KAYA VISION

Vision Point II API
Data Book

December 2025
Rev 2026.1.0

20283 State Road 7, Suite 350
Boca Raton, FL 33498
USA
+1 (561) 689-2899

info@kaya.vision
support@kaya.vision

www.kaya.vision

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

Revision History

Version	Date	Notes
2024.1.0	09.2024	Initial release
2024.1.1	09.2024	Vision Point API release 2024.1 (service pack 1). The same document without changes
2024.1.2	10.2024	Vision Point API release 2024.1 (service pack 2) - KYParametersHandler_GetParameterAttributeValue() function was added to the KYVP ParametersHandler library. - KYFG_LoadFirmware(), KYFG_GetFirmareUpdateFileInfo() , KY_GetCameraPropertyParameterValue() and KY_GetGrabberPropertyParameterValue() functions were added to the Vision Point API Adapter
2024.1.3	10.2024	The same document without changes
2025.1.0	12.2024	Vision Point API release 2025.1 - KYVPLibExtension library was added. - API Examples for Linux section was added. - Legacy Vision Point API Adapter section has been moved to the separate document titled “Vision Point Migration guide”.
2025.1.1-1.3	01-04.2025	The same document without changes
2025.2.0 - 2025.2.3	06-09.2025	The same document without changes
2025.2.4	11.2025	Added KYFoundation library section Added KY_RESULT section
2026.1.0	12.2025	API Examples section updated

Table 1 – Revision History

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

Table of Contents

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

1 Figures and Tables

1.1 List of Figures

1.2 List of Tables

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

2 Introduction

2.1 Safety precautions

With your KAYA's Frame Grabber 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.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

2.2 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

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

3 Overview

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

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

3.1 Document Structure

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

- KYVPLibTL library implements the transport layer function.
- KYVPParametersHandler library provides access to devices' Gen<i>Cam parameters.
- KYVPLibExtension library enhances the functionality of the KYVPLibTL library.
- KYVPImage Processing library to perform various image processing tasks with acquired stream images.
- KYFoundation library to handling KY_RESULT
- API examples.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

4 API Notes and Limitations

4.1 API Usage in Multi-Threaded Applications

Vision Point II 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:

- KYVPLibTL Library accessed via an instance of KYVP_TL_HANDLE
- PCI Interface accessed via an instance KYVP_PCI_INTERFACE_HANDLE
- Local Device accessed via an instance of KYVP_DEVICE_HANDLE
- Stream accessed via an instance of KYVP_STREAM_HANDLE
- Event accessed via an instance KYVP_EVENT_HANDLE
- Remote Device accessed via an instance of KYVP_REMOTE_DEVICE_HANDLE
- ParametersHandler library accessed via an instance KYVP_COLLECTION_HANDLE
- A frame buffer accessed via an instance of KYVP_BUFFER_HANDLE

4.2 Important notes

1. 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 more than the simplest initialization is required, it is recommended to perform it 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.

2. KYVP ParametersHandler performance:
   Functions of KYVP ParametersHandler are inherently relatively slow because they utilize the Gen<I>Cam 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 KYVPLibTL_DSGetBufferInfo() with a relevant command to retrieve the required information. Those functions can still be used in non-performance-critical parts for example at the system initialization, before or after acquisition sessions etc.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

5 KYVPLibTL library

5.1 Function Call Sequence

KYVPLibTL library is low level library similar to GenTL Producer. KYVPLibTL library implements the transport layer function. This library provides a transport layer interface to acquire images or other data and facilitate communication with a device. Its role is not to configure the device, except for transport-related settings, though it may be used indirectly to transmit configuration information to and from the device.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

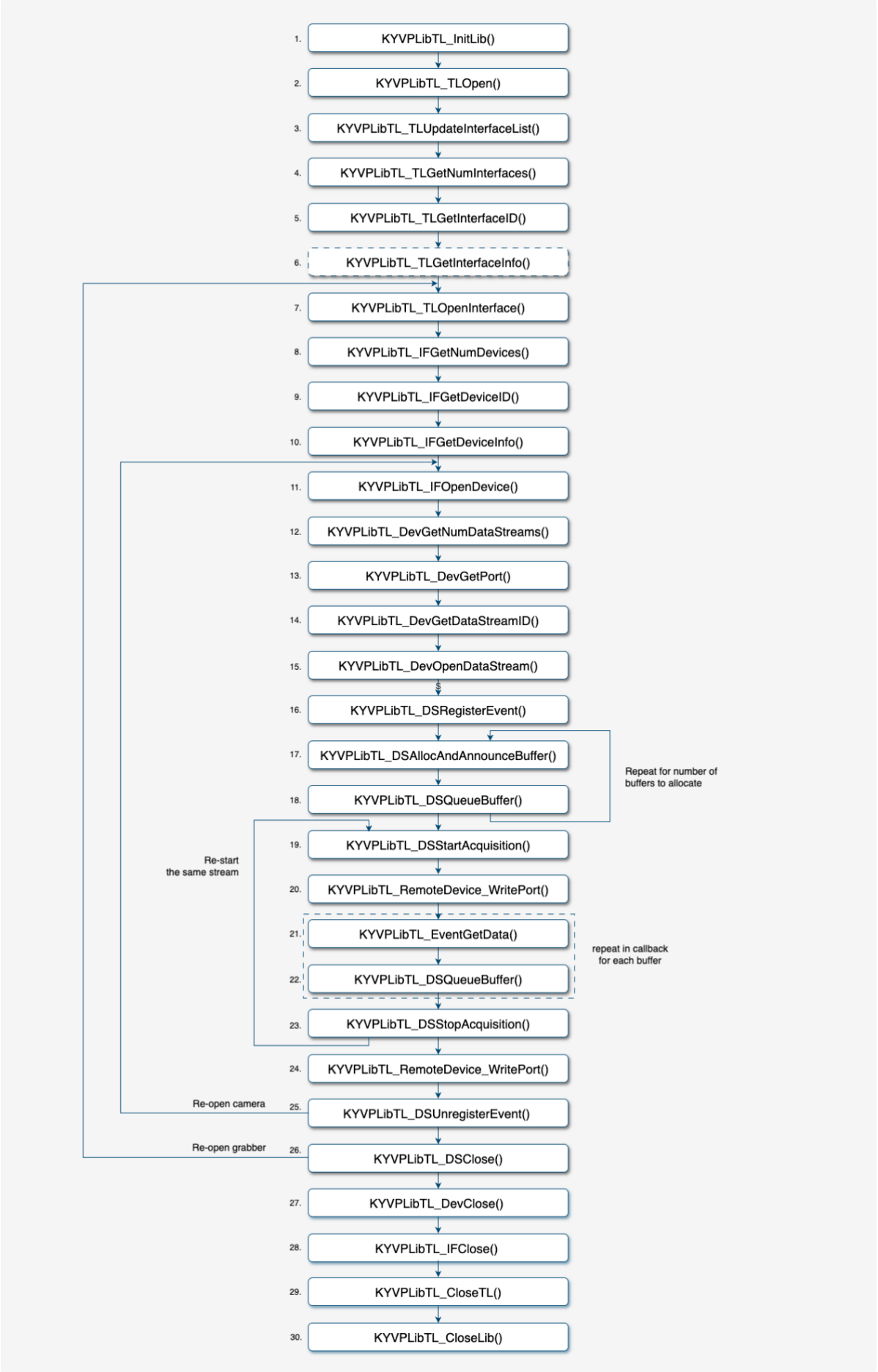

Figure 1 – KYVPLibTL function call sequence

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

1. KYVPLibTL_InitLib() – Initialize the library.
2. KYVPLibTL_TLOpen() – Open the system module.
3. KYVPLibTL_TLUpdateInterfaceList() – Update the internal list of available interfaces.
4. KYVPLibTL_TLGetNumInterfaces() – Get the number of available Devices (Frame Grabbers).
5. KYVPLibTL_TLGetInterfaceID() – Get the unique ID of the Device. To obtain the HANDLE required for operating on the System module's functions, the ID of the Device must be called.
6. KYVPLibTL_TLGetInterfaceInfo() – (Optional) Get information about Device.
7. KYVPLibTL_TLOpenInterface() – Open a control to a selected Device.
8. KYVPLibTL_IFGetNumDevices() – Get the number of available Local Devices on the current Device.
9. KYVPLibTL_IFGetDeviceID() – Get the unique ID of the Local Device. To obtain the HANDLE required for operating on the Local Device functions, the ID of the Device must be called.
10. KYVPLibTL_IFGetDeviceInfo() – (Optional) Get information about Local Device.
11. KYVPLibTL_IFOpenDevice() – Open a connection to the selected Local Device.
12. KYVPLibTL_DevGetNumDataStreams() – Get the number of available streams on this Local device.
13. KYVPLibTL_DevGetPort() – Retrieve the Handle for the associated Remote Device.
14. KYVPLibTL_DevGetDataStreamID() – Queries the unique ID of the data stream.
15. KYVPLibTL_DevOpenDataStream() – Open the given stream on the given Remote Device.
16. KYVPLibTL_DSRegisterEvent() – Register an event for KYVP_EVENT_TYPE_NEW_BUFFER().
17. KYVPLibTL_DSAllocAndAnnounceBuffer() – 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.
18. KYVPLibTL_DSQueueBuffer() – This function queues a specific buffer for acquisition. A buffer can be queued at any time after it has been announced, whether before or after the acquisition process has begun, as long as it isn't already in the queue. The order in which buffers are delivered might differ from the order in which they were queued.
19. KYVPLibTL_DSStartAcquisition() – Start the acquisition for the specified Remote Device.
20. KYVPLibTL_RemoteDevice_WritePort() – Send command to the Remote Device to start acquisition.
21. KYVPLibTL_EventGetData() – Retrieves the next event data entry from the event data queue associated with the KYVP_EVENT_HANDLE.
22. KYVPLibTL_DSQueueBuffer() – This function queues a specific buffer for acquisition during new buffer Event. A buffer can be queued during the Data Stream.
23. KYVPLibTL_DSStopAcquisition() – Stop the acquisition for the specified Remote Device.
24. KYVPLibTL_RemoteDevice_WritePort() – Send command to the Remote Device to stop acquisition.
25. KYVPLibTL_DSUnregisterEvent() – Unregister the given event.
26. KYVPLibTL_DSClose() – Close the Data 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.
27. KYVPLibTL_DevClose() – Close the connection to the chosen Remote Device.
28. KYVPLibTL_IFClose() – Close the connection to the PCI Interface, this will clear all relevant resources.
29. KYVPLibTL_CloseTL() – Shutdown the system module.
30. KYVPLibTL_CloseLib() – Shutdown the library.

5.2 System Modules

The KYVPLibTL standard defines a layered structure for libraries implementing the KYVPLibTL Interface. Each layer is defined in a module. The modules are presented in a tree structure with the System module as its root.

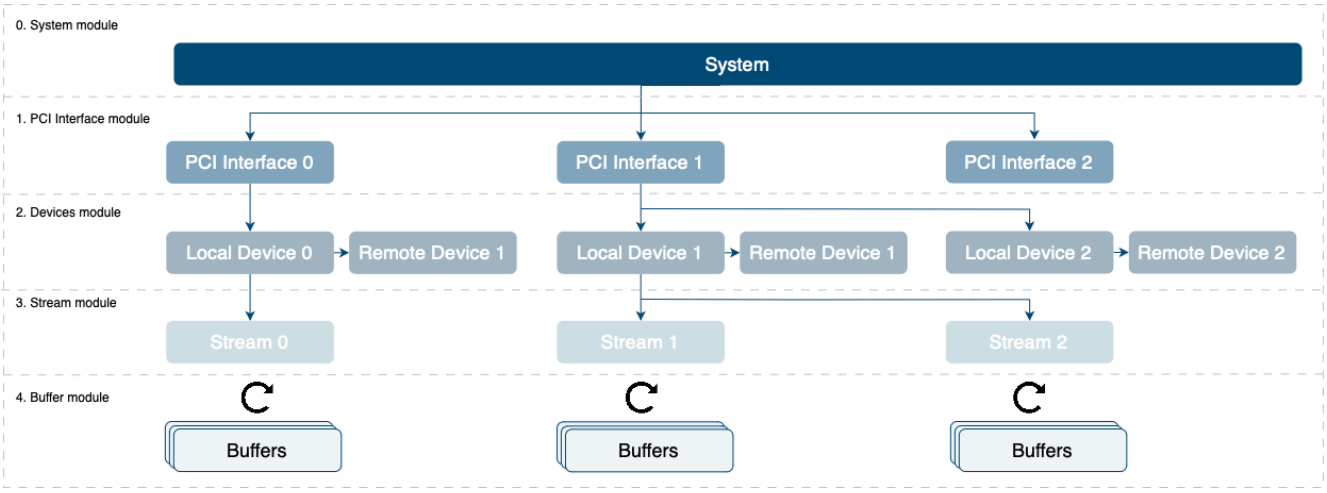

Figure 2 – Module hierarchy

5.2.1 System Module

The System module in KYVPLibTL serves as the entry point for any KYVPLibTL consumer, acting as the root of the hierarchy for accessing a KYVPLibTL Producer software driver. It represents the entire system from the perspective of the KYVPLibTL libraries and is responsible for enumerating and instantiating available interfaces. Additionally, the System module handles signaling and internal configuration.

5.2.2 PCI Interface Module

The PCI Interface module in a KYVPLibTL system represents a single physical interface – a Frame Grabber. Its main function is to enumerate and instantiate available devices on the interface, also signaling and module configuration capabilities to the KYVPLibTL Consumer. Each PCI Interface module supports only one transport layer technology, so multiple technologies require separate PCI Interfaces. The number of devices on an PCI Interface is not limited by the system but depends on the hardware capabilities.

5.2.3 Local Device Module

The Local Device module in a KYVPLibTL system serves as a proxy for one physical remote device on the PCI Interafce side, enabling communication and managing Data Stream modules. Local Device is a certain set of parameters that are actually on the grabber's side, but logically relate to a remote camera. Local Device is a port that can be read and written to. When reading or writing to it, parameters that are actually related to the camera, but are on the Frame Grabber side, change. It also provides signaling and configuration options to the KYVPLibTL Consumer. Each PCI Interface module can support 0, one or multiple Local Device modules, each tied to a single transport layer technology. The number of Local Devices connected to an interface is not limited by the system but depends on the hardware used.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

5.2.4 Remote Device Module

The Remote Device module in a KYVPLibTL system represents one physical Remote Device. Remote Device is a port that can be read and written to. During write or read to the Remote Device, a command directly sent to the camera. Each PCI Interface module can support 0, one or multiple Remote Device modules, each tied to a single transport layer technology. The number of devices connected to an interface is not limited by the system but depends on the hardware used.

5.2.5 Data Stream Module

The Data Stream module represents a single image data stream from a Remote Device, serving as the acquisition engine and managing the internal buffer pool. It also offers signaling and configuration options to the KYVPLibTL Consumer. A device can support zero, one or multiple data streams, it limited only by the hardware and implementation.

5.2.6 Buffer Module

The Buffer module represents a single memory buffer intended for acquisition. This buffer can be allocated either by the user or by the KYVPLibTL Producer, such as pre-allocated system memory. The module also offers signaling and configuration options to the KYVPLibTL Consumer. To enable data streaming, at least one buffer must be announced to the Data Stream module and placed in the input buffer pool. The KYVPLibTL Producer may preprocess the image data which changes image format and/or buffer size.

5.3 Module Enumeration and Instantiation

The behavior described below is observed from the perspective of a single process. A KYVPLibTL Producer implementation must ensure that each process accessing the resources has an independent view of the hardware, without needing to be aware that other processes are also involved.

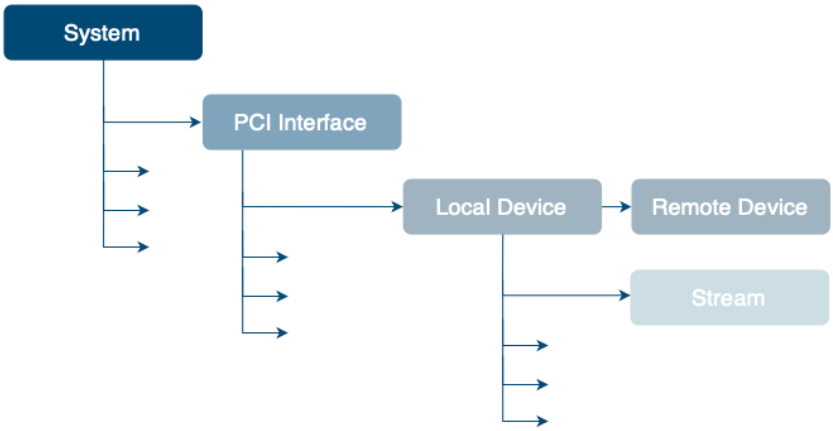

Figure 3 – Enumeration hierarchy

Before the System module can be opened or any operation performed on the KYVPLibTL Producer driver, the KYVPLibTL_InitLib() must be called (once per process). After closing the System module (e.g., when the KYVPLibTL Consumer is closed), the KYVPLibTL_CloseLib() function should be invoked to free all resources. If KYVPLibTL_InitLib() is called again without an accompanying KYVPLibTL_CloseLib(), the second call will result in an error. Likewise, multiple calls to KYVPLibTL_CloseLib() without reinitializing with KYVPLibTL_InitLib() will also free resources improperly.

5.3.1 System

The System module serves as the entry point for a KYVPLibTL Consumer to interact with a KYVPLibTL Producer. It enables the enumeration of hardware interfaces and requires a KYVP_TL_HANDLE to perform operations. The module manages communication between processes and ensures that system resources are properly allocated and freed. Interface modules can be enumerated and accessed via unique IDs, and updates to the interface list should be managed carefully. Proper closure of the System module ensures resources are released, and the module can be re-opened if needed.

5.3.2 Interface

The Interface module in KYVPLibTL represents a PCI Interface (Frame Grabber). It allows enumeration of attached devices, with each interface identified by a unique ID. The module list can be updated using KYVPLibTL_IFUpdateDeviceList(), and interfaces can be closed in any order. Devices on an interface can be enumerated without opening them, and devices can be opened directly using their unique IDs. The PCI Interface and device lists are not thread-safe and must be accessed carefully to avoid conflicts.
After closing an PCI Interface module, it can be reopened again and the handle to the module may be different from the first instantiation.

5.3.3 Local and Remote Devices

A Device module represents the KYVPLibTL Producer driver's view on a remote device. It handles the enumeration of available Data Streams, which is limited by the device and the KYVPLibTL implementation. Devices are identified by unique IDs, and the module can manage multiple Data Streams. Only one Local Device can exist for the same Remote Device within the same process. The module does not track references within a process, so closing it frees all related resources.
After closing a Device module, it can be reopened again and the handle to the module may be different from the first instantiation.

5.3.4 Data Stream

The Data Stream module is primarily focused on acquisition and does not enumerate its child modules. Each stream is identified by a unique ID within the Device module, interpreted solely by the KYVPLibTL Producer. When no longer needed, the KYVPLibTL_DSClose() function must be called to free resources, stop acquisitions, flush buffers, and revoke them. The module does not support access from different processes and lacks reference counting.
After closing a Data Stream module, it can be reopened again and the handle to the module may be different from the first instantiation.

Stream interface functions are used to handle received data. 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 becomes available for the user application via KYVPLibTL_EventGetData(). This frame memory will not be affected until it is returned to *Input Queue* using KYVPLibTL_DSQueueBuffer() function call. The user application is responsible for putting frames to *Input Queue* for each frame supplied to the host application through Stream event. 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 KYVPLibTL_OpenDataStream() function.

5.3.5 Buffer

Each Buffer is uniquely identified by a handle obtained from either the KYVPLibTL_DSAnnounceBuffer() or KYVPLibTL_DSAllocAndAnnounceBuffer() functions. Buffers can be allocated by the KYVPLibTL Consumer or Producer and must be announced to the Data Stream module for use. The required Buffer size should be requested from the Data Stream via function KYVPLibTL_DSGetBufferInfo(). To enable the acquisition engine to stream data into a Buffer, the Buffer must first be added to the Input Buffer Pool by calling the KYVPLibTL_DSQueueBuffer() function using the KYVP_BUFFER_HANDLE obtained from the announcement functions.
The KYVP_BUFFER_HANDLE can be released by calling the KYVPLibTL_DSRevokeBuffer() function. However, if the buffer is still in the Input Buffer Queue or the Output Buffer Queue of the acquisition engine, it cannot be revoked, and an error will be returned if attempted. A memory buffer should only be announced once per stream.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

6 KYVPLibExtension library

KYVPLibExtension is a low-level extension library designed to enhance the functionality of the KYVPLibTL library. It provides additional functionalities not available via TL Library. For example, firmware update, direct access to hardware registers etc. For more details see reference.

7 KYVPParametersHandler library

7.1 Function Call Sequence

This KYVPParametersHandler is a high level library that loads Gen<i>Cam XML and provides software interface for reading and writing values by a named parameter, and also queuering various attributes of parameter such as min, max, description etc (similar to what Gen<i>Cam reference library implementation provides). It translates high level API calls with parameter names into low level transport requests with register addresses. It does not implement transport functionality and should be used in conjunction with a separate transport implementation.

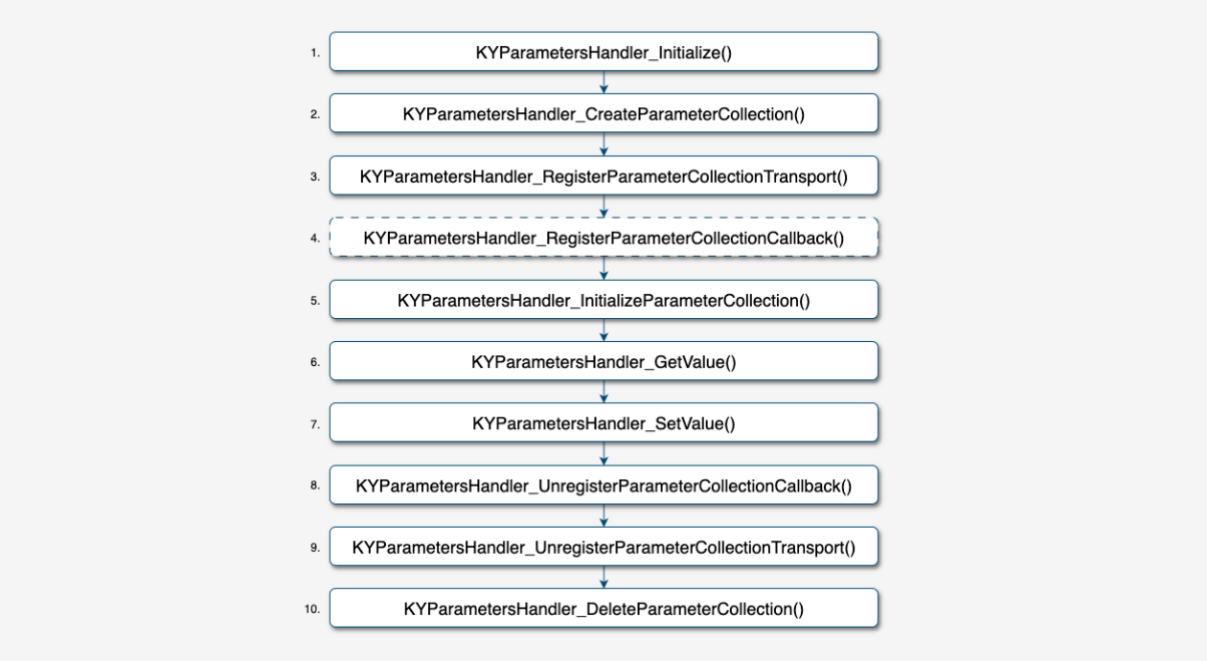

Figure 4 – KYVPParameterHandler function call sequence

1. KYParametersHandler_Initialize() – Handle the allocation of necessary resources and prepare the library for opening.
2. KYParametersHandler_CreateParameterCollection() – This function initializes and returns a handle to a new instance of the library. The handle can then be used for subsequent operations provided by the library. The function ensures that all necessary resources are allocated, and that the library is ready for use.
3. KYParametersHandler_RegisterParameterCollectionTransport() – Register and configure a get and set function collection of parameters within a system.
4. KYParametersHandler_RegisterParameterCollectionCallback() – (Optional) The function called after setting new parameter.
5. KYParametersHandler_InitializeParameterCollection() – The function is generally used to set up or initialize a collection of parameters within a system or application. This function is typically employed to prepare a set of parameters for use, ensuring that they are properly configured and ready for subsequent operations.
6. KYParametersHandler_GetValue() – Retrieve the value associated with a specific key, parameter, or identifier.
7. KYParametersHandler_SetValue() – Assign or update the value associated with a specific key, parameter, or identifier.
8. KYParametersHandler_UnregisterParameterCollectionCallback() – Unregister the function called after setting new parameter.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

9. KYParametersHandler_UnregisterParameterCollectionTransport() – Unregister a get or set function collection of parameters within a system.
10. KYParametersHandler_DeleteParameterCollection() – Delete a handle to a new instance of the library.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

8 KYVPImageProcessing library

The KYVPImageProcessing is a library designed to perform some image processing tasks within application. This library offers robust functionality for converting video formats, saving various types of data, including raw data, etc.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

9 KYFoundation library

The KYFoundation library includes tools and utilities for efficient handling of KY_RESULT, promoting consistent status management and efficient error processing across the system.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

10 KY_RESULT

All Vision Point II SDK functions return a KY_RESULT, which is a generic status code indicating the outcome of the function execution.
Some error statuses are general, while others point to a specific condition or failure.

To determine whether a function call succeeded or failed, a user might use macros from the KYFoundation library:

- KY_RESULT_SUCCEEDED(result)
- KY_RESULT_FAILED(result)

In certain cases, it may be useful to determine the exact severity of a KY_RESULT. Use the GetSeverity() function to retrieve the corresponding severity level (such as error, warning, or information).

For a descriptive error code call the KYFoundation_GetErrorCode() function.
Generic error codes are described in the public header KYFoundation_Errors.h.

To obtain an extended description of the result, use:
const char* KYFoundation_What(KY_RESULT result);
This returns a string with additional information about the status.

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

11 Library Exiting

During its operation, our KYVPLibTL library and KYVPParametersHandler library allocate certain resources that must be freed before the library is unloaded from the process:

11.1 Memory buffers

When KYVPLibTL_DSAllocAndAnnounceBuffer() used and memory buffers are allocated by our library, they are marked as such. These buffers are released by our library when the user calls KYVPLibTL_DSClose().

11.2 Background monitoring thread

There is a background monitoring thread started by our library when an application calls KYVPLibTL_TLOpen(). This thread is stopped when KYVPLibTL_TLClose() is called. Unloading the library without closing all grabber handles may lead to undefined behaviour (uncaught exceptions, etc.). The application must call KYVPLibTL_TLClose() before unloading the library.

NOTE about DllMain's DLL_PROCESS_DETACH in Windows

When our library is unloaded from the process it tries to release allocated resources, e.g., stop threads, unpin and release memory, etc. But there are significant limitations on what can be done at this stage – 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". Therefore, it is still responsibility of the user application to properly close the library and all used resourses before the DLL is unloaded.

12 API Examples

The API samples provide practical examples to help developers efficiently integrate and utilize the API. These samples demonstrate common use cases, offer code snippets, and provide guidance on implementing various features, ensuring developers can quickly get started and optimize their applications with the Vision Point II.

Vision Point II API provides next examples:

Vision Point II API example	Description
KYFGLib_Adapter_ManualDetection_Example	Example of using Adapter to run Vision Point legacy code under Vision Point II API using Manual Detection feature.
KYFGLib_Adapter_Example	Example of using Adapter to run Vision Point legacy code under Vision Point II API with Queued Buffers approach.
KYFGLib_Example	Example of using Adapter to run Vision Point legacy code under Vision Point II API with Cyclic Buffers approach.
KYFGLibA.NET	Example of using Adapter to run Vision Point legacy code under Vision Point II API with Cyclic Buffers approach.
KYFGLibA.NET_Example_QueuedBuffers	Example of using Adapter to run Vision Point legacy code under Vision Point II API with Queued Buffers approach.
KYVP_FirmwareUpdate_Example	Example demonstrating KAYA PCIe device firmware update procedure.
KYVP_ManualDetection_Example	Example demonstrating the correct procedure for Define device manually.
KYVP_ParametersHandle_Example	Example shows how to handle cameras parameters using KYVP ParametersHandler library.
KYVP_QueuedBuffers_Example	Example for Queued Buffers image acquisition using KYVPLibTL and KYVP ParametersHandler library. Best API sample for acquitance with VP II API.
KYVPLibTL_Example	Example demonstrating transport layer operations. Requires camera registers to be specified.
SerialPort_Examples	Example of Serial Port API implementation. Refer to KYFGLibA_SerialPort_Example.cpp for full description.

Table 2 – Vision Point II API Examples

12.1 Building API example for Windows

1. Open an example project for Microsoft Visual Studio, provided in the download directory. The “API Samples” directory can be easily found using the shortcut on the desktop or Windows quick search.

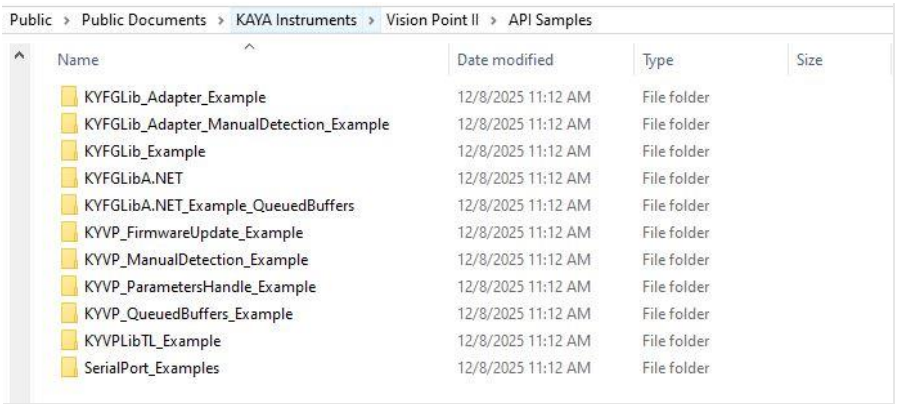

Figure 5 – API Samples folder

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

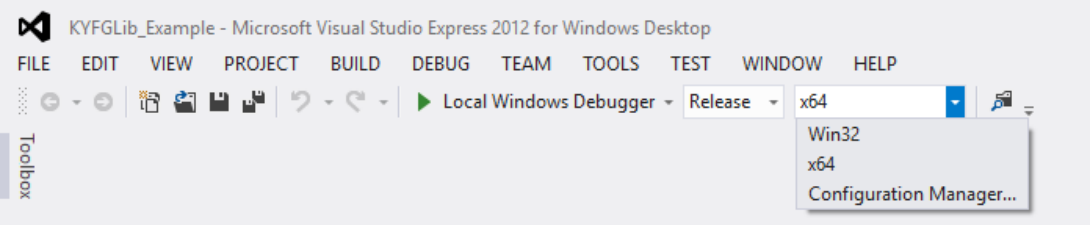

Figure 6 – Visual Studio choosing a solution platform

3. Build a solution.
4. Run the application.

An example is shown in the image below:

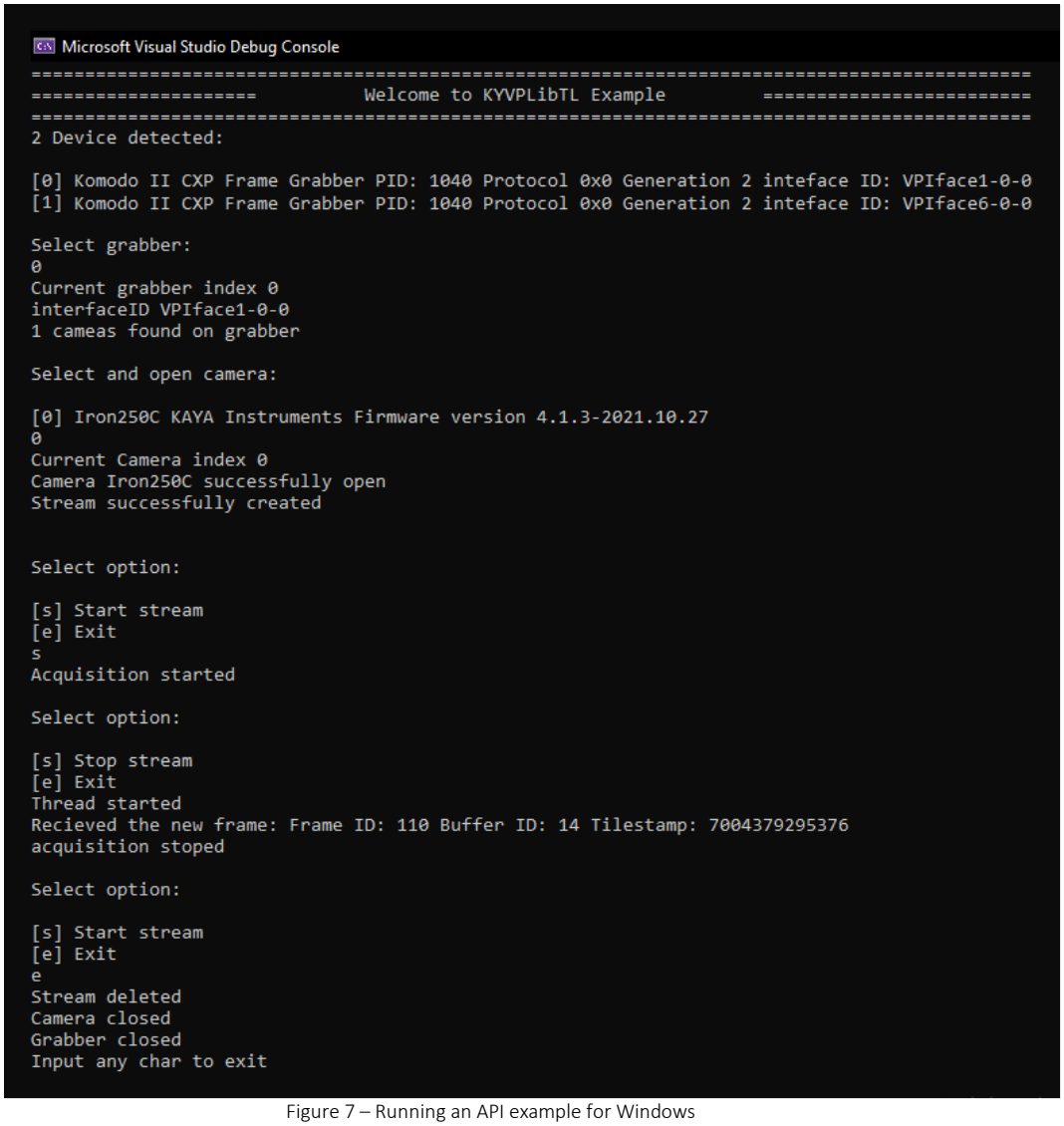

Figure 7 – Running an API example for Windows

+1 (561) 689-2899
info@kaya.vision
www.kaya.vision

12.2 Building API example for Linux

1. Open the Terminal and navigate to directory stored in environment variable $KAYA_VISION_POINT_2_SAMPLE_API.
2. Change directory to KYVPLibTL_Example.
3. Type “make” and make sure the KYVPLibTL_Example executable file was created, in the same directory.
4. To run the API Example, simply type “./KYVPLibTL_Example” followed by “Enter”.

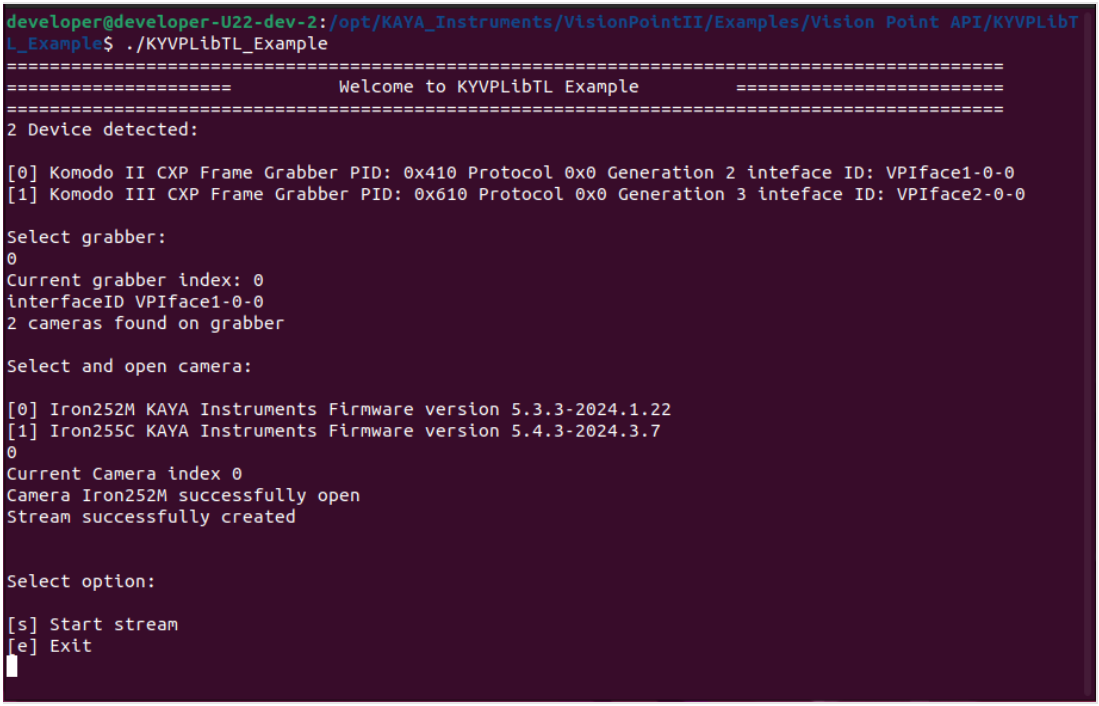

Figure 8 – Running an API example for Linux

13 Collect Diagnostic Info

13.1 Windows Operating System

The Collect Diagnostic Info menu option initializes KYInfo script, which gathers all required system information, including Log Files, and generates an archive named “KAYA”.
This archive can be sent to support to help diagnose and resolve customer issues efficiently.

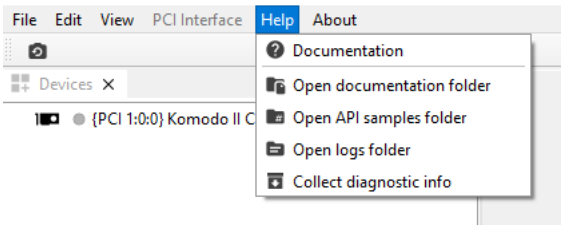

Figure 9 – Collect diagostic info from Vision Point II Help menu

To collect Diagnostic info use the KYInfo.bat file from folder that located in {KAYA Instruments installation folder}\Common\bin\debug tools[1].
It will collect full system information and prepare zip archive[2].

Remarks:

1. By default, installation folder located at C:\Program Files\KAYA Instruments.
2. Diagnostic information archive KAYA.zip location: C:\ProgramData\KAYA Instruments.
3. Installation log files folder can be found: C:\Program Files\KAYA Instruments\Log\Installer.

13.1 Linux Operating System

To find logs files, go to the folder that located in /var/log/KAYA_Instruments/.

REFERENCES

Supported vision standards:

CoaXPress GEN<i>CAM

Vision Point documentation:

Vision Point

TECHNICAL SUPPORT AND PROFESSIONAL SERVICE

If you searched the documents and could not find the answers you need, contact KAYA Vision support service:

- Create a support request on the web: support.kaya.vision
- Our knowledge base is available on: kb.kaya.vision

Visit us at www.kaya.vision for comprehensive information.

SUBMITTING A SUPPORT REQUEST

When opening a support request, please provide the following information when applicable:

For Frame Grabbers:	For Camera:	For Range Extender:
• Vision Point Diagnostic Info* • Serial number of Frame Grabber • Camera model • SFP+ module model • CoaXPress/Fiber cable model and length • External power or PoCXP • PC motherboard model *In the Vision Point app, use menu option Help > Collect diagnostic info.	• Vision Point Diagnostic Info (or frame grabber being utilized) • Serial Number of Camera • XML dump and/or description of how the camera is being utilized • Description of issue • SFP+ module model • CoaXPress/Fiber cable model and length • External power or PoCXP	• Range Extender Model • Serial Number of Range Extender • SFP+ module model • CoaXPress/Fiber Cable model and length • PC configuration • Operating System • Software Revision • Camera and Frame Grabber Manufacturer and Model

KAYA VISION

Have questions about pricing, availability, documentation, or custom options?
We're always ready to assist and provide expert guidance.
Sales Inquiries: info@kaya.vision
Technical Support: support@kaya.vision
www.kaya.vision

KAYA Vision, Inc.
20283 State Road 7
Suite 350
Boca Raton, FL 33498
USA
+1 561 698-2899