stream will be filled with data acquired from the camera. The total number of acquired frames can be limited by the <frames> parameter; 0 value will indicate a continuous frame acquisition.

Generation mode: Starts stream generation for the specified stream and camera. The number of frames to be generated will be defined by the <frames> parameter; 0 value will result in continuous frame generation.

Remarks:

1. Starting from API 5.2 this function will return FGSTATUS_STREAM_CANNOT_LOCK if an acquisition is in process at the time of calling this function. The application must stop the previous acquisition before starting a new one.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera
streamHandle	STREAM_HANDLE	API handle to the data stream for selected camera
frames	int	The number of frames to be acquired/ generated, 0 for continuous acquisition mode.

Return value:

FGSTATUS - Status and error report. [1]

C#

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

Parameter Name	Type	Description
streamHandle	IStream	API handle to data stream for selected camera
frames	int	The number of frames to be acquired/ generated, 0 for continuous acquisition mode.

Python

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	The number of frames to be acquired/ generated, 0 for continuous acquisition mode.

Return value:

FGSTATUS - Status and error report.

17.2 KYFG_CameraStop()

Acquisition mode: Stops stream acquisition of the active stream for the selected camera.
Generation mode: Stops stream generation of the active stream for the selected camera.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the connected camera

Return value:

FGSTATUS - Status and error report.

C#

```
void ICamera.Stop();
```

Python

```
def KYFG_CameraStop(camHandle)
```

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

Return value:

FGSTATUS - Status and error report.

18 Data loading

This chapter describes API functions for handling data files for the Chameleon camera simulator.

18.1 KYFG_LoadPatternData()

Previously allocated frames will be filed with specified data from files and committed it to stream as a video source for simulation mode. Several patterns types are available for generation. The patterns image format is defined by the camera configuration parameters regardless of whether it's a colored or non-colored pattern.

C

```
FGSTATUS KYFG_LoadPatternData(
        STREAM_HANDLE streamHandle,
        PATTERN_TYPE type,
        uint16_t  *FixedPatternColor);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to the chosen simulator
type	PATTERN_TYPE	Pattern type to be simulated.
FixedPatternColor	uint16_t *	A pointer to an array of 16bit (Little Endian) aligned color channels. [1]

Return value:
FGSTATUS – Simulator status and error report.

Example code:
“<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KY_Chameleon_Example.c”

C#

Not applicable.

Python

```
def KYFG_LoadPatternData(streamHandle, type, FixedPatternColor)
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to chosen simulator
type	PATTERN_TYPE	Pattern type to be simulated.
FixedPatternColor	bytearray	A pointer to an array of 16bit (Little Endian) aligned color channels. Can be NULL. [1]

Return value:
FGSTATUS – Simulator status and error report.

Remarks:

1. In case a fixed pattern (PATTERN_FIXED) is to be generated, a color should be specified; whether an 8, 10, 12, 14 or 16bit color plane is chosen, the array values should be 16bit values, cropped to the right bit width.

18.2 KYFG_LoadFileData()

Previously allocated frames will be filed with specified data from files and committed it to stream as a video source for simulation mode. Image types ".bmp", ".tif", ".pgn", and ".raw" are supported.

C

```
FGSTATUS KYFG_LoadFileData(STREAM_HANDLE streamHandle, const char* path, const char* type, int frames);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to the chosen simulator
path	const char*	The path of chosen image file.
type	const char*	Type of image file: "bmp", "tif", "png" or "raw"
frames	int	Number of frames to generate

Return value:

FGSTATUS – Simulator status and error report.

Example code:

Example code can be found under:
"<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KY_Chameleon_Example.c".

C#

Not applicable.

Python

```
def KYFG_LoadFileData(streamHandle, path, type, frames)
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to chosen simulator
path	str	The path of chosen image file
type	str	Type of image file: "bmp", "tif", "png" or "raw"
frames	int	Number of frames to generate

Return value:

FGSTATUS – Simulator status and error report.

Remarks:

1. The "path" may contain a directory of files. In that case, several files may be loaded to fill the specified number of frames.
2. If the number of existing files doesn't match the number of specified "frames" the last files will not be used or the last file will be duplicated to match the number of specified "frames", depending on relation.
3. Data will be filled according to each input files type:
   "BMP", "tif", "png"- the image will be scaled to specified size of the camera parameters and filled into buffers.
   "RAW"- the file's raw data will be distributed across all the available frames. The number of frames is calculated according to image format configurations and RAW file size.

19 Image header access

This chapter describes API functions for accessing the image header in the Chameleon camera simulator.

19.1 KYCS_GetImageHeader()

Retrieves current image generation header.

C

```
CSSTATUS KYCS_GetImageHeader(CSHANDLE handle, unsigned char* header, unsigned int size);
```

Parameter name	Type	Description
handle	CSHANDLE	Handle to the camera the simulator device
header	unsigned char*	The user-supplied buffer where header should be stored
size	unsigned int	Size of user-supplied buffer. Maximum IMAGE_HEADER_SIZE can be specified, otherwise, CSSTATUS_IMAGE_HEADER_INJECTION_SIZE_TOO_BIG will be returned by the function

Return value:
CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

Not applicable.

19.2 KYCS_InjectImageHeader()

Allow overwriting one field of the standard stream image during a configurable number of image generation cycles.

C

`CSSTATUS KYCS_InjectImageHeader(CSHANDLE handle, unsigned char* header, unsigned int size, int cycles);`

Parameter name	Type	Description
handle	CSHANDLE	Handle to the camera the simulator device
header	unsigned char*	User-supplied buffer with header
size	unsigned int	Size of user-supplied buffer. Maximum IMAGE_HEADER_SIZE can be specified, otherwise, CSSTATUS_IMAGE_HEADER_INJECTION_SIZE_TOO_BIG will be returned by the function.
cycles	int	The number of image generation cycles. Any value less than 1 is treated as -1 and stops further injections

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

Not applicable.

20 CRC injection

This chapter describes API functions for CRC injection in the Chameleon camera simulator.

20.1 KYCS_InjectVideoCRCErrors()

Allows setting a wrong CRC in one stream packet of one image generation, during a configurable number of image generation cycles.

C

```
CSSTATUS KYCS_InjectVideoCRCErrors(CSHANDLE handle, int cycles);
```

Parameter name	Type	Description
handle	CSHANDLE	API handle to the camera simulator device
cycles	Int	The number of image generation cycles. Any value less than 1 is treated as -1 and stops further injections

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

```
def KYCS_InjectVideoCRCErrors(handle, cycles)
```

Parameter name	Type	Description
handle	CSHANDLE	API handle to the camera simulator device
cycles	int	Number of image generation cycles. Any value less than 1 is treated as -1 and stops further injections

Return value:

CSSTATUS - Error code of function execution status.

20.2 KYCS_InjectControlCRCErrors()

Allows setting a wrong CRC in the next return Control/Command packets, during a configurable number of cycles.

C

```
CSSTATUS KYCS_InjectControlCRCErrors(CSHANDLE handle, int cycles);
```

Parameter name	Type	Description
handle	CSHANDLE	API handle to the camera simulator device
cycles	Int	The number of image generation cycles. Any value less than 1 is treated as -1 and stops further injections

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

```
def KYCS_InjectControlCRCErrors(handle, cycles)
```

Parameter name	Type	Description
handle	CSHANDLE	API handle to the camera simulator device
cycles	int	Number of image generation cycles. Any value less than 1 is treated as -1 and stops further injections

Return value:

CSSTATUS - Error code of function execution status.

21 Low-level bootstrap access

21.1 KYFG_ReadPortReg()

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

C

```
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.

C#

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

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. Type: System::UInt32

Python

```
def KYFG_ReadPortReg(handle, port, address)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
port	int	Device port index
address	int	Address of the register

Return value:
FGSTATUS - Status and error report.
read_data - Integer of 4 bytes, that represents the read register

21.2 KYFG_ReadPortBlock()

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

C

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

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

Return value:

FGSTATUS - Status and error report.

C#

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

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. Type: Byte array

Python

```
def KYFG_ReadPortBlock(handle, port, address, pBuffer, pSize)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
port	int	Device port index
address	int	Address of the register to read
pBuffer	bytearray	Empty list that will hold the data
pSize	int	Size in bytes of buffer to read

Return value:

FGSTATUS - Status and error report.

pSize - size of read bytes

21.3 KYFG_WritePortReg()

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

C

`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.

C#

`void IDevice.WritePortReg(int port, System::UInt64 address, System::UInt32 data);`

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

Python

`def KYFG_WritePortReg(handle, port, address, data)`

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
port	int	Device port index
address	int	Address of the register
pBuffer	int	Integer of 4 bytes, that represents the Bootstrap registers value

Return value:

FGSTATUS - Status and error report.

21.4 KYFG_WritePortBlock()

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

C

```
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 the size of the data buffer. In: size in bytes of buffer to write Out: the size of written bytes

Return value:

FGSTATUS - Status and error report.

C#

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

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

Python

```
def KYFG_WritePortBlock(handle, port, address, pBuffer)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen Frame Grabber
port	int	Frame Grabber port index
address	int	Start address of the data to write
pBuffer	int	Integer of 4 bytes, that represents the registers value

Return value:

FGSTATUS - Status and error report.

pSize - Size of written bytes

21.5 KYFG_CameraReadReg()

Direct read data buffer from the selected camera.

C

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

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

Return value:
FGSTATUS - Status and error report.

C#

`array<System::Byte> ICamera.ReadReg(System::UInt64 address, System::UInt32 size);`

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 data from register. Type: Byte array

Python

`def KYFG_CameraReadReg(camHandle, address, pBuffer, pSize)`

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
address	int	Start address of the data to read
pBuffer	bytearray	Empty list which will be filled with data
pSize	int	Size in bytes of buffer to read

Return value:
FGSTATUS - Status and error report.
pSize - Size of read bytes

21.6 KYFG_CameraWriteReg()

Direct write data buffer to the selected camera.

C

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to the 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 the size of the data buffer. In: size in bytes of buffer to read Out: the size of read bytes

Return value:

FGSTATUS - Status and error report.

C#

```
System::UInt32 ICamera.WriteReg(System::UInt64 address, array<System::Byte> buffer);
```

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. Type: System::UInt32

Python

```
def KYFG_CameraWriteReg(camHandle, address, pBuffer)
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
address	int	Start address of the data to write
pBuffer	int	Integer of 4 bytes, that represents the registers value

Return value:

FGSTATUS - Status and error report.

pSize - Size of written bytes

21.7 KYFG_GrabberReadReg()

Access to logical register space of a device (the register space is described in the device's xml file).

C

```
FGSTATUS KYFG_GrabberReadReg(FGHANDLE handle, uint64_t address, void* pBuffer, uint32_t * pSize);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
pBuffer	void *	Pointer to the buffer that will hold read data
pSize	uint32_t *	Pointer to the size of the data buffer

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def KYFG_GrabberReadReg(handle, address, pBuffer, pSize)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected device
address	int	Start address of the data to read
pBuffer	bytearray	Empty list which will be filled with data
pSize	int	Size in bytes of buffer to read

Return value:

FGSTATUS - Status and error report.

pSize - Size of read bytes

21.8 KYFG_GrabberWriteReg()

Write to the logical register space of a device (the register space is described in the device's xml file).

C

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
pBuffer	const void *	Pointer to buffer data to write
pSize	uint32_t *	Pointer to the size of the data buffer

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def KYFG_GrabberWriteReg(handle, address, pBuffer)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected device
address	int	Start address of the data to read
pBuffer	int	Integer of 4 bytes, that represents the registers value

Return value:

FGSTATUS - Status and error report.

pSize - Size of written bytes

21.9 KYFG_DeviceDirectHardwareRead()

Direct access to the device's hardware registers.

C

```
FGSTATUS KYFG_ DeviceDirectHardwareRead(
	FGHANDLE handle,
	uint64_t address,
	void* pBuffer,
	uint32_t bytes);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
pBuffer	void *	Pointer to the buffer that will hold read data
bytes	uint32_t	Number of bytes

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def KYFG_DeviceDirectHardwareRead(handle, address, pBuffer, pSize)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected device
address	int	Start address of the data to read
pBuffer	bytearray	Empty list which will be filled with data
pSize	int	Size in bytes of buffer to read

Return value:

FGSTATUS - Status and error report.

pSize - Size of read bytes

21.10 KYFG_DeviceDirectHardwareWrite()

Write to the device's hardware registers.

C

```
FGSTATUS KYFG_ DeviceDirectHardwareWrite(
		FGHANDLE handle,
		uint64_t address,
		const void* pBuffer,
		uint32_t bytes);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
pBuffer	const void *	Pointer to buffer data to write
bytes	uint32_t	Number of bytes

Return value:

FGSTATUS - Status and error report.

C#

Not applicable.

Python

```
def KYFG_DeviceDirectHardwareWrite(handle, address, pBuffer)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected device
address	int	Start address of the data to read
pBuffer	int	Integer of 4 bytes, that represents the registers value

Return value:

FGSTATUS - Status and error report.

pSize - Size of written bytes

21.11 KYFG_GetPortStatus()

Status of specific device physical port regarding connectivity with a remote device (i.e camera).

C

`FGSTATUS KYFG_GetPortStatus(FGHANDLE handle, int port, PORT_STATUS* portStatus);`

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
port	int	Port number
portStatus	PORT_STATUS*	PORT_STATUS possible values: ▪ PORT_DISCONNECTED - the port is disconnected, no link has been established ▪ PORT_SYNCHRONIZED - port link is synchronized and awaiting connection ▪ PORT_CONNECTING - a connection is trying to be established on port ▪ PORT_CONNECTED - port is connected and assigned to the camera

Return value:

FGSTATUS - Status and error report.

C#

`PORT_STATUS IDevice.GetPortStatus(int port);`

Parameter name	Type	Description
port	int	Port number

Return value:

Status of the provided port. Type: PORT_STATUS

Python

`def KYFG_GetPortStatus(handle, port)`

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected device
port	int	Port number

Return value:

FGSTATUS - Status and error report.

portStatus- status of the provided port. Type: PORT_STATUS

21.12 KYCS_ReadBootstrapRegs()

Direct access to Chameleon camera Simulator's hardware registers.

C

```
CSSTATUS KYCS_ReadBootstrapRegs(FGHANDLE handle, uint32_t address, void* data, uint32_t size);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
data	void *	Pointer to the buffer that will hold read data
size	uint32_t	Number of processed bytes

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

```
def KYCS_ReadBootstrapRegs(handle, address, buffer, size)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
address	int	Start address of the data to read
buffer	bytearray	Empty list which will be filled with read data
size	int	Size in bytes of buffer to read

Return value:

FGSTATUS - Status and error report.

pSize - size of read bytes

21.13 KYCS_WriteBootstrapRegs()

Write to Chameleon camera Simulator's hardware registers.

C

```
CSSTATUS KYCS_WriteBootstrapRegs(FGHANDLE handle, uint32_t address, const void* data, uint32_t size);
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to the connected device
address	uint64_t	Start address of the data to read
data	const void *	Pointer to buffer data to write
size	uint32_t	Number of processed bytes

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

```
def KYCS_WriteBootstrapRegs(handle, address, data)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
address	int	Start address of the data to write
buffer	int	Integer of 4 bytes, that represents the registers value

Return value:

FGSTATUS - Status and error report.

pSize - Size of processed bytes

21.14 KYFG_CameraSendEventMessage()

Send Event Message via the Master channel of the camera.

C

```
FGSTATUS KYFG_CameraSendEventMessage(
        CAMHANDLE camHandle,
        uint32_t eventId,
        const void* pBuffer,
        uint32_t bufferSize);
```

Parameter name	Type	Description
camHandle	CAMHANDLE	Camera handle
eventId	uint32_t	ID of the event message
pBuffer	const void *	Event data payload, its length is specified by the size parameter
bufferSize	uint32_t	Event data size to process

Return value:

FGSTATUS - Status and error report.

C#

```
void ICamera.CameraSendEventMessage(System::UInt32 eventId, array<System::Byte> buffer);
```

Parameter Name	Type	Description
eventId	System::UInt32	ID of the event message
buffer	array<System::Byte>	Event data payload

Python

```
def KYFG_CameraSendEventMessage(camHandle, eventId, pBuffer)
```

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera
eventId	int	ID of the event message
pBuffer	int	Event data payload

Return value:

FGSTATUS - Status and error report.

pSize - Size of processed bytes

21.15 KYFG_DevicePortSendEventMessage()

Send Event Message via the specified port of the device [1] [2].

C

```
FGSTATUS KYFG_DevicePortSendEventMessage(
        FGHANDLE handle,
        int port,
        uint32_t eventId,
        const void* pBuffer,
        uint32_t bufferSize);
```

Parameter name	Type	Description
handle	FGHANDLE	Handle to the connected device
port	int	Device port through which data will be sent
eventId	uint32_t	ID of the event message
pBuffer	const void *	Event data payload, its length is specified by the size parameter
bufferSize	uint32_t	Event data size to process

Return value:

FGSTATUS - Status and error report.

C#

```
void IDevice.DevicePortSendEventMessage(int port, System::UInt32 eventId, array<System::Byte> buffer);
```

Parameter Name	Type	Description
port	int	Frame Grabber port index
eventId	System::UInt32	ID of the event message
buffer	array<System::Byte>	Event data payload

Python

```
def KYFG_DevicePortSendEventMessage(handle, port, eventId, pBuffer)
```

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen device
port	int	device port through which data will be sent
eventId	int	ID of the event message
pBuffer	int	Event data payload

Return value:

FGSTATUS - Status and error report.

pSize - Size of processed bytes.

Remarks:

1. CLHS protocol: send GenCP Event message; 10GigE protocol: send EVENTDATA message
2. Event messages from a remote device can be received using KYDeviceEventCallBackRegister(). The event ids for CLHS and 10GigE are KYDEVICE_EVENT_GENCP_EVENT_ID for CLHS and KYDEVICE_EVENT_GIGE_EVENTDATA_ID, respectively.

21.16 KYCS_GenerateCxpEvent()

Allows generating CXP2 HeartBeats and Events using Chameleon camera simulator (starting from firmware version 5.x.x).

C

`CSSTATUS KYCS_GenerateCxpEvent(CSHANDLE handle, KY_CXPEVENT_PACK* pEvent);`

Parameter name	Type	Description
handle	CSHANDLE	API handle to the camera simulator device
pEvent	KY_CXPEVENT_PACK *	Pointer to structure containing a CXP2 device event pack

Return value:

CSSTATUS - Error code of function execution status.

C#

Not applicable.

Python

`def KYFG_GenerateCxpEvent(handle)`

Parameter name	Type	Description
handle	FGHANDLE	API handle to connected Chameleon device

Return value:

CSSTATUS - Error code of function execution status. Type: int

pEvent - Structure containing a CXP2 device event pack. Type: KY_CXPEVENT_PACK

22 Firmware Update

22.1 KYFG_CheckUpdateFile

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

C

```
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 the minor revision number of the current firmware
pFlashMajorRev	uint16_t *	Pointer to uint16_t that will be filled with the major revision number of the current firmware
pFileMinorRev	uint16_t *	Pointer to uint16_t that will be filled with the minor revision number of the firmware contained in the update file
pFileMajorRev	uint16_t *	Pointer to uint16_t that will be filled with the major revision number of the firmware contained in the file
pFlashVendorId	uint16_t *	Pointer to uint16_t that will be filled with the vendor ID of the current firmware
pFlashBoardId	uint16_t *	Pointer to uint16_t that will be filled with board ID of the current firmware
pFileVendorId	uint16_t *	Pointer to uint16_t that will be filled with the 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.

C#

Not applicable.

Python

Not applicable.

22.2 KYFG_LoadFirmware

Updates device firmware from the supplied binary file. Progress is reported via a supplied callback function.

C

```
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
file	const char*	The file name for firmware update
callback	UPDATE_CALLBACK	Pointer to callback function to be called during an update
context	void*	User context will be passed as a parameter to each call of 'callback'

Return value:

FGSTATUS - Status and error report.

Remarks:

1. The software detects outdated firmware and disables all sets of operations, except firmware updates, which should be performed to proceed.

C#

Not applicable.

Python

Not applicable.

23 IO Configurations

23.1 IO Selectors

KAYA's Frame Grabbers provide a vast variety of configurable I/Os. This depends on the device's 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. 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).

23.2 Configuration fields

Complete available triggers list and possible configurations can be found in the "KAYA Frame Grabber Feature Guide" document.

Remarks:

1. For I/O source options of Firmware Version 1.xx, follow tables no. 4 and 5 located in the Appendices section.

Example code:

This example shows how to set the Timer 2 trigger source to be in continuous mode:

```
FGHANDLE handle;  // maximum 4 cameras can be connected
int64_t 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");
```

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 to pass the signal from Timer 0 to the camera over coax. This setup both generates a trail of trigger packets to the 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");
```

24 API Defines and Macros

24.1 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.

24.2 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
```

24.3 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 Stream interface function. A callback with buffHandle zero indicates the stop of acquisition for the camera associated with the current buffer.

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

24.4 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 the 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 the registered callback function.

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

24.5 CameraCallback

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

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

24.6 FGAuxDataCallback

The callback function will be called when various auxiliary data is generated. Auxiliary data is passed to user callback as a pointer to the 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);
```

24.7 KYDeviceEventCallBack

The callback function will be called when various device events are generated. Details of an event are passed to user callback as a pointer to the KYDEVICE_EVENT structure. Each event data should be interpreted according to event ID.

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

24.8 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);
```

24.9 UPDATE_CALLBACK

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

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

25 Enumerations

25.1 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. The hardware function failed to execute.
FGSTATUS_BUSY	0x3003	Can't execute the function at the current moment, the FG is busy.
FGSTATUS_FILE_NOT_FOUND	0x3004	Wasn't able to open the file in the given file path.
FGSTATUS_FILE_READ_ERROR	0x3005	Wasn't able to read the file, error in the file or the file is too long.
FGSTATUS_CONFIG_NOT_LOADED	0x3006	Can't load current camera configuration.
FGSTATUS_INVALID_VALUE	0x3007	The value given as a parameter is out of the acceptable range.
FGSTATUS_MAX_CONNECTIONS	0x3008	No more devices can be connected to the 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 the 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 weren'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 the 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_BUFFER_UNSUPPORTED_SIZE	0x3018	Provided buffer size is unsupported and unable to hold the amount of information needed to be filled in the provided buffer.
FGSTATUS_GRABBER_FIRMWARE_NOT_SUPPORTED	0x3019	This indicates that no authentication HW controller exists.

FGSTATUS_PARAMETER_NOT_WRITABLE	0x301A	Unable to set value for the provided parameter.
FGSTATUS_CANNOT_START_HW_STREAM	0x301B	Unable to start camera acquisition.
FGSTATUS_WRONG_SCHEMA_VERSION	0x301C	Wrong or missed required camera configuration value.
FGSTATUS_CAMERA_OR_GRABBER_SECTION_NOT_ARRAY	0x301D	JSON file section type mismatch.
FGSTATUS_ROOT_IS_NOT_OBJECT	0x301E	Unable to Get next (grabber or camera) parameter from an array.
FGSTATUS_NO_PARAMETER_TYPE	0x301F	Parameter type is not supported.
FGSTATUS_FILE_CREATE_ERROR	0x3020	Unable to create file with the provided file path.
FGSTATUS_COULD_NOT_STOP_STREAM	0x3021	Unable to sto camera acquisition.
FGSTATUS_BUFFER_MEMORY_OVERLAP	0x3022	Buffer memory ovelap with previously announced buffer.
FGSTATUS_UNSUPPORTED_PARAMETER_TYPE	0x3023	Provided data type is unsupported.
FGSTATUS_OPERATION_TIMEOUT	0x3024	Command timeout error on specified channel.
FGSTATUS_OPERATION_BLOCKED	0x3025	Operation blocked on specified channel.
FGSTATUS_PARAMETER_NOT_READABLE	0x3026	An attempt to call a currently unreadable parameter will result in this error.
FGSTATUS_PARAMETER_NO_CONTEXT	0x3027	Provided parameter has no context.
FGSTATUS_EXCEEDED_MAX_CAMERA_CONNECTIONS	0x3100	The number of connected cameras exceeds the maximum allowed connected cameras.
FGSTATUS_QUEUED_BUFFERS_NOT_SUPPORTED	0x3101	Queued buffer does not supported by the hardware.
FGSTATUS_DESTINATION_QUEUE_NOT_SUPPORTED	0x3102	Feature is not yet implemented Move buffers to this queue are reserved for future library enhancements.
FGSTATUS_INVALID_STREAM_INFO_CMD	0x3103	Requested stream information is invalid.
FGSTATUS_INVALID_STREAM_BUFFER_INFO_CMD	0x3104	Requested information about previously announced buffer is invalid.
FGSTATUS_STREAM_NOT_CREATED	0x3105	Reserved.
FGSTATUS_GRABBER_NOT_CONNECTED	0x3106	Specified hardware is not connected.
FGSTATUS_CAMERA_NOT_CONNECTED	0x3107	Specified camera is not connected.
FGSTATUS_GRABBER_NOT_OPENED	0x3108	Specified hardware is not open.
FGSTATUS_CAMERA_NOT_OPENED	0x3109	Specified camera is not open.
FGSTATUS_BUFFER_ALREADY_IN_INPUT_QUEUE	0x310A	Specified buffer is already been moved to queue by the application.
FGSTATUS_STREAM_CANNOT_LOCK	0x310B	Stream is started and cannot be re-started.
FGSTATUS_STREAM_IS_LOCKED	0x310C	Stream is being used and cannot be deleted.
FGSTATUS_CAMERA_NODES_NOT_INITIALIZED	0x3200	Reserved.
FGSTATUS_UPDATE_WRONG_VID	0x3300	Retrieved information about firmware contained in supplied binary file is wrong (vendor IDs).
FGSTATUS_UPDATE_WRONG_BOARD_ID	0x3301	Retrieved information about firmware contained in supplied binary file is wrong (board ID).
FGSTATUS_CANNOT_WRITE_IMAGE	0x3400	Unable to save the specified image file.
FGSTATUS_FACILITY_DISABLED	0x3FFD	Camera stopped when trying to detect retaining running camera. Unable to release previous stream.
FGSTATUS_FEATURE_NOT_IMPLEMENTED	0x3FFE	Unable to change parameter specified by the user if it is not implemented or if the user should not be able to change it.
FGSTATUS_UNKNOWN_ERROR	0x3FFF	Unknown error.

25.2 CSSTATUS

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

Enumeration Field	Value	Description
CSSTATUS_OK	0x2000	The operation has successfully executed.
CSSTATUS_UNKNOWN_SIM_HANDLE	0x2001	Unknown API handle.
CSSTATUS_HW_NOT_FOUND	0x2002	Error with hardware. The hardware function failed to execute.
CSSTATUS_BUSY	0x2003	Can't execute the function at the current moment, the simulator is busy.
CSSTATUS_FILE_NOT_FOUND	0x2004	Wasn't able to open file in given file path.
CSSTATUS_FILE_READ_ERROR	0x2005	Wasn't able to read the file, error in the file or the file is too long.
CSSTATUS_CONFIG_NOT_LOADED	0x2006	Can't load current camera configuration.
CSSTATUS_INVALID_VALUE	0x2007	The value given as a parameter is out of the acceptable range.
CSSTATUS_MAX_CONNECTIONS	0x2008	No more devices can be connected to system.
CSSTATUS_COULD_NOT_STOP	0x2009	The simulation couldn't stop.
CSSTATUS_CANNOT_LOAD_IMAGE_FILE	0x200A	Wasn't able to load image file.
CSSTATUS_MEMORY_ERROR	0x200B	General memory error or an allocation has failed.
CSSTATUS_UNKNOWN_SIM_CONTROL	0x200C	Wrong simulation command.
CSSTATUS_WRONG_PARAMETER_NAME	0x200D	Camera configuration wasn't found.
CSSTATUS_WRONG_PARAMETER_TYPE	0x200E	Wrong parameter type for camera configuration.
CSSTATUS_GENICAM_EXCEPTION	0x200F	General Gen<i>Cam exception.
CSSTATUS_OUT_OF_RANGE_ADDRESS	0x2010	The specified address is not suitable for writing.
CSSTATUS_PATH_INVALID	0x2011	The specified file couldn't be opened, wrong file path or file not found.
CSSTATUS_FILE_TYPE_INVALID	0x2012	Invalid file type was entered.
CSSTATUS_UNSUPPORTED_IMAGE	0x2013	Mounted image is not supported.
CSSTATUS_UNSUPPORTED_IMAGE_CONVERSION	0x2014	Couldn't convert image type or couldn't rescale current image.
CSSTATUS_UNSUPPORTED_DEPTH_CONVERSION	0x2015	Couldn't convert the image due to unsupported color depth.
CSSTATUS_INVALID_VALUES_FILE	0x2016	Invalid camera configuration values file was loaded.
CSSTATUS_FILE_WRITE_ERROR	0x2017	Wasn't able to save the camera configuration values file.
CSSTATUS_BUFFER_NOT_LOADED	0x2018	An incompatible buffer was selected, or no buffer was allocated.
CSSTATUS_TRIGGER_NOT_SET	0x2019	Unable to start triggered generation, was not set.
CSSTATUS_CANNOT_SET_USER_REGISTER_ADDRESS	0x201A	Unable to set user's select start address.
CSSTATUS_CANNOT_READ_USER_REGISTER	0x201B	Unable to read user's data chunks (in bytes).
CSSTATUS_CANNOT_WRITE_USER_REGISTER	0x201C	Unable to write user's data chunks (in bytes).
CSSTATUS_CANNOT_WRITE_REGISTER	0x201D	Unable to write to the requested register.
CSSTATUS_IMAGE_HEADER_INJECTION_SIZE_TOO_BIG	0x201E	Requested image header size is too big.
CSSTATUS_NO_EXTENDED_HW_FEATURES	0x201F	Unable to read user's register address.
CSSTATUS_MAX_USER_ADDRESS_EXCEEDED	0x2020	Unable to write user's register address.
CSSTATUS_UNKNOWN_ERROR	0x2FFF	Unknown error.

25.3 KY_DEVICE_PROTOCOL

Structure Field	Value	Description
KY_DEVICE_PROTOCOL_CoaXPress	0x0	Defines the device protocol as CoaXPress.
KY_DEVICE_PROTOCOL_CLHS	0x1	Defines the device protocol as CLHS.
KY_DEVICE_PROTOCOL_GigE	0x2	Defines the device protocol as GigE.
KY_DEVICE_PROTOCOL_Mixed	0xFF	Reserved for future use.
KY_DEVICE_PROTOCOL_Unknown	0xFFFF	Unknown protocol.

25.4 KY_STREAM_BUFFER_INFO_CMD

Stream Buffer configuration information.

Structure 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 the previous timestamp.
KY_STREAM_BUFFER_INFO_IMAGEID	16	Image ID as reported by a transport protocol, e.g. CXP's "source image index"
KY_STREAM_BUFFER_INFO_ID	1000	Unique id of buffer in the stream.
KY_STREAM_BUFFER_INFO_STREAM_HANDLE	1001	Handle of a stream to which this buffer belongs

25.5 KY_STREAM_INFO_CMD

Stream information.

Structure Field	Value	Description
KY_STREAM_INFO_PAYLOAD_SIZE	7	Size of the expected data in bytes, of currently configured stream.
KY_STREAM_INFO_BUF_ALIGNMENT	13	The alignment of memory allocated for a buffer, in bytes.
KY_STREAM_INFO_PAYLOAD_SIZE_INCREMENT_FACTOR	1000	Payload size (should be divisible by increment factor).
KY_STREAM_INFO_ACTUAL_PAYLOAD_SIZE	1001	Size of the actual data in bytes that will be delivered, for current device configurations (change every update of parameters which define the image dimensions).
KY_STREAM_INFO_BUF_COUNT	2000	The number of buffers in the stream.
KY_STREAM_INFO_INSTANTFPS	2001	Last calculated FPS. Valid after at least two frames have been acquired.
KY_STREAM_INFO_CAMHANDLE	2002	CAMHANDLE of the camera to which this stream belongs. May be NULL_CAMHANDLE in future enhancements with multi-camera streams.

25.6 KY_DATA_TYPE

Data types information.

Structure Field	Value	Description
KY_DATATYPE_UNKNOWN	0	Unknown data type
KY_DATATYPE_STRING	1	NULL-terminated C string (ASCII encoded)
KY_DATATYPE_STRINGLIST	2	Concatenated INFO_DATATYPE_STRING list. End of list is signaled with an additional NULL
KY_DATATYPE_INT16	3	Signed 16 bit integer
KY_DATATYPE_UINT16	4	Unsigned 16 bit integer
KY_DATATYPE_INT32	5	Signed 32 bit integer
KY_DATATYPE_UINT32	6	Unsigned 32 bit integer
KY_DATATYPE_INT64	7	Signed 64 bit integer
KY_DATATYPE_UINT64	8	Unsigned 64 bit integer
KY_DATATYPE_FLOAT64	9	Signed 64 bit floating point number
KY_DATATYPE_PTR	10	Pointer type (void*). Size is platform dependent (32 bit on 32 bit platforms)
KY_DATATYPE_BOOL8	11	Boolean value occupying 8 bit. 0 for false and anything for true
KY_DATATYPE_SIZET	12	Platform dependent unsigned integer (32 bit on 32 bit platforms)
KY_DATATYPE_BUFFER	13	Like a INFO_DATATYPE_STRING but with arbitrary data and no NULL termination
KY_DATATYPE_STREAM_HANDLE	1001	STREAM_HANDLE
KY_DATATYPE_CAMHANDLE	1002	CAMHANDLE

25.7 KY_ACQ_QUEUE_TYPE

Queued Buffer configuration information.

Structure Field	Value	Description
KY_ACQ_QUEUE_INPUT	0	Buffers in the INPUT queue are ready to be filled with data.
KY_ACQ_QUEUE_OUTPUT	1	Buffers in the OUTPUT queue have been filled and awaiting user processing. This queue is filled internally by the library. The ability of the application to move buffers to this queue is reserved for future library enhancements and currently will result in the error code FGSTATUS_DESTINATION_QUEUE_NOT_SUPPORTED.
KY_ACQ_QUEUE_UNQUEUED	2	Buffers in the UNQUEUED set have been announced but are inactive for the acquisition mechanism. By default, all buffers are placed in the UNQUEUED set.
KY_ACQ_QUEUE_AUTO	3	Buffers in the AUTO queue are managed automatically, in a cyclic matter, without the need for the user to re-queue them.

25.8 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

25.9 KY_CAM_PROPERTY_TYPE

Gen<i>Cam field type. Camera configuration field type as stated in the 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

25.10 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

25.11 VIDEO_DATA_TYPE

Data width of the pixel, defined in section 9.4.1.2 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 the 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 of the 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 of the 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 of the 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 of the JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_UCBCR_xxx.
DATA_TYPE_INVALID	0xFF	Invalid data type.

25.12 VIDEO_DATA_SUBTYPE H

Data width of the pixel, defined in section 9.4.1.2 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
DATA_SUBTYPE_INVALID	0x0F	Invalid data type.

25.13 KYFGLIB_CONCURRENCY_FLAGS

The KYFGLIB_CONCURRENCY_SA_RESTART flag can be added to the "concurrency_mode" field of the passed KYFGLib_InitParameters instance to the KYFGLib_Initialize() function, before opening any grabbers, cameras and starting streams. Please refer to Linux blocking calls and signals in the streaming thread for more detailed information.

Enumeration Field	Value	Description
KYFGLIB_CONCURRENCY_SA_RESTART	1	Only valid in Linux interrupts compatible with blocking calls in the same thread.

25.14 PORT_STATUS

Enumeration Field	Value	Description
PORT_DISCONNECTED	0x00	The port is disconnected, no link has been established
PORT_SYNCHRONIZED	0x01	Port link is synchronized and awaiting connection
PORT_CONNECTING	0x10	A connection is trying to be established on port
PORT_CONNECTED	0x11	Port is connected and assigned to a camera

25.15 SUBMIT_BUFF_FLAGS

Enumeration Field	Value	Description
SUBMIT_BUFF_REGULAR_BUFFER	0x00	Regular buffer
SUBMIT_BUFF_PHYSICAL_ADDRESS	0x01	Provided buffer addresses are physical addresses

25.16 VIDEO_SOURCE_TYPE

Chameleon simulator video source types.

Structure Field	Value	Description
VIDEO_SOURCE_NONE	-1	The video source is not specified.
VIDEO_SOURCE_PATTERN	0	Video source PATTERN_TYPE specified in the section below.
VIDEO_SOURCE_FILE	1	Video source file types ".bmp", ".tif", ".pgn", and ".raw" are supported.
VIDEO_SOURCE_FOLDER	2	The video source folder may contain single or several files. The type and number of files should be specified.

25.17 PATTERN_TYPE

Chameleon simulator patterns generation types. Actual pattern scaling and format are defined by values in the appropriate camera configuration fields.

Structure Field	Value	Description
PATTERN_XRAMP	0	Ramp pattern over X coordinate
PATTERN_XRAMP_COLOR	1	Ramp pattern over X coordinate with color (3 color format)
PATTERN_YRAMP	2	Ramp pattern over Y coordinate
PATTERN_YRAMP_COLOR	3	Ramp pattern over Y coordinate with color (3 color format)
PATTERN_XYRAMP	4	Ramp pattern over XY coordinates
PATTERN_XYRAMP_COLOR	5	Ramp pattern over XY coordinates with color (3 color format)
PATTERN_FIXED	6	Fixed color pattern

26 Structures

26.1 KY_SOFTWARE_VERSION

Structure Field	Type	Description
Should be set before calling KY_GetSoftwareVersion() function, (input part):		
struct_version	uint16_t	Must be set to 0…2 before calling KY_GetSoftwareVersion() function

Filled by KY_GetSoftwareVersion() function (output part):		
Major	uint16_t	Software major version
Minor	uint16_t	Software minor version
SubMinor	uint16_t	Software sub minor version

Filled if “struct” version is 1 or greater:		
Beta	uint16_t	Non-zero value indicates a "Beta build"
RC	uint16_t	Non-zero value indicates a "Release Candidate build"

Filled if “struct” version is 2 or greater:		
Alpha	uint16_t	Non-zero value indicates a "Alpha build"

26.2 KYFGLib_InitParameters

Structure Field	Type	Description
version	uint32_t	The version of this structure definition. Must be 1 or 2.
concurrency_mode	uint32_t	Combination of KYFGLIB_CONCURRENCY_FLAGS, all unused bits must be set to 0. [1]
logging_mode	uint32_t	Reserved, must be set to 0.
noVideoStreamProcess	KYBOOL	Since version 2 of structure. Use library without requesting video stream facilities, e.g for camera control only.[1]
ky_aligned_malloc[2]	ky_aligned_malloc_t	A user-defined function used by "KYFG_BufferAllocAndAnnounce()" to allocate acquisition buffers.

Remarks:

1. See **important notes** regarding the usage of KYFGLIB_CONCURRENCY_FLAGS for Linux users.
2. Users who want this member to be considered by KYFGLib_Initialize() call. Must set the version value to 2 or higher.
3. Usage sample in the "KYFGLib_Example.c" - User-defined function used by "KYFG_BufferAllocAndAnnounce()" to allocate acquisition buffers

26.3 KY_DEVICE_INFO

Structure Field	Type	Description
version	uint32_t	The version of this structure definition. On input must be no greater than KY_MAX_DEVICE_INFO_VERSION, value 0 is treated as 1. On output indicates version supported by current library
szDeviceDisplayName	char	The display name of the device.
nBus	int	Device PCIe bus.
nSlot	int	Device PCIe slot.
nFunction	int	Device function.

DevicePID	uint32_t	Unique device ID.
isVirtual	KYBOOL	Indicates whether the device is virtual.
m_Flags	uint8_t	Mask KY_DEVICE_STREAM_GRABBER indicates device that supports grabbing (input streams). Mask KY_DEVICE_STREAM_GENERATOR indicates device that supports generation (output streams).
m_Protocol	KY_DEVICE_PROTOCOL	For possible values see the table below.
DeviceGeneration	uint32_t	Device generation (1 or 2).

26.4 KYFGCAMERA_INFO

Camera configuration information. These fields are updated when the camera is connected using the 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 a virtual camera, i.e. internally implemented stream, which does not represent any physical camera, KYFALSE otherwise. This parameter can be KYTRUE only in the case of custom firmware implementations.

26.5 KYFGCAMERA_INFO2

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
version	uint32_t	The version of the structure. For API versions prior to 6.2.0 must be set to 0. Since version 6.2.0, can be set to 1.
master_link	uint8_t	The master link channel.
link_mask	uint8_t	The mask of connected links.
link_speed	CXP_LINK_SPEED	The current connection speed (according to CoaXPress specification).
deviceVersion	char [65]	Camera version.
deviceVendorName	char [65]	Vendor name.
deviceManufacturerInfo	char [65]	Additional manufacturer info.
deviceModelName	char [65]	Camera model name.
deviceID	char [65]	Device id.
deviceUserID	char [65]	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 a virtual camera, i.e. internally implemented stream, which does not represent any physical camera, KYFALSE otherwise. This parameter can be KYTRUE only in the case of custom firmware implementations.
deviceFirmwareVersion	char [65]	Camera firmware version. This field was introduced in API 6.2.0, and will only be filled by the function KYFG_CameraInfo2() if 'version' field is set to 1.

26.6 VIDEO_PIXELIF

The pixel format code is formed as shown in Table 24 of the 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)

26.7 KYFG_AUX_DATA

Auxiliary data structure holding information of received Auxiliary message.

Structure Field	Type	Description
messageID	uint32_t	Unique ID of a received auxiliary data message. For possible values see Table 2.
reserved	KYBOOL	Reserved for future use.
dataSize	size_t	Size, in Bytes, of data delivered with Auxiliary message.
u_data	union	See remarks
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.

Remarks:

1. Starting with version 6.0 the KYFG_AUX_DATA structure definition has changed. Usage of this structure requires client code changes - adding "u_data." to where these struct members are accessed, to avoid compilation warning about "nonstandard extension used: nameless struct/union".

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

Table 2 – MessageID possible values

26.8 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 the allocated frame buffer
timestamp	uint64_t	Frame arrival timestamp in units of nano-seconds (nsec)

26.9 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 the table in the remarks.[1][2][3][4][5]
timestamp	uint64_t	Event timestamp in units of nano-seconds (nsec)

Remarks:

1. 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

26.10 KYDEVICE_EVENT

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

Structure Field	Type	Description
eventId	KYDEVICE_EVENT_ID	Unique ID of received device event. For possible values see the table below.

Device event ID possible values	
KYDEVICE_EVENT_CAMERA_START_REQUEST	The device detected a remote request to start transmission on a camera.
KYDEVICE_EVENT_CAMERA_CONNECTION_LOST_ID	The device detected a remote request to start transmission on a lost camera.
KYDEVICE_EVENT_SYSTEM_TEMPERATURE_ID	Device temperature reached a threshold.
KYDEVICE_EVENT_CXP2_HEARTBEAT_ID	CXP 2.0 Heartbeat packet received from connected camera
KYDEVICE_EVENT_CXP2_EVENT_ID	CXP 2.0 Event packet received from connected camera
KYDEVICE_EVENT_GENCP_EVENT_ID	GenCP Event message id for CLHS
KYDEVICE_EVENT_GIGE_EVENTDATA_ID	EVENTDATA Event message id for 10GigE

Table 4 – Device event ID possible values

26.11 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 the KYDeviceEventCallBack function when the '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 KYFG_CameraStart() function to start acquisition on the specified camera after performing application-specific preparation

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

26.12 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 the KYDeviceEventCallBack function when the 'eventId' field is KYDEVICE_EVENT_CAMERA_CONNECTION_LOST_ID. This event is sent when a link loss occurs on a detected camera.

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	The base part of the structure.
camHandle	CAMHANDLE	Identifies camera for which connection loss was detected.
iDeviceLink	size_t	The ID of the link on the device side, which lost connection.
iCameraLink	size_t	The ID of the link on the camera side as it was originally detected.

Remarks:

1. Camera connection loss event is currently implemented for CoaXPress and CLHS Frame Grabbers only. For 10Gige Frame Grabbers, the camera connection loss event will not be received from the callback event.

26.13 KYDEVICE_EVENT_SYSTEM_TEMPERATURE

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_SYSTEM_TEMPERATURE_ID. This structure is derived from KYDEVICE_EVENT and passed to the KYDeviceEventCallBack function when the ‘eventId’ field is KYDEVICE_EVENT_SYSTEM_TEMPERATURE_ID. This event is sent when the device temperature reached a threshold.

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	The base part of the structure.
temperatureThresholdId	KYDEVICE_EVENT_SYSTEM_TEMPERATURE_THRESHOLDS_ID	Identifies if a device temperature threshold had been crossed.

KYDEVICE_EVENT_SYSTEM_ TEMPERATURE _NORMAL	Device temperature is normal.
KYDEVICE_EVENT_SYSTEM_ TEMPERATURE _WARNING	Device temperature reached the warning threshold. The application should decrease device load, for example, stop a running acquisition.
KYDEVICE_EVENT_SYSTEM_TEMPERATURE_CRITICAL	Device temperature reached a critical threshold. The KAYA library may reset the device and the application should perform necessary preparation.

26.14 KYDEVICE_EVENT_CXP2_HEARTBEAT

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_CXP2_HEARTBEAT_ID. This structure is derived from KYDEVICE_EVENT and passed to the KYDeviceEventCallBack function when the ‘eventId’ field is KYDEVICE_EVENT_CXP2_HEARTBEAT_ID.

The usage is demonstrated in "KYDeviceEventCallBackImpl" function, located at:
<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KYFGLibExampleQueuedBuffers.c

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	The base part of the structure.
camHandle	CAMHANDLE	Identifies camera for which connection loss was detected.
heartBeat	KY_CXP2_HEARTBEAT	The structure that holds information on the received CXP2 heartbeat.

26.15 KY_CXP2_HEARTBEAT

Data structure holding information of received KY_CXP2_HEARTBEAT event from the device.

```
typedef struct _CXP_HEARTBEAT
{
        uint32_t masterHostConnectionID;
        uint64_t cameraTime;
}KY_CXP2_HEARTBEAT;
```

Structure Field	Type	Description
masterHostConnectionID	uint32_t	According to CXP 2.0 standard, holds the Host Connection ID of the Host connection connected to the Device Master connection.
cameraTime	uint64_t	The Device time is expressed in nanoseconds. Note that this field is called "Device time" in the CXP standard but in this API the notion "device" is used for PCI devices - grabbers and simulators.

26.16 KYDEVICE_EVENT_CXP2_EVENT

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_CXP2_EVENT_ID. This structure is derived from KYDEVICE_EVENT and passed to the KYDeviceEventCallBack function when the 'eventId' field is KYDEVICE_EVENT_CXP2_EVENT_ID.

The usage is demonstrated in "KYDeviceEventCallBackImpl" function, located at:
<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KYFGLibExampleQueuedBuffers.c

Structure Field	Type	Description
deviceEvent	KYDEVICE_EVENT	The base part of the structure.
camHandle	CAMHANDLE	API handle to the connected camera.
cxp2Event	KY_CXP2_EVENT	A structure that holds information on the received CXP2 event.

26.17 KYDEVICE_EVENT_GENCP_EVENT

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_GENCP_EVENT. This structure is derived from KYDEVICE_EVENT and passed to the KYDeviceEventCallBack function when the 'eventId' field is KYDEVICE_EVENT_GENCP_EVENT_ID.

The usage is demonstrated in "KYDeviceEventCallBackImpl" function, located at:
<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KYFGLibExampleQueuedBuffers.c

Structure Field	Type	Description
eventSize	uint16_t	Size of event data objects in bytes including event_size, event_id, timestamp and optional data.
eventId	uint16_t	The event_id is a number identifying an event source.
timestamp	uint64_t	64 bit timestamp value in ns as defined in the timestamp bootstrap register.
data	uint8_t	Data payload with the valid size of 'dataSize'.

26.18 KYDEVICE_EVENT_GIGE_EVENTDATA

Data portion interpretation of device event structure when event is KYDEVICE_EVENT_GIGE_EVENTDATA. This structure is derived from KYDEVICE_EVENT and passed to the KYDeviceEventCallBack function when the 'eventId' field is KYDEVICE_EVENT_GIGE_EVENTDATA_ID.
The usage is demonstrated in "KYDeviceEventCallBackImpl" function, located at:
<Public Documents>/KAYA Instruments/Vision Point/API Samples/Vision Point API/KYFGLibExampleQueuedBuffers.c

Structure Field	Type	Description
eventSize	uint16_t	Size of event data objects in bytes including event_size, event_id, timestamp and optional data.
eventId	uint16_t	The event_id is a number identifying an event source.
streamChannel	uint16_t	Index of stream channel associated with this even. 0xFFF if no stream channel is involved.
blockId	uint16_t	The ID of the data block associated with this event. 0 if no blockId is associated to this event.
timestamp	uint64_t	64 bit timestamp value in ns as defined in the timestamp bootstrap register.
data	uint8_t	Data payload with the valid size of 'dataSize'.

26.19 KY_CXP2_EVENT

Data structure holding information of received KY_CXP2_EVENT event from the device.

Structure Field	Type	Description
masterHostConnectionID	uint32_t	According to CXP 2.0 standard, holds the Host Connection ID of the Host connection connected to the Device Master connection.
tag	uint8_t	8 bit tag. Incremented for each new Event packet.
dataSize	uint16_t	The number of event data words, Maximum size = CXP_EVENT_MAX_DATA_SIZE bytes according to CXP 2.0 standard.
dataWords	uint32_t	'dataSize' words with one or more event messages.

```
#define KY_CXP_EVENT_MAX_DATA_SIZE 1024   //according to CXP 2.0 standard
typedef struct _KY_CXP2_EVENT
{
        uint32_t masterHostConnectionID;
        uint8_t  tag;
        uint16_t dataSize;
        uint32_t dataWords[KY_CXP_EVENT_MAX_DATA_SIZE / 4];

}KY_CXP2_EVENT;
```

26.20 KY_CXPEVENT_PACK

Structure Field	Type	Description
nDataWords	uint16_t	Number of data WORDS in 'eventDataWord'
eventDataWord[256]	uint32_t	according to CXP standard, there can be maximum 1024 bytes of data

26.21 NodeDescriptor

A descriptor of a node passed to the 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	The current value in the 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	The current value in the case parameter has 'intfIFloat' interfaceType.
curBoolValue	bool	The current value in the case parameter has 'intfIBoolean' interfaceType.
curStringValue	const char*	The current value in the case parameter has 'intfIString' interfaceType.
isSelector	bool	`true' if this node acts as a selector for other nodes, 'false' otherwise.
selectorName	const char*	Name of another node that acts as a selector for this node, NULL if this node is not selected.
pParentNode	NodeDescriptor*	Pointer to the parent node in nodes hierarchy.

26.21.1 ParameterInterfaceType

Name	Description
intfIValue	Not currently used.
intfIBase	Not currently used.
intfIInteger	The parameter is of integer type. Get/Set operations to expect 'int64_t' C type.
intfIBoolean	The parameter is of boolean type. Get/Set operations to expect 'bool' C type.
intfICommand	Not currently used.
intfIFloat	The parameter is of float type. Get/Set operations to expect 'double' C type.
intfIString	The parameter is of string type. Get/Set operations to expect 'char *' C type.
intfIRegister	Not currently used.
intfICategory	The category of the parameter. Used for parameters grouping.
intfIEnumeration	The parameter is of enumeration type. Get/Set operations to 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 to expect 'int64_t' C type.
intfIPort	Not currently used.

26.21.2 ParameterRepresentation

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

Name	Description
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	An IPV4 Address editor.
MACAddress	A MAC Address editor.
_UndefinedRepresentation	No suggested representation

26.21.3 NodeDescriptorType

The type of NodeDescriptor can be of the following:

Name	Description
Invalid	An error has occurred.
NewNode	A new node is being announced during the enumeration of all nodes.
NewEnumEntry	A new entry of the previously announced node of type 'intfIEnumeration' is being announced.
UpdateNode	The current value of the described parameter has been changed.

26.22 KY_AuthKey

Authentication key secret character array

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

26.23 UPDATE_STATUS

Firmware update progress supplied via a 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 struct in the future code will be changed and initialize 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.

26.24 KYFGLib_CameraScanParameters

Structure Field	Type	Description
deviceEvent	uint32_t	The version of this structure definition. Must be 1.
pCamHandleArray	CAMHANDLE*	An array of CAMHANDLE is allocated by the caller.
nCameraCount	int	On entry number, CAMHANDLEs allocated in 'pCamHandleArray', on exit number of detected cameras.
bRetainOpenCameras	KYBOOL	KYTRUE if scan should skip currently active links and detect only new connections. The open camera handles will not be affected by KYFG_CameraScanEx() call and will be retained at the same places of the array where they were returned by the previous call, except for camera(s) that were closed between calls. KYFALSE if all currently open camera handles should be reset and full re-scan should be performed.

27 Building an API Example

27.1 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.

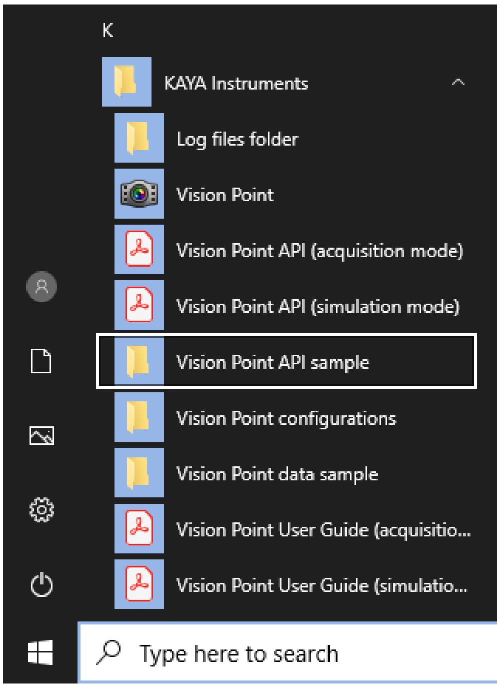

K
KAYA Instruments
Log files folder
Vision Point
Vision Point API (acquisition mode)
Vision Point API (simulation mode)
Vision Point API sample
Vision Point configurations
Vision Point data sample
Vision Point User Guide (acquisitio...
Vision Point User Guide (simulatio...
Type here to search

Figure 3 – API Samples folder from the quick start menu path

2. Choose a 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.*

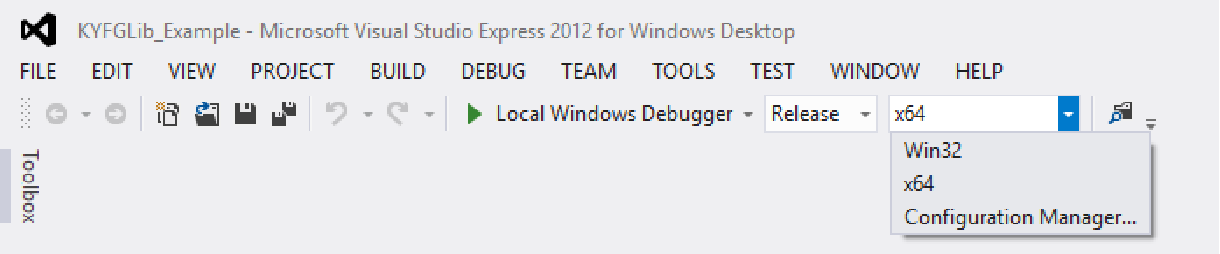
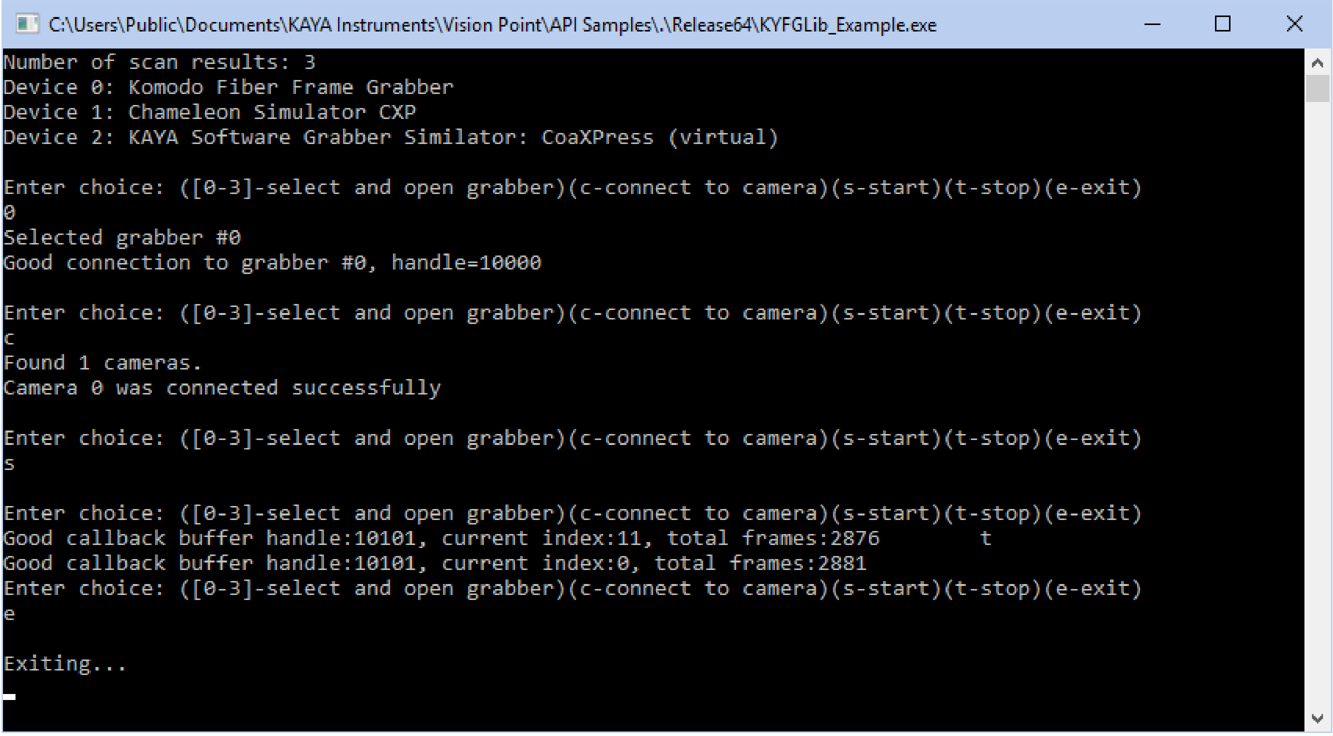


Figure 4 – Visual Studio choosing a solution platform

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]	Select and open device
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 Fiber Frame Grabber
Device 1: Chameleon Simulator CXP
Device 2: KAYA Software Grabber Similator: CoaXPress (virtual)

Enter choice: ([0-3]-select and open grabber)(c-connect to camera)(s-start)(t-stop)(e-exit)
0
Selected grabber #0
Good connection to grabber #0, handle=10000

Enter choice: ([0-3]-select and open grabber)(c-connect to camera)(s-start)(t-stop)(e-exit)
c
Found 1 cameras.
Camera 0 was connected successfully

Enter choice: ([0-3]-select and open grabber)(c-connect to camera)(s-start)(t-stop)(e-exit)
s

Enter choice: ([0-3]-select and open grabber)(c-connect to camera)(s-start)(t-stop)(e-exit)
Good callback buffer handle:10101, current index:11, total frames:2876        t
Good callback buffer handle:10101, current index:0, total frames:2881
Enter choice: ([0-3]-select and open grabber)(c-connect to camera)(s-start)(t-stop)(e-exit)
e

Exiting...
```

Figure 5 – Running an API example for Windows

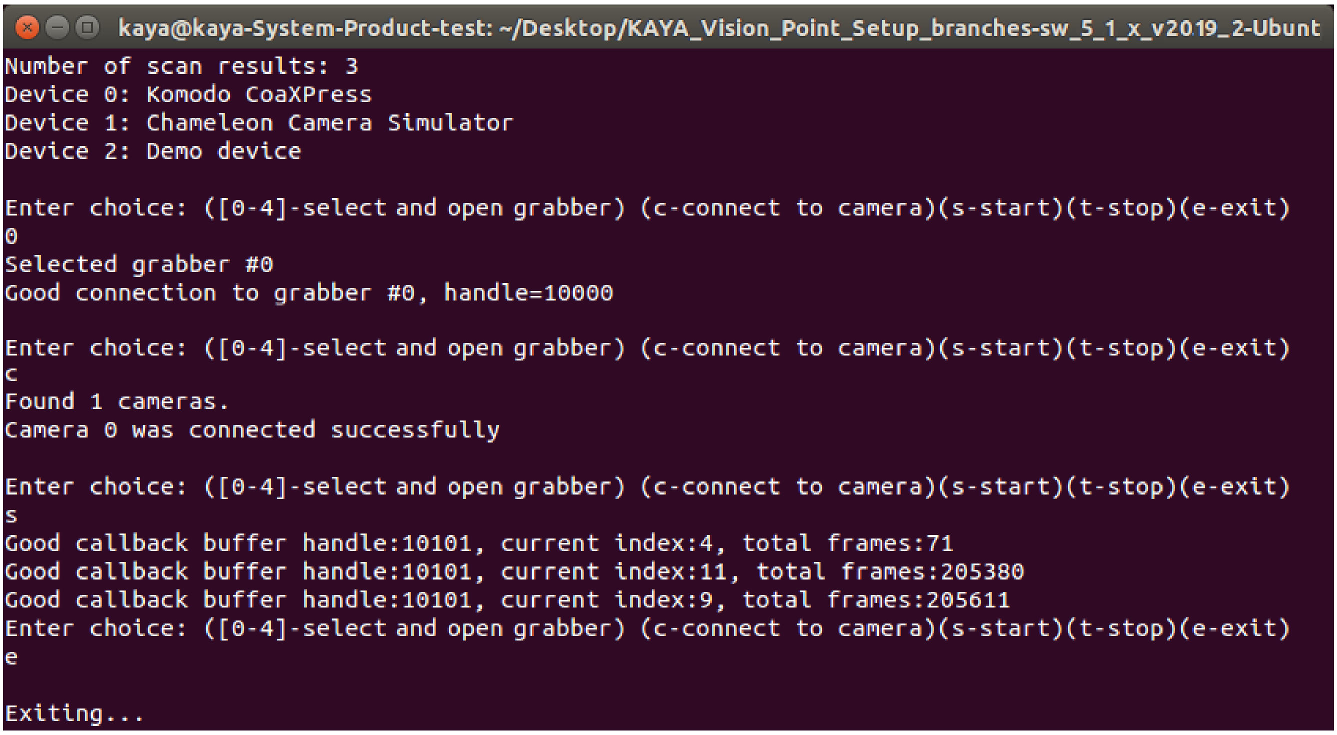
NOTE: *By default, the KYFGLib_Example.vcxproj project contains KYFGLib_Example.c which uses Cyclic Buffer. In order to use Queued Buffer, in the loaded project, change the KYFGLib_Example.c file to KYFGLib_Example_QueuedBuffers.c located in the same directory, and rebuild example*.

27.2 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.
3. To run the API Example, simply type "./KYFGLib _Example" followed by "Enter".
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]	Select and open device
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_5_1_x_v2019_2-Ubunt

```
Number of scan results: 3
Device 0: Komodo CoaXPress
Device 1: Chameleon Camera Simulator
Device 2: Demo device

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

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

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

Exiting...
```

Figure 6 – Running an API example for Linux

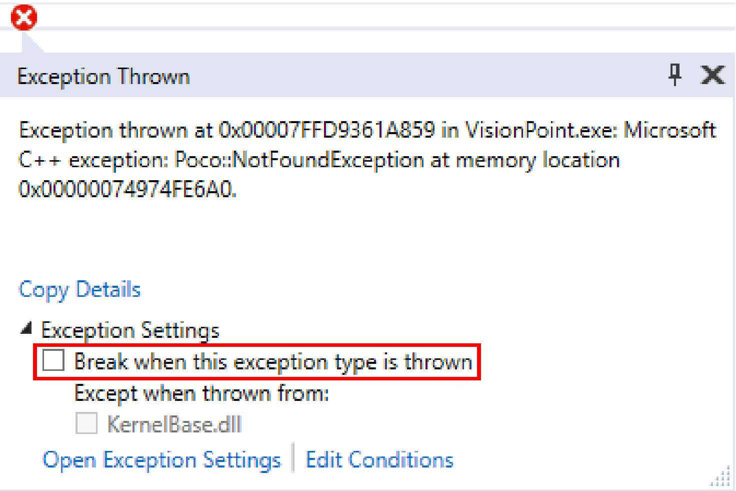
NOTE: *By default, the make file includes KYFGLib_Example.c which uses Cyclic Buffer example. In order to use Queued Buffer, change the KYFGLib_Example.c file to KYFGLib_Example_QueuedBuffers.c located in the same directory, and rebuild example.*

28 Working with the debugger

28.1 C++ exceptions

When an exception is first thrown, whether from the application itself or dependent or platform code, it is known as a "*first-chance*" exception. It is not an actual exception that indicates a real problem, but a debugging instrument for C++ code that uses exceptions in normal flow.

The user can edit the debugger settings to disable those "implicit breakpoints" caused by first chance exceptions. An example of this type of exception is shown in the image below:



Exception Thrown

Exception thrown at 0x00007FFD9361A859 in VisionPoint.exe: Microsoft C++ exception: Poco::NotFoundException at memory location 0x00000074974FE6A0.

☐ Break when this exception type is thrown

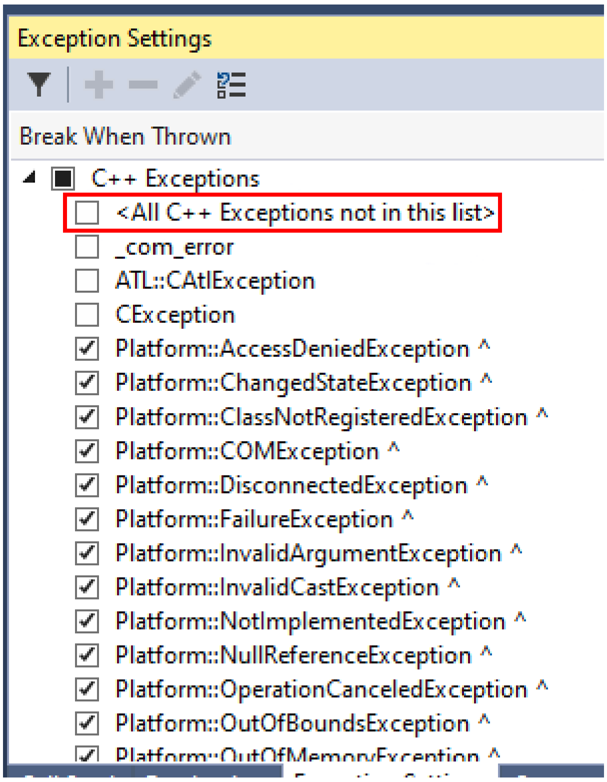
Except when thrown from:

☐ KernelBase.dll

Figure 7 – C++ first-chance exception example

You can find more detailed information on the "first-chance" exceptions in this Microsoft article: https://docs.microsoft.com/en-us/security-risk-detection/concepts/first-chance-exception

The "*first-chance*" exceptions can be managed by either disabling all C++ exceptions except those explicitly enabled, as shown in the image below:



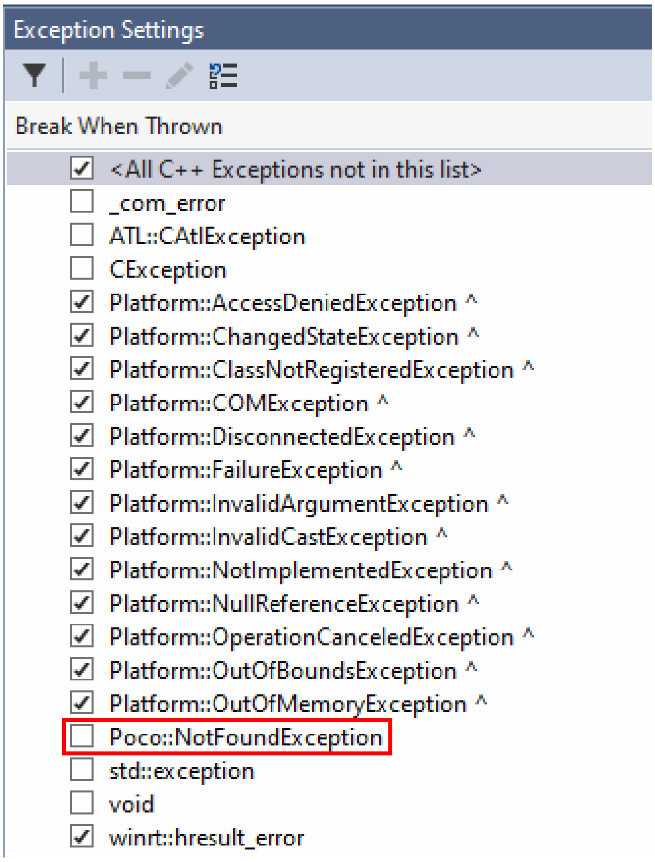
Exception Settings

Break When Thrown

- ■ C++ Exceptions
  - ☐ <All C++ Exceptions not in this list>
  - ☐ _com_error
  - ☐ ATL::CAtlException
  - ☐ CException
  - ☑ Platform::AccessDeniedException ^
  - ☑ Platform::ChangedStateException ^
  - ☑ Platform::ClassNotRegisteredException ^
  - ☑ Platform::COMException ^
  - ☑ Platform::DisconnectedException ^
  - ☑ Platform::FailureException ^
  - ☑ Platform::InvalidArgumentException ^
  - ☑ Platform::InvalidCastException ^
  - ☑ Platform::NotImplementedException ^
  - ☑ Platform::NullReferenceException ^
  - ☑ Platform::OperationCanceledException ^
  - ☑ Platform::OutOfBoundsException ^
  - ☑ Platform::OutOfMemoryException ^

Figure 8 – Disabling all exceptions, except those explicitly enabled

Or by enabling all exceptions except those explicitly disabled can be as seen in the image below:



Exception Settings

Break When Thrown

- ☑ <All C++ Exceptions not in this list>
- ☐ _com_error
- ☐ ATL::CAtlException
- ☐ CException
- ☑ Platform::AccessDeniedException ^
- ☑ Platform::ChangedStateException ^
- ☑ Platform::ClassNotRegisteredException ^
- ☑ Platform::COMException ^
- ☑ Platform::DisconnectedException ^
- ☑ Platform::FailureException ^
- ☑ Platform::InvalidArgumentException ^
- ☑ Platform::InvalidCastException ^
- ☑ Platform::NotImplementedException ^
- ☑ Platform::NullReferenceException ^
- ☑ Platform::OperationCanceledException ^
- ☑ Platform::OutOfBoundsException ^
- ☑ Platform::OutOfMemoryException ^
- ☐ Poco::NotFoundException
- ☐ std::exception
- ☐ void
- ☑ winrt::hresult_error

Figure 9 – Enabling all exceptions, except those explicitly disabled

More detailed information on how to manage You can find "first-chance" exceptions in this Microsoft article: https://docs.microsoft.com/en-us/visualstudio/debugger/managing-exceptions-with-the-debugger?view=vs-2019 (Make sure to expand the "C++ Exceptions" group of "Exception Settings")

Note: *You can prevent the first chance warning from appearing in the output window of Visual Studio using its context menu (right-click) when it is showing output from "Debug."*

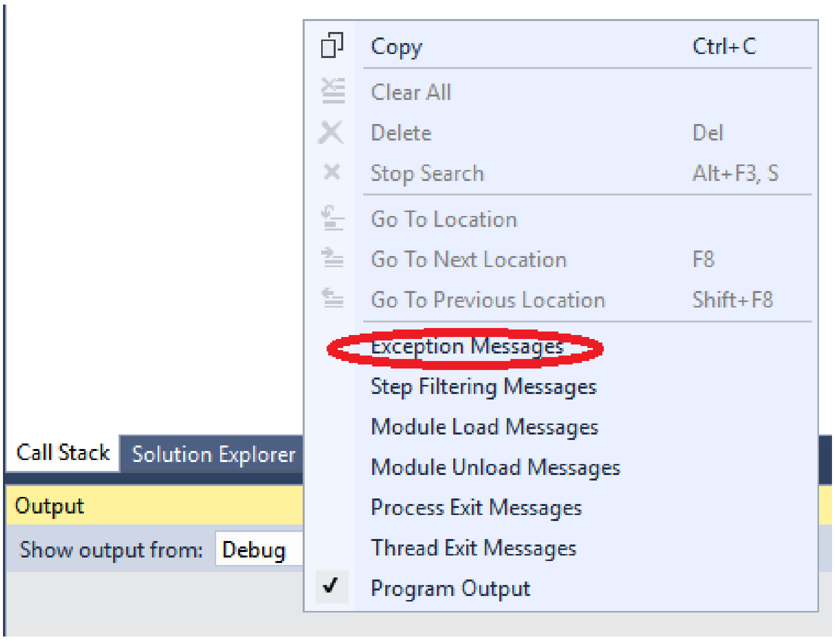


Figure 10 –Preventing the first chance warning from appearing

28.2 gdb SIG44 handling

When running KAYA's KYFGLib library under a debugging tool, the signal (SIG44), sent from kernel to the user space stops the application.

In order to prevent the debugger from stopping the application when signals are sent, it is advised to setup gdb to ignore them during debugging.

For example:

1. Create the "~/.gdbinit" file
2. Add the "handle SIG44 nostop noprint pass" string to this file
3. Restart the environment

29 Frame Grabber Operational Configurations

29.1 Frame Grabber General Configuration for Operation

In general, there is no need to configure the Frame Grabber to achieve basic operation. However, certain advanced features activation requires a Frame Grabber configuration. 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 "KAYA Frame Grabber Feature Guide" document.

29.2 Silent Discovery Mode

The silent-camera-discovery process is mainly used for retransmission 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 an 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 Frame Grabber Feature Guide" document for more details.
3. Make sure the camera is already configured and ready to be connected to. Take into account that no camera Reset or connection reconfiguration commands will be sent.
4. Now camera scan can be initiated using the KYFG_UpdateCameraList() function.

29.3 Komodo 4R4T system configuration

The following configuration should be used with the Komodo or Predator Frame Grabber when setting up the Komodo4R4T transmit channels towards the Frame Grabber receive channels.

29.3.1 Setup

1. Insert the Komodo/Predator Frame Grabber and the Komodo 4R4T Frame Grabber into a PC and connect the power connectors of Komodo 4R4T Frame Grabber. The regular Frame Grabber and the Komodo4R4T Frame Grabber can be installed in a single computer, or in two different computers.
2. Connect a CoaXPress camera or a Chameleon Simulator to one or more of the 4 top DIN connectors (channels 0-3) of the Komodo4R4T using 4 DIN cables.
3. Using different DIN cables, connect the bottom DIN connectors (channels 4-7) of Komodo4R4T to a regular Frame Grabber.
4. Make sure the Komodo4R4T links are connected in the same order (link 0 of the will be retransmitted to link 4). See the image below as reference.

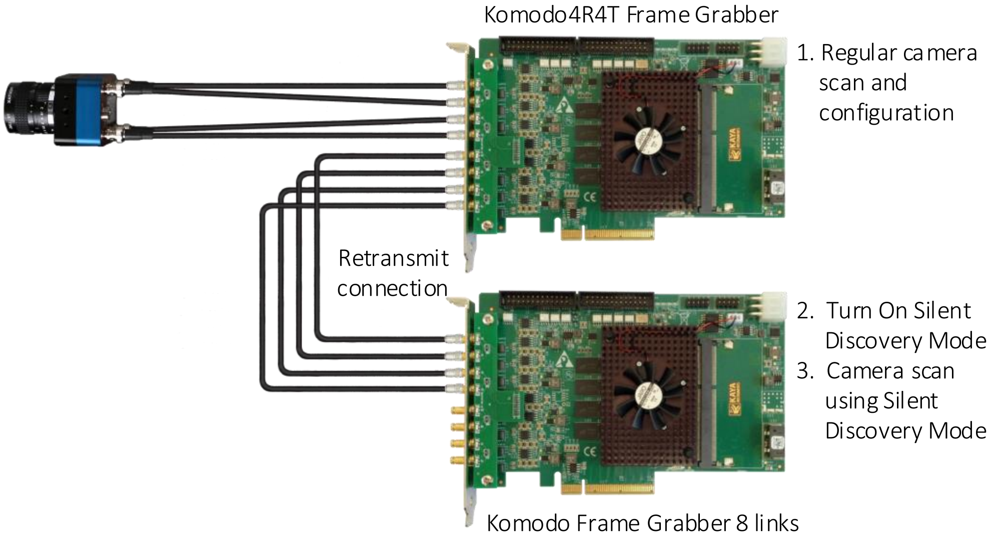

Komodo4R4T Frame Grabber
1. Regular camera scan and configuration
Retransmit connection
2. Turn On Silent Discovery Mode
3. Camera scan using Silent Discovery Mode
Komodo Frame Grabber 8 links

Figure 11 – Retransmission setup example

29.3.2 Configuration

The following steps should be performed on different instances of Vision Point application in order to configure and use the Komodo 4R4T Frame Grabber with regular Frame Grabber:

Step	Komodo 4R4T Frame Grabber	Komodo/Predator Frame Grabber
1	Open Vision Point application and select the Komodo4R4T device	-
2	-	Open an additional instance of the Vision Point application and select the Komodo/Predator regular Frame Grabber device
3	-	Activate the “Silent Discovery Mode”. This option is located in Frame Grabber tab -> Device control category -> Silent Discovery Mode - ON
4	Scan and configure connected camera – this will initiate the camera correctly to be ready for silent discovery. **NOTE:** *I case you are using Chameleon Simulator, please open an additional instance if Vision Point application and configure the Chameleon Simulator to, before step no. 5*	-
5	-	Scan for connected camera
6	-	Start acquisition – this won’t start the acquisition yet
7	Start acquisition – this will initiate acquisition on both Frame Grabbers	-

29.4 **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 an indication of new data acquisition. This prevents the software from receiving frames too frequently thus relieving the CPU operation.

To configure the 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. Since 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 the “KAYA Frame Grabber Feature Guide” document for more details.

30 Appendices

30.1 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 0	KY_OPTO_IN_0
1	OptoCoupled Input 1	KY_OPTO_IN_1
2	OptoCoupled Input 2	KY_OPTO_IN_2
3	OptoCoupled Input 3	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 Output 0	KY_OPTO_OUT_0
21	OptoCoupled Output 1	KY_OPTO_OUT_1
22	OptoCoupled Output 2	KY_OPTO_OUT_2
23	OptoCoupled Output 3	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)

30.2 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.

Value	Source	Gen<i>Cam	I/O	Timer	Trigger	Encoder
0	Disabled	KY_DISABLED	✓	✓	✓	✓
1	OptoCoupled Input 0	KY_OPTO_IN_0	✓	✓	✓	✓
2	OptoCoupled Input 1	KY_OPTO_IN_1	✓	✓	✓	✓
3	OptoCoupled Input 2	KY_OPTO_IN_2	✓	✓	✓	✓
4	OptoCoupled Input 3	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 Output 0					
22	OptoCoupled Output 1					
23	OptoCoupled Output 2					
24	OptoCoupled Output 3					
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_0	✓	✓	✓	
41	Timer1Active	KY_TIMER_ACTIVE_1	✓	✓	✓	
42	Timer2Active	KY_TIMER_ACTIVE_2	✓	✓	✓	
43	Timer3Active	KY_TIMER_ACTIVE_3	✓	✓	✓	
44	Timer4Active	KY_TIMER_ACTIVE_4	✓	✓	✓	
45	Timer5Active	KY_TIMER_ACTIVE_5	✓	✓	✓	
46	Timer6Active	KY_TIMER_ACTIVE_6	✓	✓	✓	
47	Timer7Active	KY_TIMER_ACTIVE_7	✓	✓	✓	
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	✓			

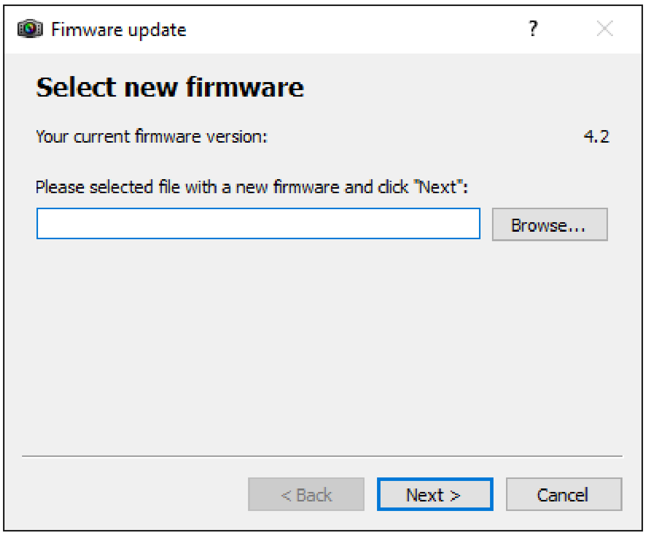
Table 6 – Frame Grabber I/O source (Firmware version 1.xx)

31 Troubleshooting

31.1 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



Firmware update

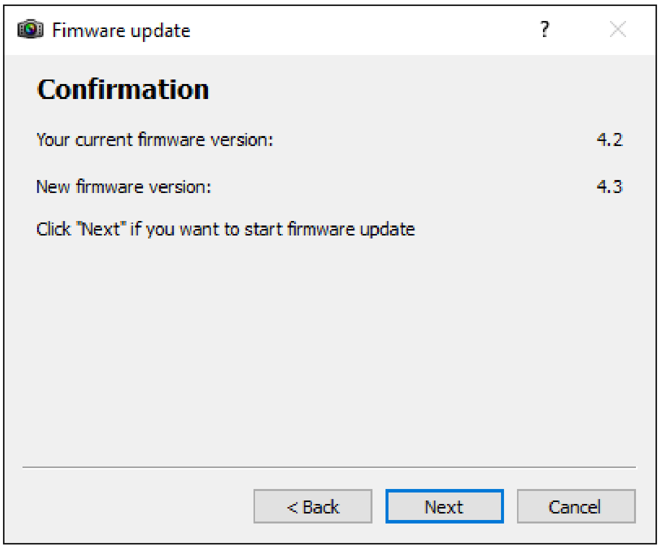
Select new firmware

Your current firmware version: 4.2

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

Figure 12 – Firmware Update selection window

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



Firmware update

Confirmation

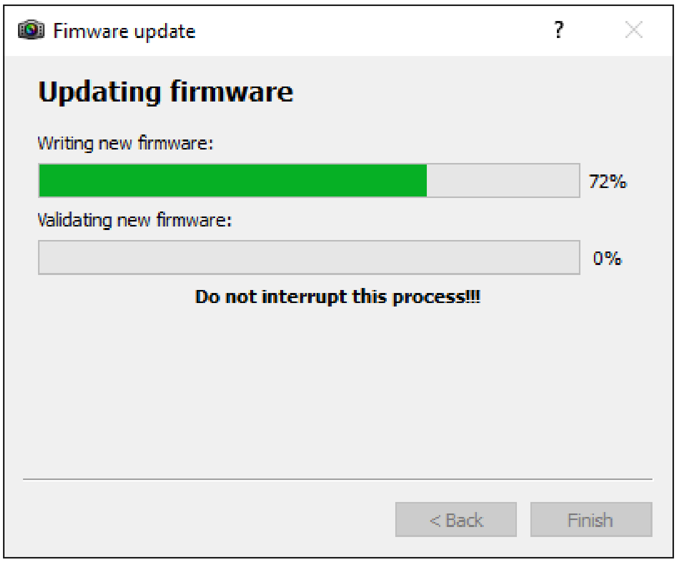
Your current firmware version: 4.2

New firmware version: 4.3

Click "Next" if you want to start firmware update

Figure 13 – Firmware 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 the figure below.
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.


Fimware update

Updating firmware

Writing new firmware: 72%

Validating new firmware: 0%

Do not interrupt this process!!!

Figure 14 – Firmware Update process window

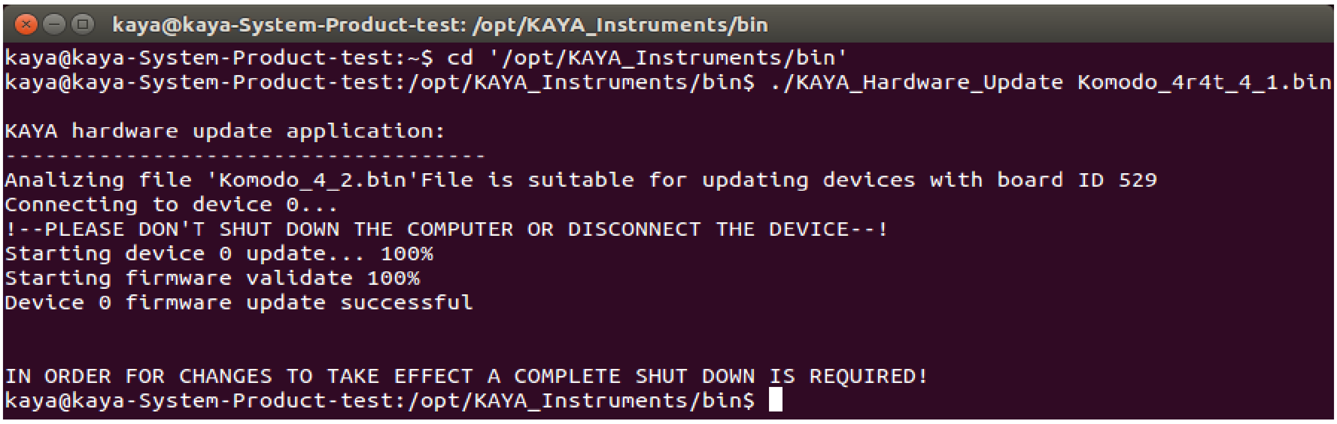
31.2 Updating the device firmware using pre-built utility for Linux

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: This method is currently unsuitable for setups where more than one board with the same product ID is 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 a KAYA Instruments' representative for any question further questions.



```
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 15 – Firmware Update via Terminal process window

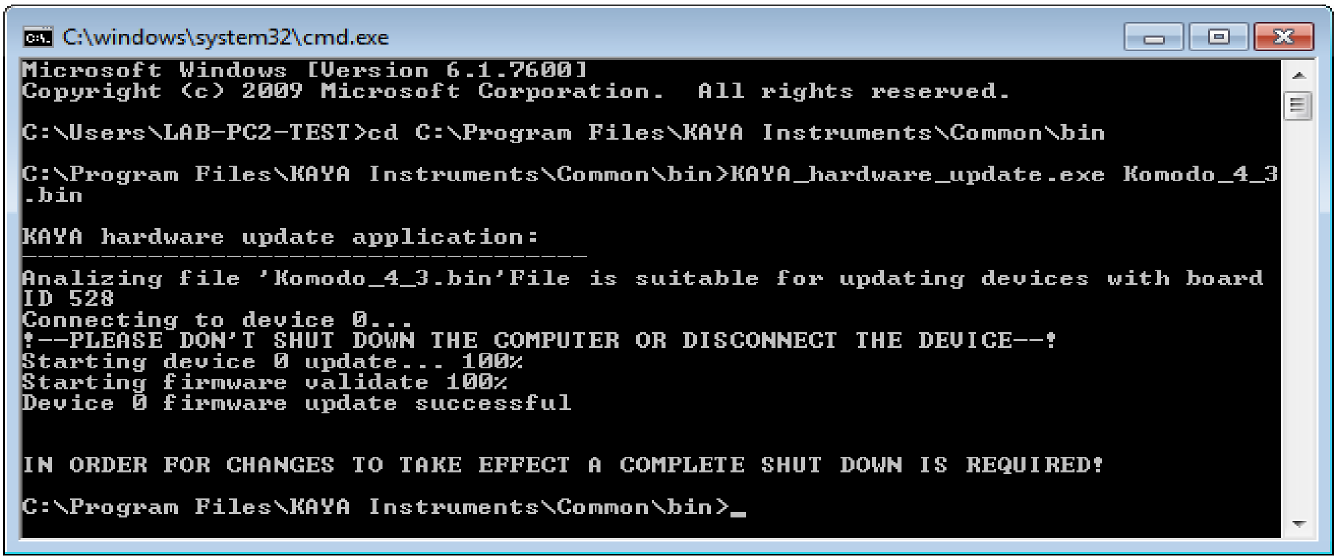
31.3 Updating the device firmware using pre-built utility for Windows

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: This method is currently unsuitable for setups where more than one board with the same product ID is 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 a KAYA Instruments' representative with any further questions.



```
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 16 – Firmware Update via Command line process window

31.4 Collecting log Files

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

31.4.1 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:

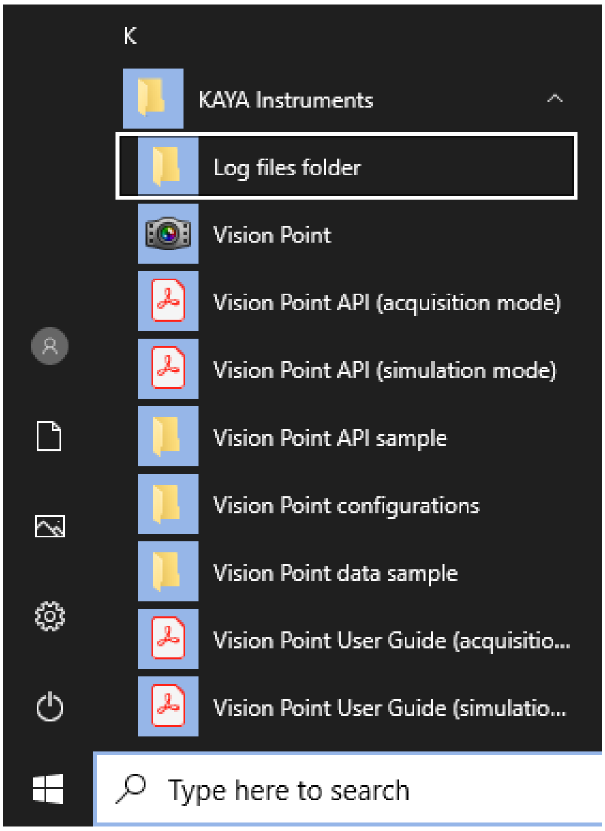

K
KAYA Instruments
Log files folder
Vision Point
Vision Point API (acquisition mode)
Vision Point API (simulation mode)
Vision Point API sample
Vision Point configurations
Vision Point data sample
Vision Point User Guide (acquisitio...
Vision Point User Guide (simulatio...
Type here to search

Figure 17 – Log files folder from the quick start menu path

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

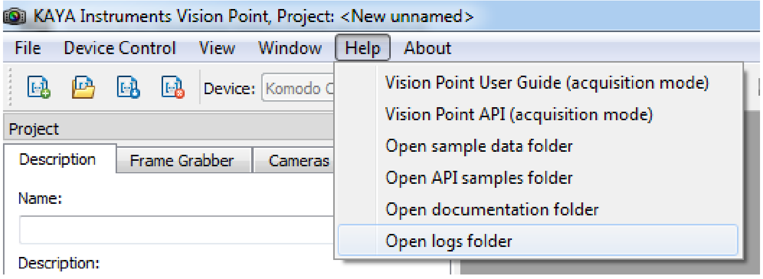


Figure 18 – Log files folder from Vision Point Help menu path

Remarks:

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.

31.4.2 Linux Operating System

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

31.4.3 Logs retaining policy

User may configure the retaining policy of the log files by using the following setting in registry:

- Number of files to keep in 'Archive' when (Default) is 2 - Keep N latest
  KYSettings::FeatureType::UserConf, "LogFilesKeep.Amount"
- Maximum age in seconds of a file to keep in 'Archive' when (Default) is 3 - Keep max age
  KYSettings::InitInteger(KYSettings::FeatureType::UserConf, "LogFilesKeep.MaxAge", 1

31.5 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.
You also can visit our kayainstruments.com Web site, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Send us an e-mail to: support@kayainstruments.com
Create a support request on the web: http://support.kayainstruments.com
Our knowledge base is available at: http://support.kayainstruments.com/kb/index.php

31.6 Submitting a Support Request

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

- Logs form Vision Point where applicable (See section 24.4)
- PC configuration
- Operation System
- Card part number or full name
- Firmware in use
- Software in use