Vision Point

Simulator API

Version 2019.1

Data Book

May 2019

International Distributors

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

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

Contents

1 Figures and Tables 3
2 Revision History 4
3 Introduction 5
3.1 Safety Precautions 5
3.2 Disclaimer 6
4 Vision Point API Overview 7
4.1 Overview 7
4.2 Document structure 7
4.3 Function call sequence 8
5 Connection and Info 10
5.1 KYFG_Scan() 10
5.2 KYFG_OpenEx() 11
5.3 KYFG_Close() 12
5.4 KY_DeviceDisplayName()(DEPRECATED) 12
5.5 KY_DeviceInfo() 13
6 Camera Configurations 14
6.1 KYFG_CameraScan() 14
6.2 KYFG_CameraOpen2() 15
6.3 KYFG_CameraClose() 15
6.4 KYFG_SetCameraConfigurationParameterCallback() (C++ only) 16
6.5 KYFG_GetCameraConfigurationParameterDefinitions() (C++ only) 16
7 Callback Functions 17
7.1 KYFG_CameraCallbackRegister() 17
7.2 KYFG_CameraCallbackUnregister() 17
7.3 KYDeviceEventCallBackRegister () 18
7.4 KYDeviceEventCallBackUnregister () 18
7.5 KYFG_SetCameraValue() 19
7.5.1 KYFG_SetCameraValueInt() 19
7.5.2 KYFG_SetCameraValueFloat() 20
7.5.3 KYFG_SetCameraValueBool() 20
7.5.4 KYFG_SetCameraValueEnum() 21
7.5.5 KYFG_ExecuteCommand() 22
7.5.6 KYFG_SetCameraValueString() 22
7.5.7 KYFG_SetCameraValueEnum_ByValueName() 23
7.6 KYFG_GetCameraValueType() 23
7.7 KYFG_GetCameraValue() 24
7.7.1 KYFG_GetCameraValueInt() 24
7.7.2 KYFG_GetCameraValueEnum() 25
7.7.3 KYFG_GetCameraValueFloat() 26
7.7.4 KYFG_GetCameraValueBool() 26
7.7.5 KYFG_GetCameraValueString () 27
7.7.6 KYFG_GetCameraValueString() 28
8 Stream Interface 30
8.1 KYFG_StreamCreateAndAlloc() 30
8.2 KYFG_StreamGetSize() 31
8.3 KYFG_StreamGetFrameIndex() 31
8.4 KYFG_StreamGetPtr() 31

8.5 KYFG_StreamDelete() 32
9 Data Loading 33
9.1 KYFG_LoadPatternData() 33
9.2 KYFG_LoadFileData() 34
10 Video stream generation 35
10.1 KYFG_CameraStart () 35
10.2 KYFG_CameraStop () 35
11 Firmware update 36
11.1 KYFG_CheckUpdateFile 36
11.2 KYFG_LoadFirmware 37
12 API Defines and Macros 38
12.1 API handles 38
12.2 KYBOOL 38
12.3 CameraCallback 38
12.4 KYDeviceEventCallBack 38
12.5 ParameterCallback 39
12.6 UPDATE_CALLBACK 39
13 Enumerations 40
13.1 FGSTATUS 40
13.2 CSSTATUS 41
13.3 PATTERN_TYPE 43
13.4 KYDEVICE_EVENT_ID 43
13.5 KY_CAM_PROPERTY_TYPE 43
13.6 VIDEO_DATA_WIDTH 44
13.7 VIDEO_DATA_TYPE 44
13.8 VIDEO_DATA_SUBTYPE 45
14 Structures 47
14.1 KYDEVICE_EVENT 47
14.1.1 KYDEVICE_EVENT_CAMERA_START 47
14.2 VIDEO_PIXELIF 47
14.3 UPDATE_STATUS 48
15 Building an API example 49
15.1 API example for Windows 49
15.2 API example for Linux 51
16 Troubleshooting 53
16.1 Updating the device firmware using Vision Point Application 53
16.2 Updating the device firmware using pre-built utility for Linux 55
16.3 Updating the device firmware using pre-built utility for Windows 56
16.4 Collecting log files 57
16.5 Technical Support and Professional Services 59
16.6 Submitting a support request 59

1 Figures and Tables

Index of Figures

FIGURE 1 : FUNCTION CALL SEQUENCE 8
FIGURE 2 : FIRMWARE UPDATE SELECTION WINDOW 53
FIGURE 3 : FIRMARE UPDATE CONFIRMATION WINDOW 54
FIGURE 4 : FIRMARE UPDATE PROCESS WINDOW 55
FIGURE 5 : FIRMARE UPDATE VIA TERMINAL PROCESS WINDOW 56
FIGURE 6 : FIRMARE UPDATE VIA COMMAND LINE PROCESS WINDOW 57

Index of Tables

TABLE 1 : REVISION HISTORY 4

2 Revision History

Version	Date	Notes
4.1	07/2017	Visio Point API release 4.1 • Initial release
4.2	08/2017	Visio Point API release 4.2 • Troubleshoot and building an API example sections were added
4.3	04/2018	Visio Point API release 4.3 • New Python API • New .NET API • Minor function corrections
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 • Review and minor corrections
5.0	03/2019	Visio Point API release 2019.1 • Windows service “KYService” and display name “KAYA Instruments Service” installation. • Automatic monitoring and management of PoCXP for CoaXPress cameras. Note: The software stack requires “KYService” to be running, otherwise KYFG_Scan() will return 0 and KYFG_Open()/KYFG_OpenEx() will return INVALID_FGHANDLE. • KYFGLib_Initialize() - optional call before “KYFGScan” and reserved for future usage. • Genicam and OpenCV libraries are not installed to Vision Point's "bin" folder which is added to system's PATH. Instead, will be installed into a sub-folder for internal use only. If these libraries are needed by user's application, it should be installed separately. • Visual Studio 2017 flavor support. User will be able to use our libraries linked to Visual Studio 2017 flavor on run-time: KYFGLib_vc141.dll, clserkyi_vc141.dll • Minor function corrections
5.0.1	05/2019	Visio Point API release 2019.1/API 5.0.1 • Review and minor documentation corrections

Table 1 : Revision History

3.1 Safety Precautions

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

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

Caution

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

Do not use or place the product in the following locations. • Humid and dusty locations • Airless locations such as closet or bookshelf • Locations which receive oily smoke or steam • Locations close to heating equipment • Closed inside of a car where the temperature becomes high • Static electricity replete locations • Locations close to water or chemicals Otherwise, a fire, electric shock, accident or deformation may occur due to a short circuit or heat generation.
Do not place heavy things on the product. Otherwise, the product may be damaged.

3.2 Disclaimer

This product should be used for CoaXPress video simulation. It also can be used for digital input/output (GPIO) simulation purposes. KAYA Instruments assumes no responsibility for any damages resulting from the use of this product for purposes other than those stated.

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

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

KAYA Instruments assumes no responsibility or liability for:

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

4 Vision Point API Overview

4.1 Overview

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

This API is to be used with KAYA's Camera Simulators hardware provided by KAYA Instruments. This is a high level API for interactive and highly configurable camera simulation interface for the Chameleon camera simulators. This simulator is capable of generating CoaXPress video streams and test patterns of up to 4 CoaXPress links at various speeds and topologies. Each link supports standard CoaXPress bitrates up to 6.25 Gbps.

4.2 Document structure

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

- Connection and Info
  - ✓ Connect/disconnect to a specific Simulator
- Camera Configurations
  - ✓ Simulating and configuring different cameras
  - ✓ Use camera Simulator built-in XML or override with another camera XML file
- Callback Functions
  - ✓ Callback functions for data generation
- Camera values
  - ✓ Use XML fields to configure camera 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 generation
  - ✓ Generation of data stream
- Low level bootstrap access
  - ✓ Write/read to camera bootstrap space directly with no enforcement

- Defines, Macros, Structures and Enumerations
  - ✓ All available parameters types and definitions that can be also used in the host application These can be found under "<installation folder>/Vision Point/include".
- Configuration parameters
  - ✓ KAYA additional Gen<i>Cam configuration parameters for controlling, analyzing and configuring the system.

4.3 Function call sequence

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

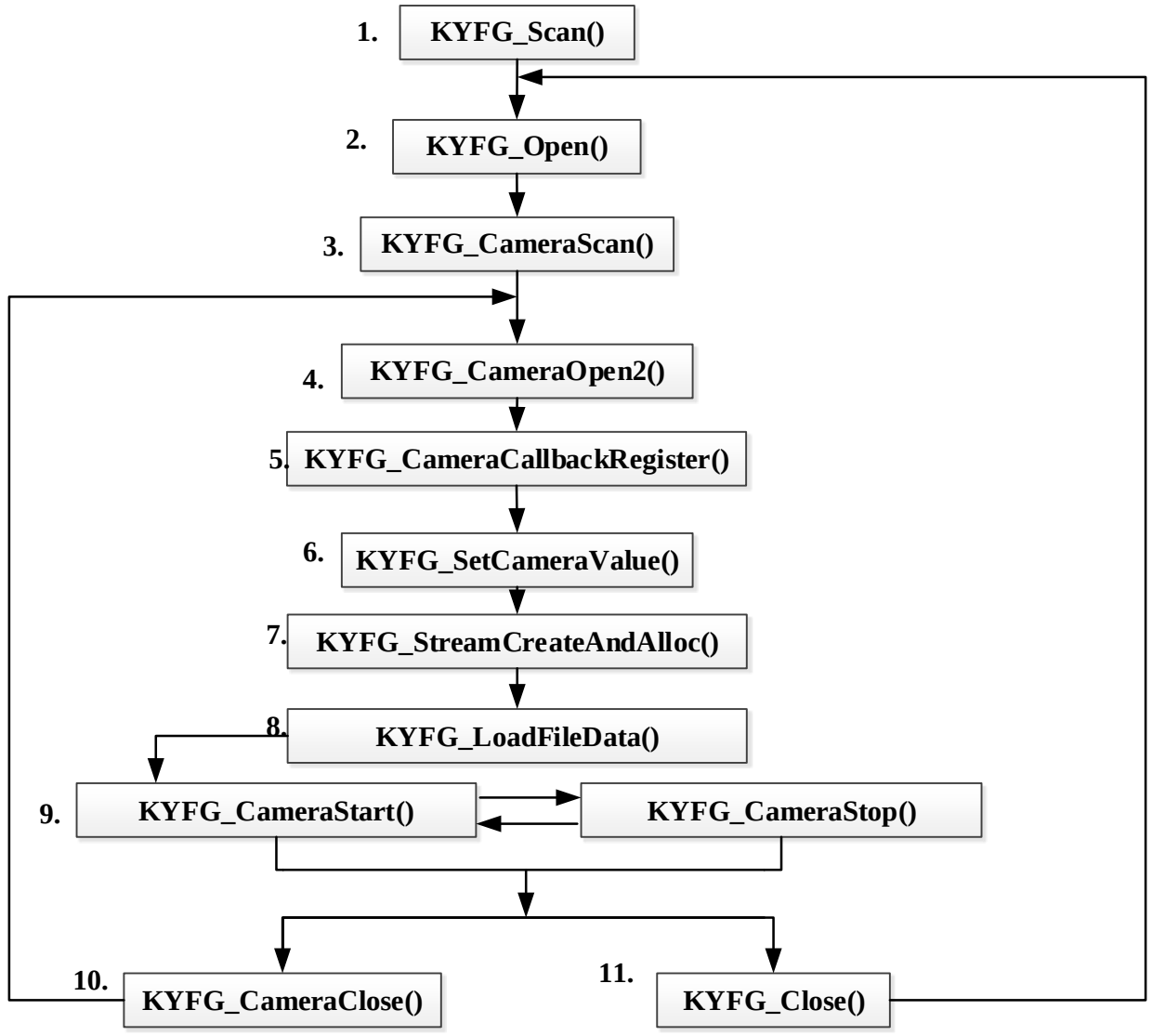


Figure 1 : Function call sequence

1. Scan for PCI devices currently connected to the PC. This will return an array of found hardware and Virtual device PID's.
2. Find index of device with Product ID equal CHAMELEON_DEVICE_ID.
3. Open a connection to a selected PCI device Use the index corresponding to a PCI device from the array acquired in step 1.
4. Detect implemented cameras. There is no restriction on connection topology or CoaXPress camera speed.
5. A callback is an optional function which can be registered after camera was found in the scan process. In order to work properly, the callback should be registered before actually starting the stream generation. This callback will be called upon each frame generation on a specific camera. With the callback function, a handle to the relevant buffer will arrive. Use the Stream Interface functions to retrieve currently generation frame. KYFG_CameraOpen2() and KYFG_CameraClose() doesn't invalidate the callback registration.
6. Open a connection to selected camera and load native or external XML file.
7. Set the desired values to determine the camera parameters using the different available methods.
8. Allocate the memory required for generation of video stream. Several frames should be allocated in order not to immediately run over previously received data.
9. Start/Stop the generation of a video stream.
10. If you no longer want to continue using the camera, close the connection to chosen camera.
    To connect back to the camera and only if this camera wasn't physically disconnected, no camera scan is needed, and KYFG_CameraOpen2() can be called.

5.1 KYFG_Scan()

Scans for KAYA's PCI devices currently connected to the PC PCIe slots.

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

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

Return value

Returns the number of connected hardware and virtual KAYA"s PCI devices.

If pids_info is not NULL pointed array is filled with each device's Product ID (pid). Only devices with Product ID equal CHAMELEON_DEVICE_ID are KAYA Camera Simulators.

Remarks

If this function is called with 'pids_info' parameter set to NULL, pids_info array will not be filled and the function will only return number of connected KAYA's PCI devices.

Example code

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

        infosize = KYFG_Scan(NULL, 0); // Scans for KAYA's PCI devices currently connected
                                       // to PC. Returns number of detected devices
        info = (unsigned int *) malloc ( sizeof(unsigned int) * infosize );
        infosize = KYFG_Scan(info,infosize); // Fills array *'info' with Product IDs of each found device
        for (i = 0; i < infosize; i++)
        {
            if(info[i] == CHAMELEON_DEVICE_ID)
            {
```

```
            printf("device N %d is Chameleon simulator");
        }
    }
```

5.2 KYFG_OpenEx()

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

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

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

Return value

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

Remarks

1. A project file with previously saved values can be passed in order to initialize camera parameters. For additional information regarding project file please refer to Vision Point application user guide: "Vision_Point_App_User_Guide_For_Simulation_Mode".

5.3 KYFG_Close()

Close PCI device specified by its handle. Stops any running camera simulation and deletes previously created buffers associated with this device and its simulated camera.

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen PCI device

Return value

FGSTATUS - Status and error report.

5.4 KY_DeviceDisplayName()(DEPRECATED)

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

Retrieve device name for the specified index.

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

Parameter name	Type	Description
index	int	Discovered device index

Return value

The name of PCI device at the specified index.

5.5 KY_DeviceInfo()

Fills structure with info about the relevant device.

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

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

Return value

FGSTATUS - Status and error report.

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

6.1 KYFG_CameraScan()

This function is used to retrieve number of cameras implemented by given Chameleon Simulator and fill array with their API handles.

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

Parameter name	Type	Description
handle	FGHANDLE	API handle to chosen PCI device
camHandleArray	CAMHANDLE*	Array of API camera handles of implemented cameras
detectedCameras	int*	Number of implemented cameras

Return value

FGSTATUS - Status and error report.

Example code

```
        CAMHANDLE CamHandleArray[MAX_CAMERAS] = {0};  // maximum MAX_CAMERAS cameras can be implemented
        int detectedCamerasNum = 0;
        …
        KYFG_CameraScan(deviceHandle ,CamHandleArray, &detectedCamerasNum);
        printf("This Simulator implements %d cameras", detectedCamerasNum);
        …
```

6.2 KYFG_CameraOpen2()

Opens a connection to chosen camera, loads built-in XML file or uses external XML file provided to override the native one.

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

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

Return value

FGSTATUS - Status and error report.

Remarks

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

6.3 KYFG_CameraClose()

Close a connection to the selected camera. Stops data generation and deletes previously created buffers associated with the camera.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera

Return value

FGSTATUS - Status and error report.

6.4 KYFG_SetCameraConfigurationParameterCallback() (C++ only)

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

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

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

Return value

FGSTATUS - Status and error report.

6.5 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.

7 Callback Functions

7.1 KYFG_CameraCallbackRegister()

Register a camera simulation runtime callback function. The callback (userFunc) will be called at the end of each new frame generation, of a valid stream from specific camera, with appropriate STREAM_HANDLE. Use the Stream interface functions to handle received data. Additionally, registered user context pointer is retrieved which consequently can be interpreted by host application for internal use.

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

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

Return value

FGSTATUS - Status and error report.

7.2 KYFG_CameraCallbackUnregister()

Unregisters a previously registered camera simulation runtime callback function.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
userFunc	CameraCallback	Pointer to callback function

Return value

FGSTATUS - Status and error report.

7.3 KYDeviceEventCallBackRegister ()

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

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

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

Return value

FGSTATUS - Status and error report.

7.4 KYDeviceEventCallBackUnregister ()

Unregisters a previously registered camera simulation runtime callback function.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle that represents PCI device
userFunc	KYDeviceEventCallBack	Pointer to callback function

Return value

FGSTATUS - Status and error report.

7.5 KYFG_SetCameraValue()

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

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

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

Return value

FGSTATUS - Status and error report.

7.5.1 KYFG_SetCameraValueInt()

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

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera

paramName	const char*	Name of configuration parameter
value	long long	Value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

7.5.2 KYFG_SetCameraValueFloat()

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

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
value	Double	Floating point value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

7.5.3 KYFG_SetCameraValueBool()

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

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
value	KYBOOL	Boolean value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

7.5.4 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.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
value	long long	Enumeration value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

Example code

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

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

7.5.5 KYFG_ExecuteCommand()

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

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

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

Return value

FGSTATUS - Status and error report.

7.5.6 KYFG_SetCameraValueString()

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

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
value	const char*	String value of chosen camera configuration

Return value

FGSTATUS - Status and error report.

7.5.7 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.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
paramValueName	const char*	Name of parameter enumeration choice

Return value

FGSTATUS - Status and error report.

7.6 KYFG_GetCameraValueType()

Get the camera configuration field type.

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

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

Return value

Type of camera parameter as described in KY_CAM_PROPERTY_TYPE enumeration.

7.7 KYFG_GetCameraValue()

Get camera configuration field value.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramValue	void*	Pointer to camera configuration value

Return value

FGSTATUS - Status and error report.

7.7.1 KYFG_GetCameraValueInt()

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

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

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

Return value

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

7.7.2 KYFG_GetCameraValueEnum()

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

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

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

Return value

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

Example code

```
…
linkconfig = KYFG_GetCameraValueEnum (CamHandleArray[0], “LinkConfig”);
…
```

7.7.3 KYFG_GetCameraValueFloat()

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

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

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

Return value

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

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

7.7.4 KYFG_GetCameraValueBool()

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

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

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

Return value

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

7.7.5 KYFG_GetCameraValueString ()

Get camera configuration value of String type field. Value is copied to user allocated char array.

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

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

Return value

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

FGSTATUS - Status and error report.

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

Remarks

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

Example code

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

7.7.6 KYFG_GetCameraValueString()

Get camera configuration value of String type field.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to chosen camera
paramName	const char*	Name of configuration parameter
ptr	char**	Pointer to pointer of string value for chosen parameter

Return value

FGSTATUS - Status and error report.

Remarks

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

Known issues:

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

Example code

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

8.1 KYFG_StreamCreateAndAlloc()

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

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

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

Return value

FGSTATUS - Status and error report.

Example code

```
CAMHANDLE camHandleArray[4] = {0};  // maximum 4 cameras can be connected
STREAM_HANDLE streamHandle = 0;
…
if (FGSTATUS_OK == (KYFG_StreamCreateAndAlloc (camHandleArray[0],
                                               &streamHandle,
                                               16,
```

```
                                                                0))
    {
        printf("New stream was allocated with handle %X", streamHandle);
    }
```

8.2 KYFG_StreamGetSize()

Retrieves the size of the last generated frame of the specified stream.

```
int64_t KYFG_StreamGetSize (
        STREAM_HANDLE streamHandle);
```

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value

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

8.3 KYFG_StreamGetFrameIndex()

Retrieves the index of the last generated frame of the specified stream.

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

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to a stream

Return value

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

8.4 KYFG_StreamGetPtr()

Retrieves a pointer to data memory space of a single frame in the chosen stream.

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

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

Return value

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

Example code

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

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

8.5 KYFG_StreamDelete()

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

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

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle of a stream

Return value

FGSTATUS - Status and error report.

9.1 KYFG_LoadPatternData()

Allocates the needed space in memory and commits it to stream as a video source for simulation. Several patterns types are available for generation. Patterns image format is defined by the camera configuration parameters regardless whether it's colored or non-colored pattern.

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

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to 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.

Return value

FGSTATUS - Simulator status and error report.

Remarks

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.

Example code

Example code can be found under "<User Documents>/KAYA Instruments/Vision Point/API Samples/KY_Simulation_Example.c"

9.2 KYFG_LoadFileData()

Allocates the needed space in memory, and commits it to stream as a video source for simulation. Image types ".bmp", ".tif" , ".pgn", and ".raw" are supported. A RAW file may contain several frames; number of frames is calculated according to image format configurations and RAW file size.

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

Parameter name	Type	Description
streamHandle	STREAM_HANDLE	API handle to chosen simulator
path	const char*	The path of chosen image file
type	const char*	Type of image file: "bmp", "tip", "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 "<User Documents>/KAYA Instruments/Vision Point/API Samples/KY_Simulation_Example.c"

10.1 KYFG_CameraStart ()

Starts video stream generation for the chosen camera. Number of frames to be generated should be specified by 3rd parameter, passing 0 starts continuous generation mode that should be stopped by making KYFG_CameraStop() call.

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

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

Return value

FGSTATUS - Status and error report.

10.2 KYFG_CameraStop ()

Stops generation for the chosen camera.

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

Parameter name	Type	Description
camHandle	CAMHANDLE	API handle to connected camera

Return value

FGSTATUS - Status and error report.

11.1 KYFG_CheckUpdateFile

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

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

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

pFileVendorId	uint16_t *	Pointer to uint16_t that will be filled with vendor ID of firmware contained in the file
pFileBoardId	uint16_t *	Pointer to uint16_t that will be filled with board ID of firmware contained in the file

Return value

FGSTATUS - Status and error report.

11.2 KYFG_LoadFirmware

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

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

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

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

12.1 API handles

- FGHANDLE – API handle that represents KAYA's PCI device.
  INVALID_FGHANDLE – Markup for invalid FGHANDLE.
- CAMHANDLE – API handle that represents a camera being simulated.
  INVALID_CAMHANDLE – Markup for invalid CAMHANDLE.
- STREAM_HANDLE – API handle that represents video generation stream.
  INVALID_STREAMHANDLE – Markup for invalid STREAM_HANDLE.

12.2 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
```

12.3 CameraCallback

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

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

12.4 KYDeviceEventCallBack

Generic callback function prototype. Generic callbacks are issued to inform user's application about various events in the system. For more details see each event description

```
typedef void(KYFG_CALLCONV * KYDeviceEventCallBack)(
        void* userContext
        KYDEVICE_EVENT* pEvent);
```

12.5 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);
```

12.6 UPDATE_CALLBACK

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

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

13.1 FGSTATUS

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

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

_SECTION		
FGSTATUS_WRONG_PARAMETERS_SECTION	0x3013	Saved values configurations for loading wasn’t found
FGSTATUS_VALUE_HAS_NO_SELECTOR	0x3014	The parameter is not a part of a selector type field
FGSTATUS_CALLBACK_NOT_ASSIGNED	0x3015	No callback is assigned for data retrieval
FGSTATUS_HANDLE_DOES_NOT_MATCH_CONFIG	0x3016	The value of Camera Selector doesn't match the provided Camera handle This will indicate that a wrong CAMHANDLE is introduced for set/get Grabber parameter functions, which contradictory the “CameraSelector” currently set.
FGSTATUS_BUFFER_TOO_SMALL	0x3017	Provided buffer length is too small to hold the amount of information needed to be filled in the provided buffer
FGSTATUS_EXCEEDED_MAX_CAMERA_CONNECTIONS	0x3100	Number of connected cameras exceeds the maximum allowed connected cameras
FGSTATUS_UNKNOWN_ERROR	0x3FFF	Unknown error

13.2 CSSTATUS

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
CSSTATUS_OK	0x2000	The operation has successfully executed
CSSTATUS_UNKNOWN_SIM_HANDLE	0x2001	Unknown API handle
CSSTATUS_HW_NOT_FOUND	0x2002	Error with hardware. Hardware function failed to execute.
CSSTATUS_BUSY	0x2003	Can’t execute 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 file, error in file

		or the file is to long
CSSTATUS_CONFIG_NOT_LOADED	0x2006	Can't load current camera configuration
CSSTATUS_INVALID_VALUE	0x2007	The value given as parameter is out of 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	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 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 camera configuration values file
CSSTATUS_BUFFER_NOT_LOADED	0x2018	Incompatible buffer was selected, or no buffer was allocated
CSSTATUS_UNKNOWN_ERROR	0x2FFF	Unknown error

13.3 PATTERN_TYPE

Patterns generation types. Actual pattern scaling and format is defined by values in the appropriate camera configuration fields.

Enumeration 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

13.4 KYDEVICE_EVENT_ID

‘eventId’ field of KYDEVICE_EVENT structure:

Enumeration Field	Description
KYDEVICE_EVENT_CAMERA_START_REQUEST	Detected remote request to start simulation on a camera. Pointer passed to KYDeviceEventCallBack function should be C-casted to KYDEVICE_EVENT_CAMERA_START

13.5 KY_CAM_PROPERTY_TYPE

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

Enumeration Field	Value	Description
PROPERTY_TYPE_INT	0x00	Camera configuration of Integer type

PROPERTY_TYPE_BOOL	0x01	Camera configuration of Boolean type
PROPERTY_TYPE_STRING	0x02	Camera configuration of String type
PROPERTY_TYPE_FLOAT	0x03	Camera configuration of Floating Point type
PROPERTY_TYPE_ENUM	0x04	Camera configuration of Enumeration type
PROPERTY_TYPE_COMMAND	0x05	Camera configuration of Command type
PROPERTY_TYPE_UNKNOWN	-1	Camera configuration of an unknown type

13.6 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

13.7 VIDEO_DATA_TYPE

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

Enumeration Field	Value	Description
DATA_TYPE_MONO	0x01	This is used for luminance data. This has no sub-types. This is defined in Table 27 of JIIA CXP standard document
DATA_TYPE_PLANAR	0x02	This is used for planar data, such as individual red, green or blue planes, additional alpha (overlay) planes, or the separate planes in YUV420. This is defined in Table 28 JIIA CXP

		standard document. Subtypes include all the DATA_SUBTYPE_PLANAR_xx
DATA_TYPE_BAYER	0x03	This is used for Bayer data. This is defined in Table 29 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_BAYER_xx
DATA_TYPE_RGB	0x04	This is used for RGB data, transmitted in the order red, green, blue. This has no sub-types. This is defined in Table 30 JIIA CXP standard document.
DATA_TYPE_RGBA	0x05	This is used for RGBA data, where "A" is the alpha (or overlay) plane, transmitted in the order red, green, blue, alpha. This has no sub-types. This is defined in Table 31 JIIA CXP standard document.
DATA_TYPE_YUV	0x06	This is used for YUV data. This is defined in Table 32 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_YUV_xxx
DATA_TYPE_YCBCR601	0x07	This is used for YCbCr data, as specified by ITU-R BT.601.This is defined in Table 33 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_UCBCR_xxx
DATA_TYPE_YCBCR709	0x08	This is used for YCbCr data, as specified by ITU-R BT.709. This is defined in Table 34 JIIA CXP standard document. Subtypes include all DATA_SUBTYPE_UCBCR_xxx

13.8 VIDEO_DATA_SUBTYPE

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

Enumeration Field	Value	Description
DATA_SUBTYPE_NONE	0x00	None
DATA_SUBTYPE_PLANAR_RY	0x01	Standard usage: R, Y
DATA_SUBTYPE_PLANAR_GUCB	0x02	Standard usage: G, U, Cb
DATA_SUBTYPE_PLANAR_BVCR	0x03	Standard usage: B, V, Cr
DATA_SUBTYPE_BAYER_GR	0x01	1st line transmission order G, R. 2nd line transmission order B, G
DATA_SUBTYPE_BAYER_RG	0x02	1st line transmission order R, G. 2nd line

		transmission order G, B
DATA_SUBTYPE_BAYER_GB	0x03	1st line transmission order G, B. 2nd line transmission order R, G
DATA_SUBTYPE_BAYER_BG	0x04	1st line transmission order B, G. 2nd line transmission order G, R
DATA_SUBTYPE_YUV_411	0x01	Transmission order Y, Y, U, Y, Y, V
DATA_SUBTYPE_YUV_422	0x02	Transmission order Y, U, Y, V
DATA_SUBTYPE_YUV_444	0x03	Transmission order Y, U, V
DATA_SUBTYPE_YCBCR_411	0x01	Transmission order Y, Y, Cb, Y, Y, Cr
DATA_SUBTYPE_YCBCR_422	0x02	Transmission order Y, Cb, Y, Cr
DATA_SUBTYPE_YCBCR_444	0x03	Transmission order Y, Cb, Cr

14.1 KYDEVICE_EVENT

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

Structure Field	Type	Description
eventId	KYDEVICE_EVENT_ID	Identifier of an event.

14.1.1 KYDEVICE_EVENT_CAMERA_START

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

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

14.2 VIDEO_PIXELIF

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

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

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

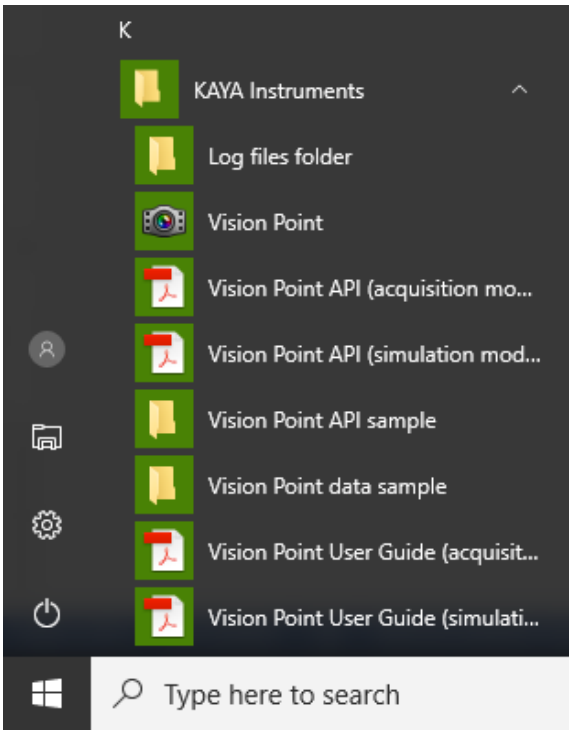
14.3 UPDATE_STATUS

Firmware update progress supplied via parameter of UPDATE_CALLBACK

Structure Field	Type	Description
struct_version	int	Currently code initializes this with "1". If more fields will be added to this stuct in future code will be changed and initialize it with "2", etc.
link_mask	uint64_t	bytes already sent.
link_speed	uint64_t	firmware file size.
is_writing	KYBOOL	Indicates current phase: KYTRUE - writing new firmware, KYFALSE - validating new firmware.

15.1 API example for Windows

1. Open an API example project KY_Simulation_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.



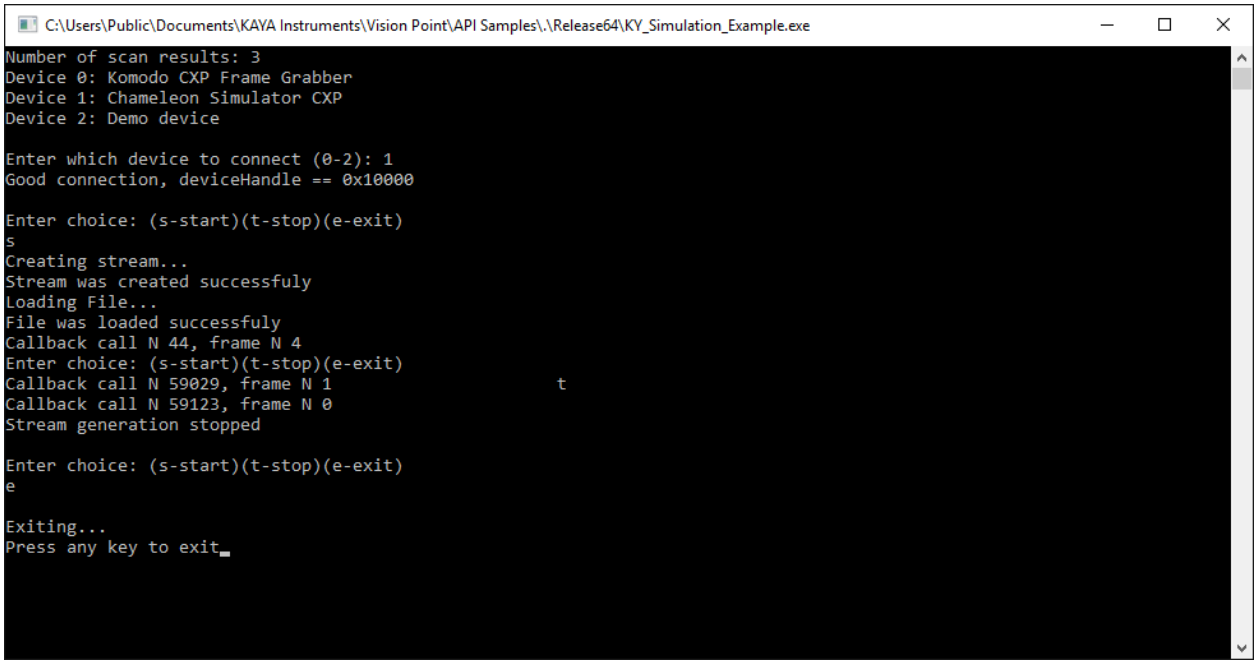
2. Choose solution platform according to your operation system, as shown in the image below.
   Note: The Vision Point software stack does not support Win32 platform with OS x64.

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

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

Command	Description
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.

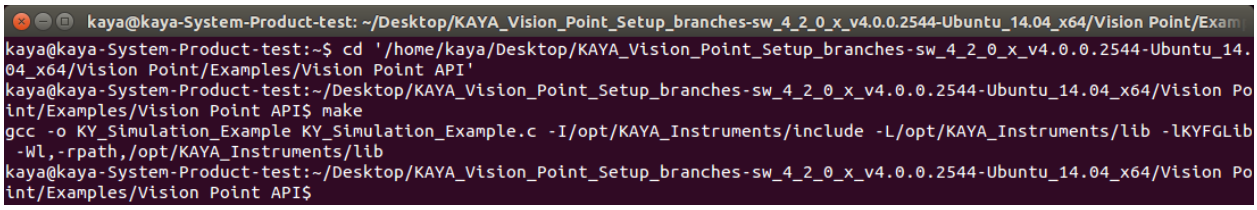


NOTE:

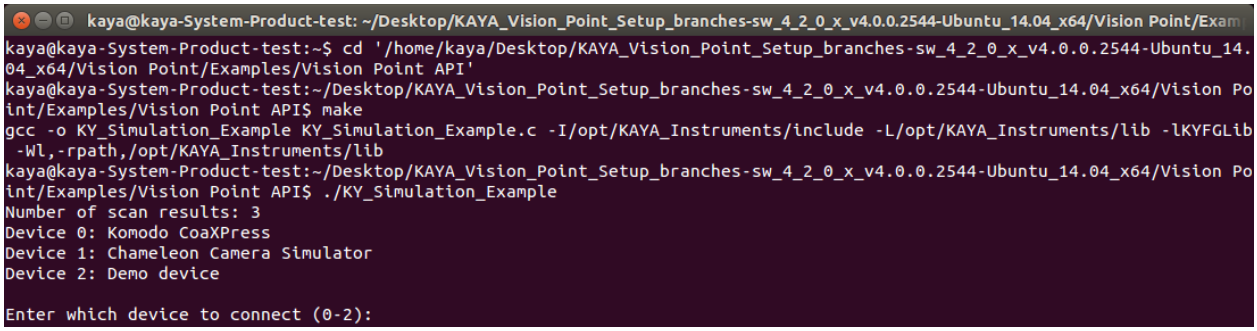
By default, the KYFGLib_Example.vcxproj project contains KYFGLib_Example.c which uses **Cyclic Buffer**. In order to use **Queued Buffer**, in the loaded project, change the KYFGLib_Example.c file to KYFGLib_Example_QueuedBuffers.c located in the same directory, and rebuild example.

15.2 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/ directory.
2. Type “make” and make sure the KY_Simulation_Example executable file was created, in the same directory. The image below shows the 1st and the 2nd step.



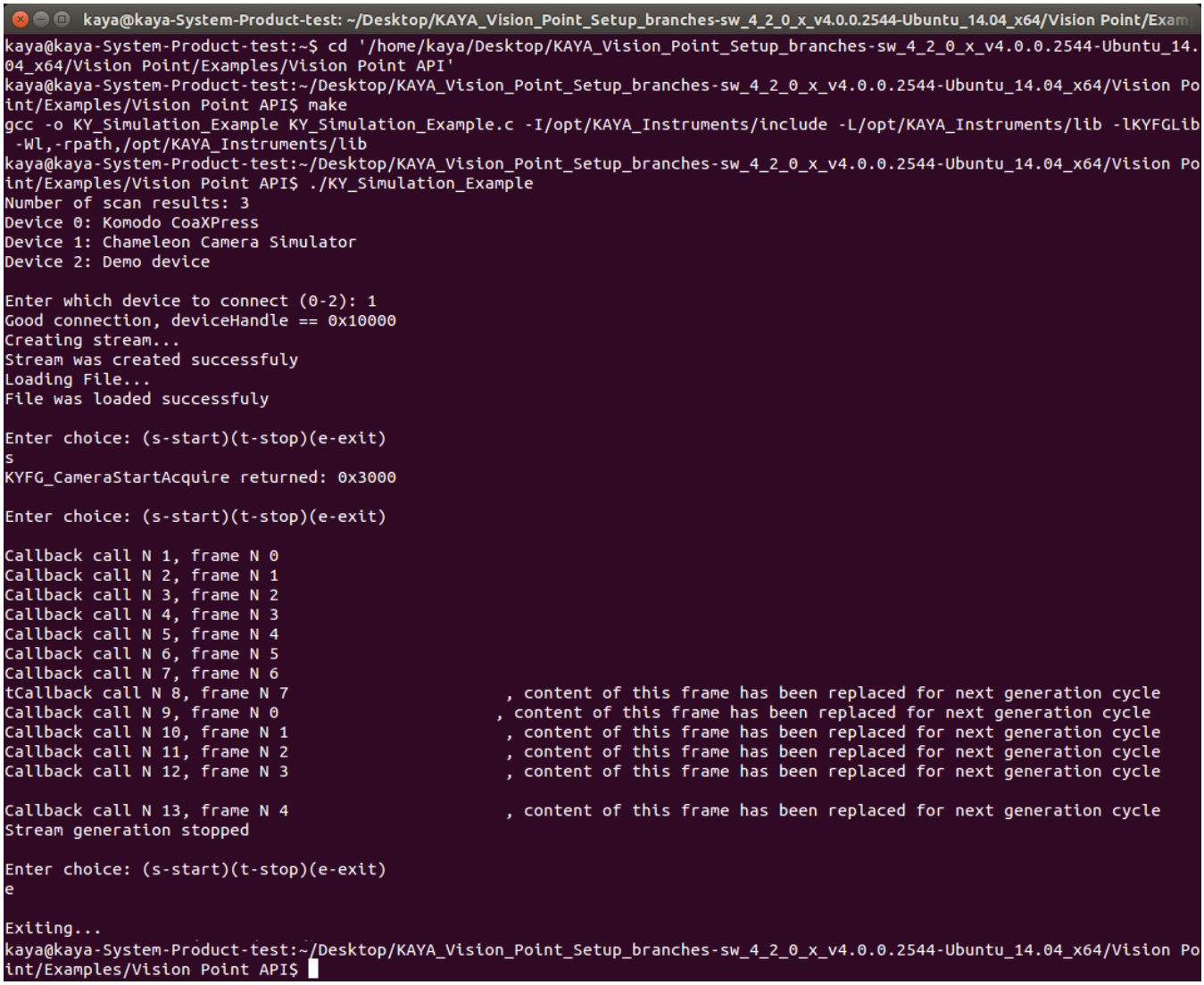
3. To run the API Example, simply type “./KY_Simulation_Example” followed by “Enter”, as shown in the image below.



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

Command	Description
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.



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.

16.1 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 2, and select the desired firmware update file, in accordance with the device chosen (.bin file extension), and Click "Next >" button.

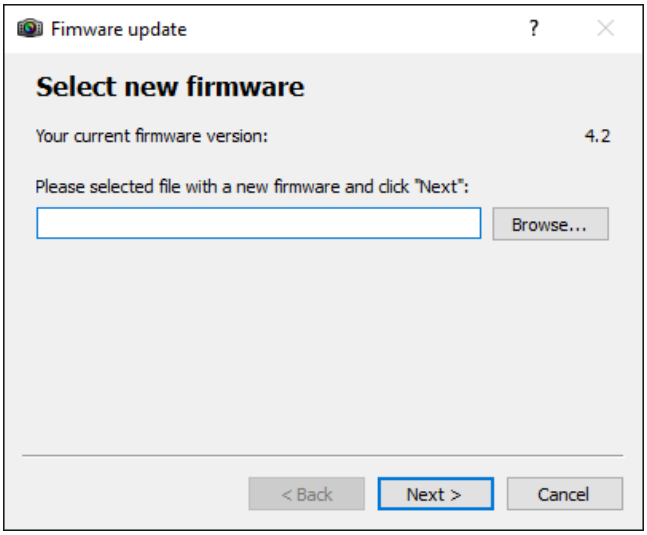


Figure 2 : Firmware Update selection window

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

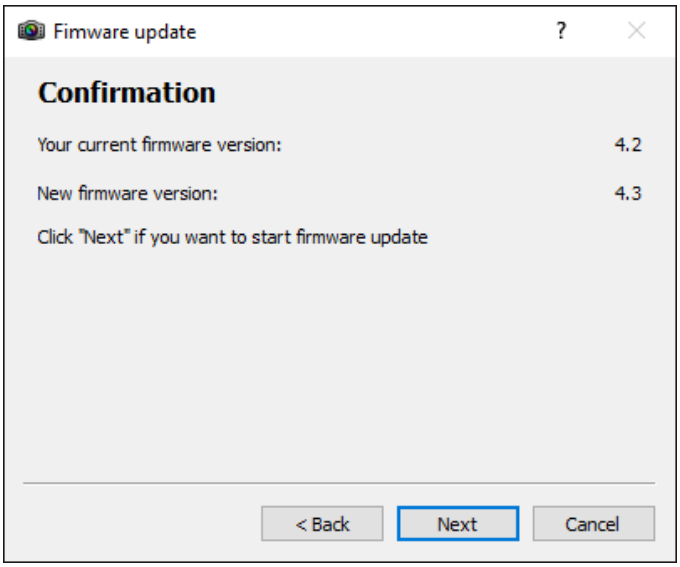


Figure 3 : Firmare Update Confirmation window

4. The next window displays the initiated firmware update. The firmware update process displayed in the first progress bar and the firmware validation displayed in the second, as shown in Figure 4.
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.

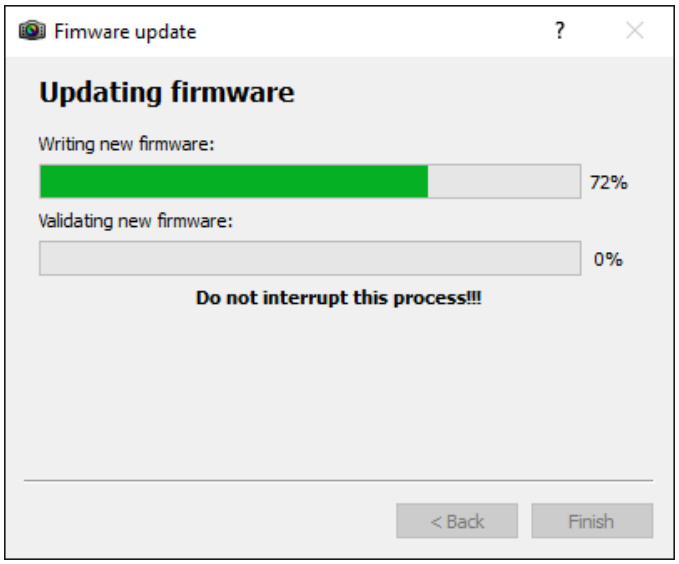


Figure 4 : Firmare Update process window

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

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

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

1. Make sure the .bin file is present in a local directory.
2. Open the Terminal and enter the directory path of KAYA Hardware Update executable file: "cd 'opt/KAYA_Instruments/bin' ".
3. Execute the KAYA Hardware Update using full path to the firmware update file as a parameter. Example: "./KAYA_Hardware_Update <*path_to_folder_with_bin_file*>/Komodo_4_3.bin ".
4. Press Enter and wait for a message that indicates the end of process.
5. **Do not interrupt the process!**
6. **A full PC power off cycle is required to activate the new firmware**.
7. The sequence of the steps is illustrated in the screenshot below.

Please, Contact KAYA Instruments' representative for any question.

Figure 5 : Firmare Update via Terminal process window

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

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

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

8. Make sure the .bin file is present in a local directory.
9. Open the Command line and enter the directory path of KAYA Hardware Update executable file:
   "cd '\Program Files\KAYA_Instruments\Common\bin' ".
10. 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*>/Chameleon_4_3.bin ".
11. Press Enter and wait for a message that indicates the end of process.
12. **Do not interrupt the process!**
13. **A full PC power off cycle is required to activate the new firmware**.
14. The sequence of the steps is illustrated in the screenshot below.

Please, Contact KAYA Instruments' representative for any question.

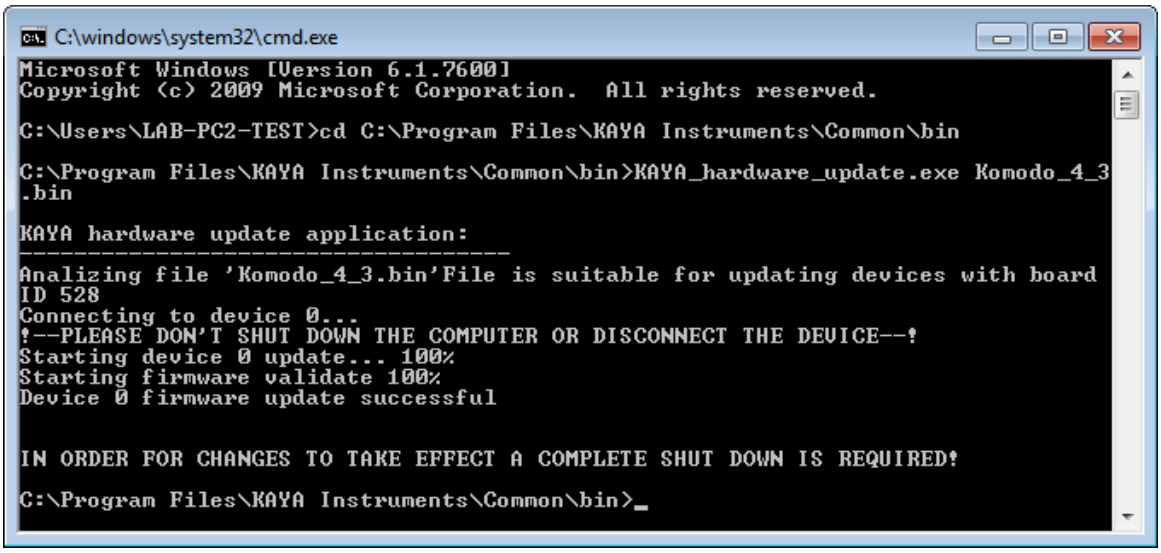


Figure 6 : Firmare Update via Command line process window

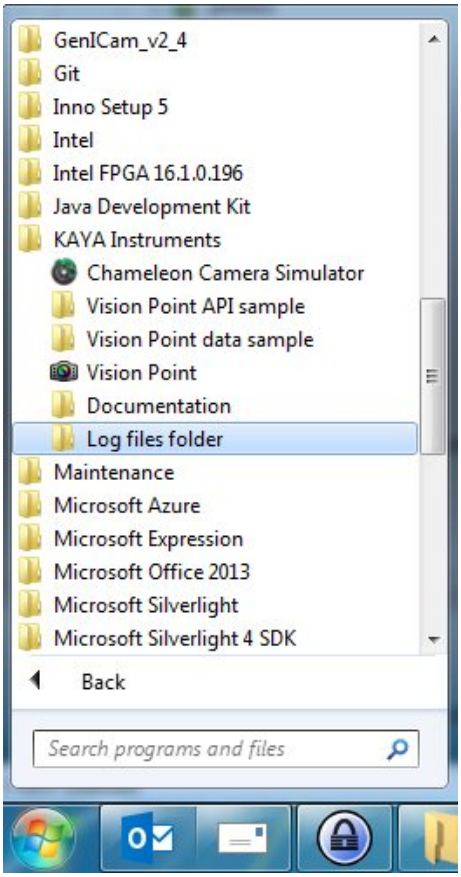
16.4 Collecting log files

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

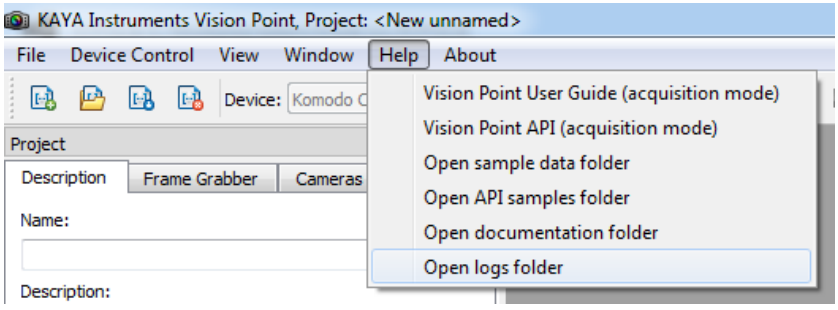
Windows operating system:

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

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



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



NOTES:

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

Linux operating system:

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

16.5 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.

16.6 Submitting a support request

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

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