Programmer's Guide
Baumer GAPI SDK v2.10.0

Document Version: v2.8
Release: 05.12.2019
Document Number: 11099008

Table of Contents

Linux

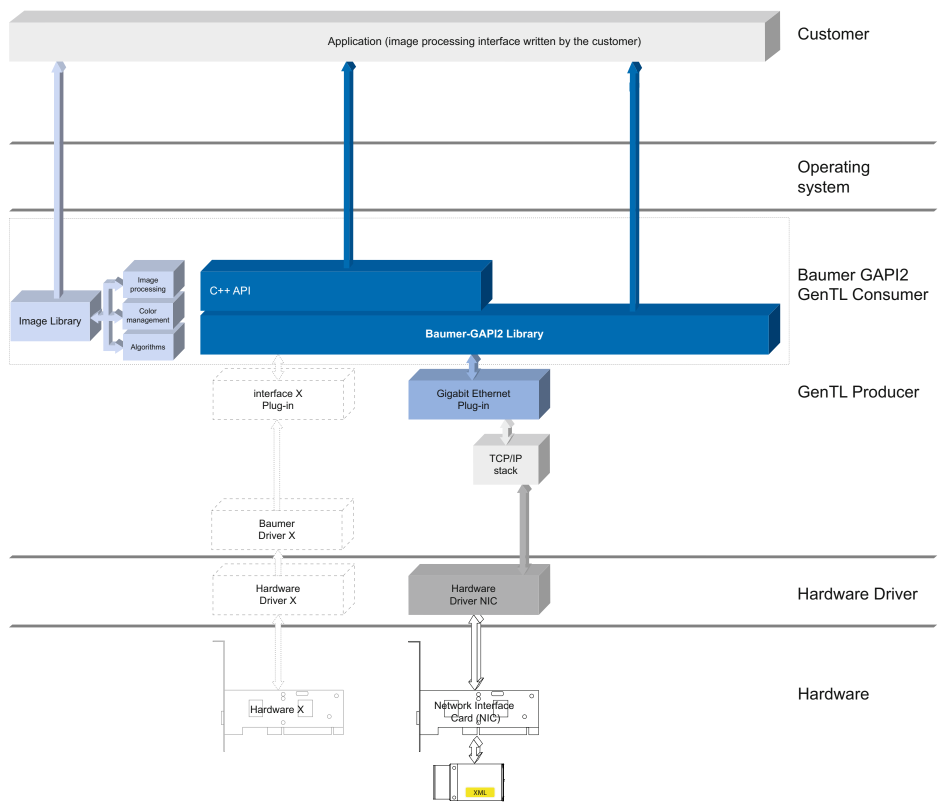
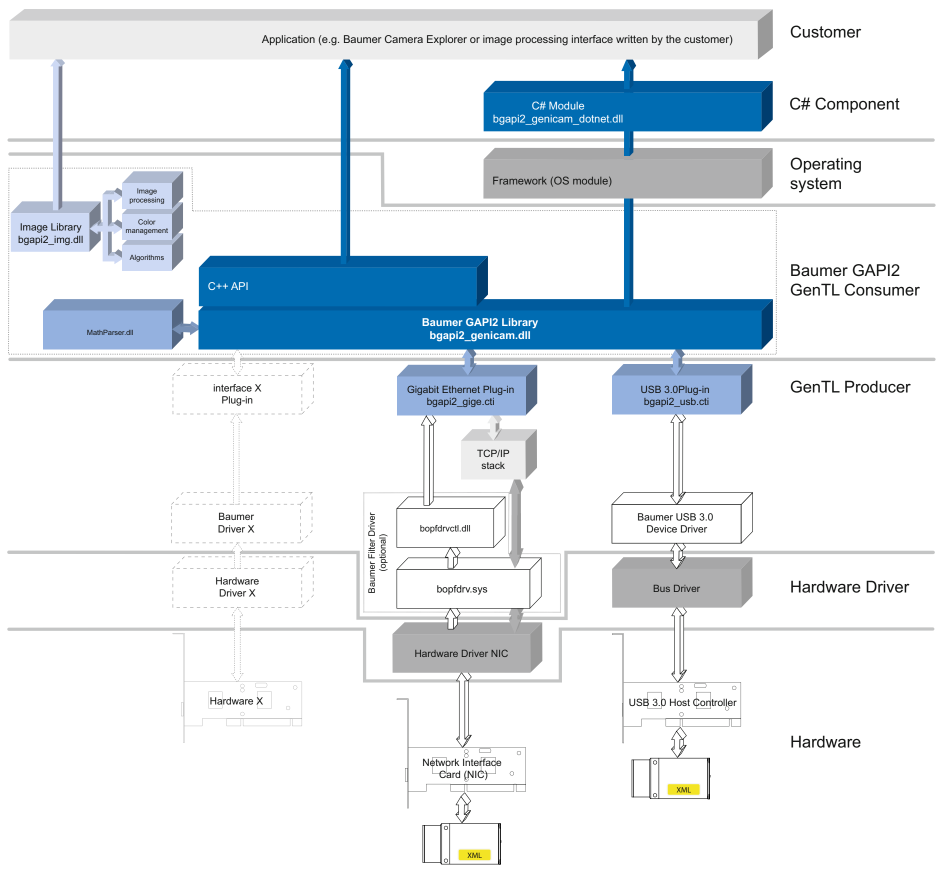


Figure 1 ►
Baumer GAPI stack (Linux)

Windows

Framework:
Baumer GAPI is tested with Windows®: .NET™ 4.0



Application (e.g. Baumer Camera Explorer or image processing interface written by the customer)
Customer
C# Module bgapi2_genicam_dotnet.dll
C# Component
Framework (OS module)
Operating system
Image Library bgapi2_img.dll
Image processing
Color management
Algorithms
C++ API
MathParser.dll
Baumer GAPI2 Library bgapi2_genicam.dll
Baumer GAPI2 GenTL Consumer
interface X Plug-in
Gigabit Ethernet Plug-in bgapi2_gige.cti
USB 3.0Plug-in bgapi2_usb.cti
GenTL Producer
TCP/IP stack
Baumer Driver X
Baumer Filter Driver (optional)
bopfdrvctl.dll
Baumer USB 3.0 Device Driver
Hardware Driver X
bopfdrv.sys
Bus Driver
Hardware Driver
Hardware Driver NIC
Hardware X
USB 3.0 Host Controller
Network Interface Card (NIC)
XML
XML
Hardware

Figure 2 ►
Baumer GAPI stack (Microsoft® Windows®)

1. Introduction

This programmer’s guide is for programmers who need to integrate Baumer cameras into their own software. It corresponds with the GenICam API for Baumer GAPI and the use of Microsoft® Visual Studio® from version 2005 (C++) and version 2010 (C#) onwards for Windows® operating systems.

Installation:
For further information on installing the Baumer GAPI SDK, refer to the appropriate Installation Guide for your operating system and hardware interface.

2. Baumer GAPI

Baumer GAPI is the abbreviation for **B**aumer **G**eneric **A**pplication **P**rogramming **I**nterface. With this API, Baumer provides an interface for optimal integration and control of Baumer cameras.

The fundamental basis for Baumer GAPI is the GenICam (Generic Interface for Cameras) standard and the use of GenTL as standardized interface.

Baumer GAPI provides interfaces to several programming languages such as C++ and to the .NET™ Framework on Windows® operating systems, which in turn allow the use of other languages such as C# or VB.NET.

API:
API stands for "Application Programming Interface". An API is a software interface between two programs, usually between an operating system and an application.

Notice
On Linux only C++ is available.

Generic:
- new functions are added dynamically - no software changes required

- functions are defined in XML format and are imported from the API.

- divided into Standard Features and Baumer Specific

2.1 Baumer GAPI Stack Components

2.1.1 Supported Hardware

2.1.1.1 Gigabit Ethernet

Working with Baumer Gigabit Ethernet cameras requires the installation of appropriate hardware – such as a NIC (Network Interface Card) - on your PC.

Notice
For Gigabit Ethernet, Baumer recommends the use of NICs with an Intel® chipset.

The hardware is delivered with a hardware driver that is required to establish communications between hardware and software.

GenICam:
GenICam stands for "Generic Interface for Cameras" and is a generic programming interface for industrial cameras. Its objective is to decouple industrial camera interface technology from the user application programming interface (API).

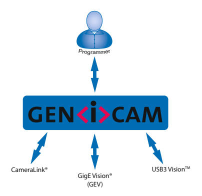
2.1.1.2 USB 3.0 Vision cameras

Working with Baumer USB 3.0 Vision cameras requires the installation of appropriate hardware on your PC. The USB drivers are installed during the installation process.

2.1.2 Baumer GAPI

2.1.2.1 Producer

A complete camera system is mapped to the software within the GenICam Producer . The Producer provides the 5 main classes (*System*, *Interface*, *Device*, *DataStream* and *Buffer*) that are used in any Baumer GAPI application.



▲ **Figure 3**
GenICam Standard

2.1.2.2 Consumer

The Consumer is the counterpart of the Producer mentioned above. This manages all of the available Producers (loading, opening, closing).

Furthermore, the Consumer provides all additional classes and enables exception handling.

2.1.2.3 Framework

A Framework represents a software platform that provides a runtime environment, an API and several programmer services. It is required, for example, when programming in C# or VB.NET.

2.2 Software Components

The Baumer GAPI is a package of several libraries in the form of dynamic link libraries. The Producer referred to above represents an exception. The Producer is delivered as a common transport interface (.cti-file).

2.2.1 Baumer GAPI Modules

The Baumer GAPI is partitioned into several modules. Depending on your chosen programming language, more or fewer of these may be used.

2.2.1.1 Baumer GAPI Library

As previously stated, Baumer GAPI supports several programming languages to create an application. Therefore the Baumer GAPI Library represents a uniform API for cross-interface control of *System*, *Interface*, *DataStream*, *Buffer* and *Device*.

2.2.1.2 Image Library

This library includes several functions for image processing.

2.2.1.3 Programming Languages

As shown in Figure 1, the implementation of C++ and C# is partially done hierarchically.

This means that the C++ API and C# API directly refer to the Baumer GAPI library.

Notice
On Linux only C++ is available.

2.3 Requirements / Recommended equipment

2.3.1 General System Requirements

		Single-camera system Recommended	Multi-camera system Recommended
CPU		Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz, Cores: 4	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, Cores: 8
RAM		4 GB	8 GB
		Operating system (OS)	
GigE Vision	**Windows®**	Microsoft® Windows® 7 32/64 bit systems Microsoft® Windows® 8 32/64 bit systems Microsoft® Windows® 10 32/64 bit systems	
	Linux	Debian 7.11 / 8.6 Fedora 23 / 24 OpenSUSE 13.1 / 13.2 Ubuntu 14.04 / 16.04	
USB 3.0 Vision	**Windows®**	Microsoft® Windows® 7 32/64 bit systems Microsoft® Windows® 8 32/64 bit systems Microsoft® Windows® 10 32/64 bit systems	
	Linux	Linux Kernel ≥ 3.3 Debian 8.6 Fedora 23 / 24 OpenSUSE 13.1 / 13.2 Ubuntu 14.04 / 16.04 **Notice** The USB module of the Linux® kernel features a memory limit of 16 MB, which will lead to restrictions in the use of USB multi camera systems. Observe the notes in the Application Note: *Baumer-GAPI-Multi-USB-Camera-systems-on-Linux.pdf* to prevent these restrictions.	
Compiler	**Windows®**	Microsoft® Compiler only (included in Microsoft® Visual Studio)	
	Linux	GCC ≥ v 4.7	
C++ Version		C++11	
Framework (optional)		Windows® OS: .Net Framework ≥ v 3.51 for C# implementation	

Notice

Older versions of Microsoft® Visual Studio (e.g. Microsoft® Visual Studio 2005) that do not provide C++11 support will still work as the Windows® version is statically linked to the C++ runtime libraries.

Notice

For full use of GigE Vision the blocking of the firewall must be repealed.

2.3.2 Recommended equipment

Help -File / Header / Camera User Guides

The individual functions of the Baumer GAPI SDK are described in the PDF file provided (*ProgrammersGuide_Baumer_SDK_Reference.pdf*).

Following installation, the file can be found under: Docs/Programmer´s Guide/

Main Class Features

As well as the standard main class features, special Baumer main class features are also available.

System, *Interface*, *Device*, *DataStream*, *Buffer* and the *Image* and *Image processor*-defined features can be requested for all main classes (see documentation Main Class Features (Standard / Baumer specific) with the methods GetNodeTree and GetNodeList.

These features can also be found in the *Main Class Features* section of *Baumer Camera Explorer*.

Camera Features

As well as the standard camera features, special Baumer camera features are also available.

Notice

The camera you use determines the SFNC version required.

Following installation, you can find the SFNC under: Docs/Programmer´s Guide/

You can download other versions of the SFNC at: http://www.emva.org

Reading out your camera's camera features via Baumer Camera Explorer

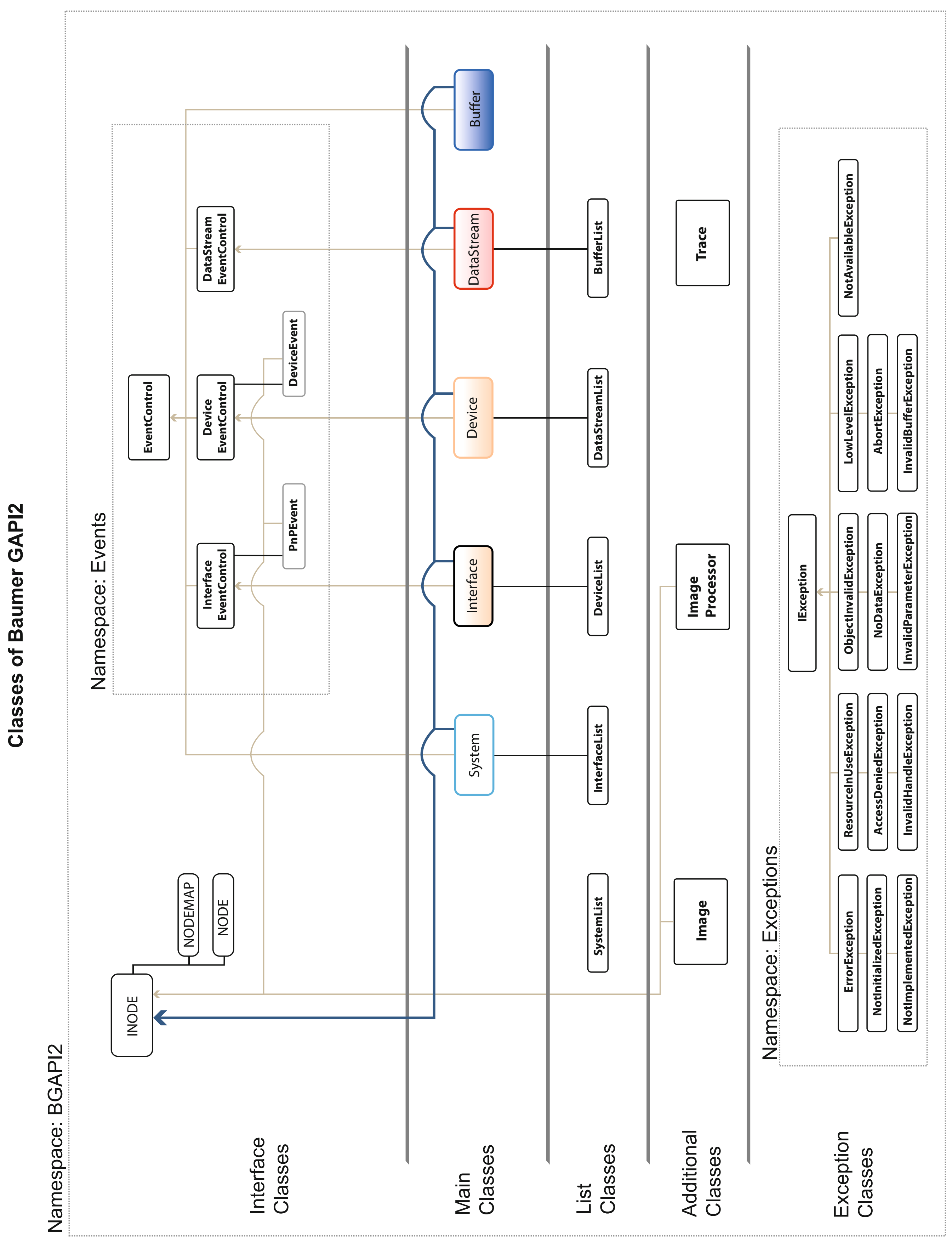
You can read out the range of features supported by your camera using *Baumer Camera Explorer*.

1. Start *Camera Explorer*.
2. Start your connected camera by clicking on the camera icon.
3. Set the profile to *Guru*. The *Feature Tree* will now show all the categories with your camera's features.

Alternatively, you can use the XML button (only visible in Advanced Mode) to display the XML description file for the camera. This also lists and describes the camera features.

Programming Examples (SDK Examples)

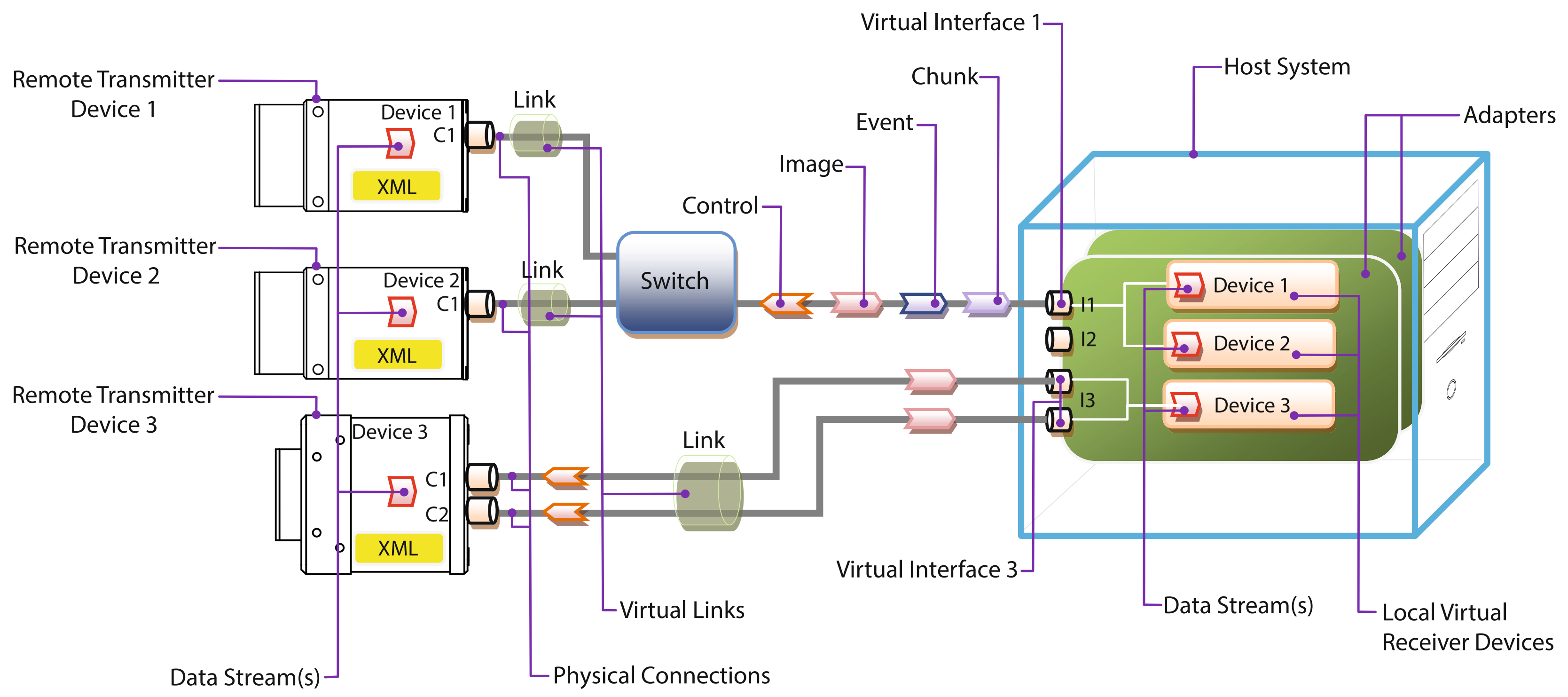
The programming examples included are only intended to be used as guidelines. The output may be different depending on the OS and interface types.



▲ **Figure 4**
Baumer GAPI 2 classes

Device Communication Model

Reference: GenICam Standard Features Naming Convention v2.1

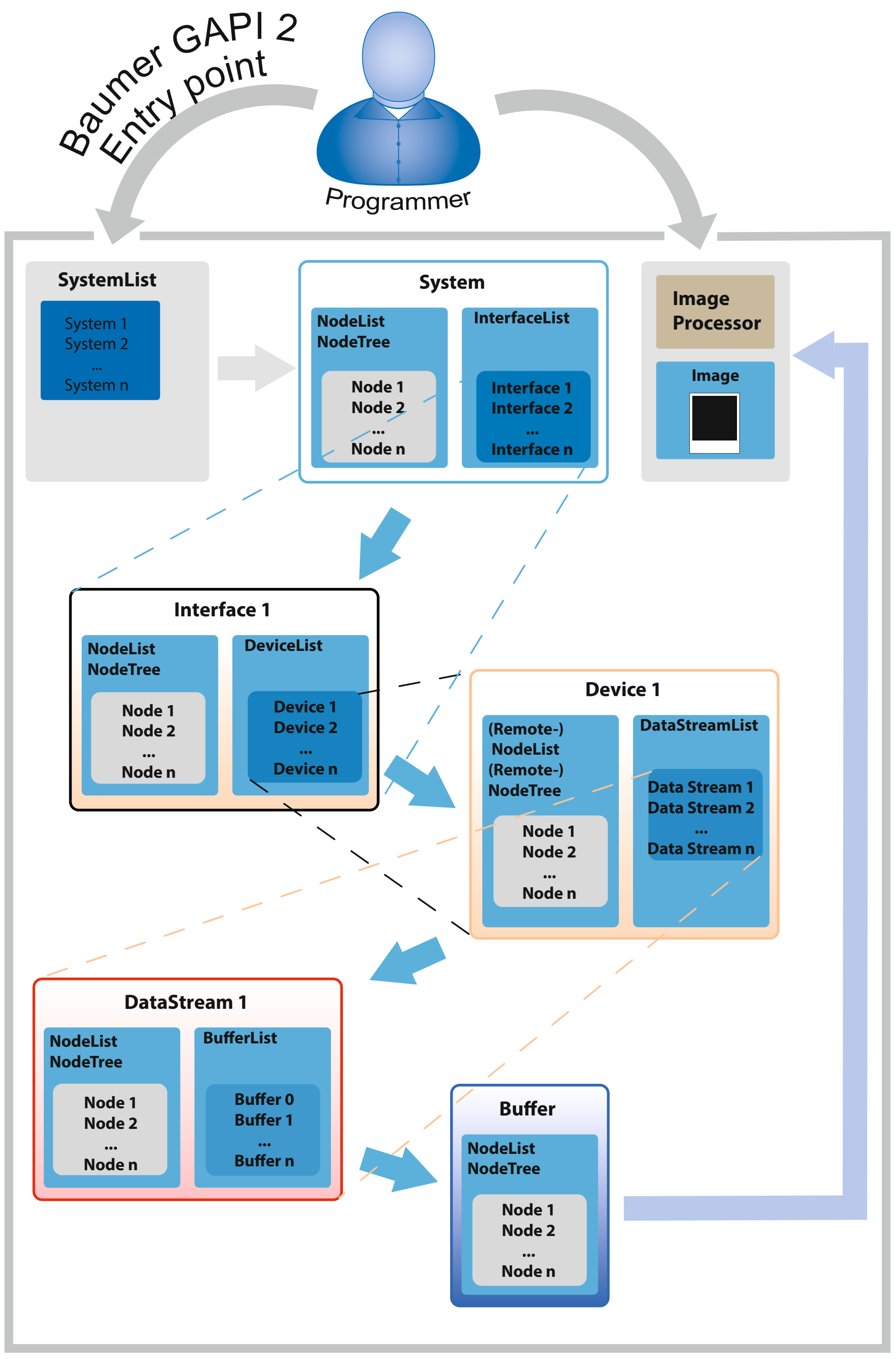


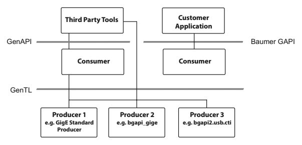
3. Central Idea Behind Baumer GAPI

The fundamental idea behind the Baumer SDK was to release programmers from the need to define and instantiate all the required and usable objects, and to transfer these tasks to the Baumer GAPI.

The API is based on five main classes (*System*, *Interface*, *Device*, *DataStream*, *Buffer*) and the use of the GenTL interface.

There are two entry points for the user in Baumer GAPI. The first option is to initiate application programming by using the system list.





Third Party Tools
Customer Application
GenAPI
Baumer GAPI
Consumer
Consumer
GenTL
Producer 1 e.g. GigE Standard Producer
Producer 2 e.g. bgapi_gige
Producer 3 e.g. bgapi2.usb.cti

▲ **Figure 5**
Fundamental Approach of GenAPI and GenTL

3.1 Interface Classes

The majority of the Baumer GAPI functionality is provided by the two interface classes NodeInterface and EventInterface. For instance, the main classes inherit functionalities from INode.

3.1.1 INode, NodeMap, Node

INode <u>Parent class for:</u> *System* *Interface* *Device* *DataStream* *Buffer* *DeviceEvent* *PnPEvent* *ImageProcessor* *Image*	The INode class provides direct access to all features of the module XML description file. Feature means all elements specified as `<pFeature>` in the XML description file.

Functions of the INode class

`BGAPI2::INode:: ...`	
`GetNode(String name);`	Returns a pointer to a specified object within the Node class.
`GetNodeTree();`	Returns the tree view showing all nodes of features and categories of the XML description file.
`GetNodeList();`	Returns a one-dimensional list of the nodes of the XML description file.

Notice

The functions listed are only sample of all of the NodeMap functions available.

You can find a complete overview in the *ProgrammersGuide_Baumer_SDK_Reference.pdf* file provided.

NodeMap Member of INode	NodeMap is the collection class for nodes.

Functions of the NodeMap class

`BGAPI2::NodeMap:: ...`	
`GetNode(String name);`	Returns a pointer to a specified object within the Node class.
`GetNodeTree();`	Returns the tree view showing all nodes of the XML description file.
`GetNodeList();`	Returns a one-dimensional list of the nodes of the XML description file.

Notice

System, *Interface*, *Device*, *DataStream*, *Buffer* and the *Image* and *Imageprocessor*-defined features can be requested for all main classes (see documentation Main Class Features (Standard / Baumer-specific) with the methods GetNodeTree and GetNodeList.

NODE Member of INode	A node presents one feature from the XML description file.

Functions of the Node class

BGAPI2::Node:: ...	
`GetInterface();`	Returns the interface of the respective Node.
`GetExtension();`	Returns custom-specific data from the XML description file.
`GetToolTip();`	Returns a Tool-Tip.
`GetDescription();`	Returns the description of the feature.
`GetName();`	Returns the name of the feature.
`GetDisplayName();`	Returns an appropriate name for use in graphical user interfaces (GUI) only.
`GetVisibility();`	Returns the visibility of the node (Invisible, Beginner, Expert, Guru).
`GetImplemented();`	Returns the implementation status of the feature.
`IsReadable();`	Returns the readability of the feature.
`IsWriteable();`	Returns the writability of the feature.
`GetAvailable();`	Returns the availability of the feature.
`GetLocked();`	Returns information as to whether the feature is locked (read only) or not.
`GetImposedAccessMode();`	Returns the startup behavior of the feature
`GetCurrentAccessMode();`	Returns the current access mode
`GetAlias();`	Returns the name of another node which describes the same feature in a different manner
`GetRepresentation();`	Gives a hint as to how to display the feature within the application (GUI).
`GetUnit();`	Returns the unit of the feature [e.g. ms, Hz].
`HasUnit();`	Queries whether the feature has a unit.
`GetEventID();`	Returns the ID of an event.
Enumeration value	
`GetEnumNodeList();`	Returns a list of Enum Entries for the feature.
Set value in string format (General purpose function)	
`GetValue();`	Retrieves values as a converted string.

`SetValue( String Value );`	Sets a value as a string, regardless of the interface type (it will be converted to string).
`Integer functions (only for Integer Interface)`	
`GetInt();`	Retrieves values.
`SetInt( bo_int64 value );`	Sets values as an integer.
`GetIntMin();`	Returns the minimum value.
`GetIntMax();`	Returns the maximum value.
`GetIntInc();`	Returns the step size for setting a value.
`Float functions (only for Interface Type Float)`	
`GetDouble();`	Retrieves values.
`GetDoubleInc();`	Returns the allowed step size for the value of the node object as float.
`HasInc();`	This function delivers a flag that indicates whether the Node object has an increment.
`SetDouble( bo_double value );`	Sets values as an integer.
`GetDoubleMin();`	Returns the minimum value.
`GetDoubleMax();`	Returns the maximum value.
`String functions (only for Interface Type String)`	
`GetMaxStringLength();`	Retrieves the maximum length of the string.
`GetString();`	Retrieves values.
`SetString( String value );`	Sets values as a string.
`Command value (only for Interface Type Command)`	
`Execute();`	One-time execution of a command.
`IsDone();`	Queries whether the command has been executed.
`Boolean value (only for Interface Type Bool)`	
`GetBool();`	Retrieves values (true / false).
`SetBool( bo_bool value );`	Sets boolean values.
`Access type (only for Interface Type Category)`	
`GetNodeTree();`	Returns the tree view showing all nodes.
`GetNodeList();`	Returns a one-dimensional list of the nodes.
`Selectors`	
`IsSelector();`	Queries whether the feature is a selector.
`GetSelectedNodeList();`	Returns a list of features that are influenced by the Selector.
`Register access (only for Interface Type Register)`	
`getLength();`	Delivers the length in bytes of the memory pointed to by the Node object. Only valid for the 'IRegister' interface type.
`getAdress();`	Delivers the address of the memory pointed to by the Node object. Only valid for the 'IRegister' interface type.
`get(void *pBuffer, bo_unint64 len);`	Reads a register.
`set(void *pBuffer, bo_unint64 len);`	Writes a register.

3.1.2 EventControl

This class works as a base class. You can use it to set the event mode.

As an alternative to polling from buffer and event data, a callback functionality is offered, which allows a previously registered function to be called up directly from the *bgapi2_genicam.dll*. The event data can be divided into DeviceEvents, which are generated by the device object, BufferEvents, which are generated by datastream object and PnPEvents, which are generated by the interface object.

3.1.2.1 Event-Handling

Types of events:
- Plug'n'Play Events (Interface)
- Statesignal (Device)
- Newbuffer Events (Datastream)

Eventmodes:
- Unregistered
- Polling (Default)
- Handler

Eventhandler

Eventmode `Polling` is active on start-up. You can switch freely between all 3 Eventmodes. The active Eventmode can be queried using `GetEventMode()`.

Eventmode: Unregistered

The Unregistered Eventmode means that EventHandling is disabled. No events can be retrieved in this mode.

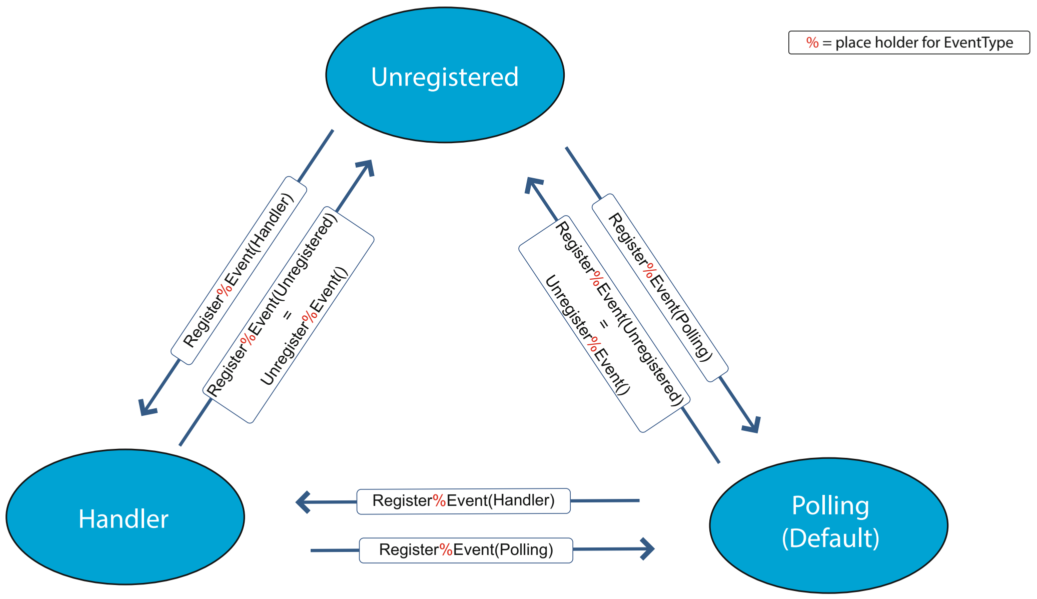
Eventmode: Polling

The Polling Eventmode allows events that occur to be notified per function call of the corresponding Get function. The user has to activate the Get function periodically to fetch multiple events.

Eventmode: Handler

The Handler Eventmode allows events that occur to be notified per callback function. This function is activated anew by Baumer GAPI for each event that occurs.

The following diagram shows the potential mode switching between Eventmodes.


% = place holder for EventType
Unregistered
Handler
Polling (Default)
Register%Event(Handler)
Register%Event(Unregistered) = Unregister%Event()
Register%Event(Polling)
Register%Event(Unregistered) = Unregister%Event()
Register%Event(Handler)
Register%Event(Polling)

3.1.2.2 InterfaceEventControl

InterfaceEventControl Inherits from: EventControl Parent class for: Interface	InterfaceEventControl provides the Plug'n'Play event.

Functions of the InterfaceEventControl class

`BGAPI2::InterfaceEventControl:: ...`	
`RegisterPnPEvent(EventMode eventMode);`	Enables P'n'P event handling.
`UnregisterPnPEvent();`	Disables P'n'P event handling.
`GetPnPEvent(PnPEvent * pPnPEvent, bo_using64 iTimeout);`	Polling function for P'n'P events.
`CancelGetPnPEvent();`	Cancels the operation currently waiting on the GetPnPEvent function.

3.1.2.3 DeviceEventControl

DeviceEventControl Inherits from: Eventcontrol Parent class for: Device	This class provides access to standard camera events. See Standard Feature Naming Convention (SFNC).

Functions of the Device EventControl class

`BGAPI2::DeviceEventControl:: ...`	
`RegisterDeviceEvent(Event eventMode);`	Enables standard compliant device events, specified within the xml description file.
`UnregisterDeviceEvent();`	Disables standard compliant device events, specified within the xml description file.
`GetDeviceEvent(DeviceEvent * pDeviceEvent, bo_uint64 iTimeout);`	Polling function for event devices.
`CancelGetDeviceEvent();`	This function cancels the operation currently waiting on the GetDeviceEvent function.

3.1.2.4 DataStreamEventControl

DataStreamEventControl <u>Inherits from:</u> EventControl <u>Parent class for:</u> DataStream	DataStreamEventControl provides new Buffer-event, which is used for image notification.

Functions of the DataStream EventControl class

`BGAPI2::DataStreamEventControl:: ...`	
`RegisterNewBufferEvent();`	Enables standard compliant buffer events, specified within the xml description file.
`UnregisterNewBuffer-Event();`	Disables standard compliant buffer events, specified within the xml description file.
`GetFilledBuffer(bo_uint64 iTimeout);`	Polling with timeout.
`CancelGetFilledBuffer();`	Cancellation routine for GetfilledBuffer.

3.2 Main Classes

The main classes represent the fundamental logical and physical components of the image processing system.

The main classes are: *System*, *Interface*, *Device*, *DataStream* and *Buffer*.

3.2.1 System

System

Inherits from:
EventControl,
INode

The *System* is the abstraction of the Producer and the Producer mentioned previously.

Functions of the System class

`BGAPI2::System:: ...`	
`System(String file-path);`	Sets the path to the Producer file.
`Open();`	Open a *System* (Producer file).
`Close();`	Close the *System* (Producer file).
`IsOpen();`	This function delivers true if the data stream is opened.
`GetInterfaces();`	Returns a list of all available interfaces *(InterfaceList).*
`GetID();`	Returns the unique string identifier of the *System* that is used in the *SystemList.*
`GetVendor();`	Returns the name of the Producer vendor.
`GetModel();`	Returns the model name of the Producer.
`GetVersion();`	Returns the version number of *System* file.
`GetTLType();`	Returns the type of system (GEV, U3V, ...) .
`GetFileName();`	Returns the name of *System* file.
`GetPathName();`	Retrieves the filepath.
`GetDisplayName();`	Returns an appropriate name of the *Interface* for display only.

3.2.2 Interface

Interface

Inherits from:
InterfaceEvent-Control, INode

The *Interface* class represents a physical interface, e.g. GEV network adapter or a logical interface, such as USB.

Functions of the Interface class

BGAPI2::Interface:: ...	
`Open();`	Opens an *Interface.*
`Close();`	Closes the *Interface.*
`IsOpen();`	This function delivers true if the data stream is opened.
`GetParent();`	This function delivers the superordinate Device object.
`GetDevices();`	Returns a list of available devices for this interface *(DeviceL-ist).*
`GetID();`	Returns the unique string ID that is used in *InterfaceList.*
`GetDisplayName();`	Returns an appropriate name of the *Interface* for display only.
`GetTLType();`	Returns interface type (GEV, U3V, ...).

3.2.3 Device

Device

Inherits from:
DeviceEvent-Control, INode

The *Device* main class is used to retrieve information (e.g. model, manufacturer, access modes) about the device (camera) and also to control the device.

Functions of the Device class

BGAPI2::Device:: ...	
`OpenReadOnly();`	Opens as read only, no parameters can be changed.
`Open();`	Opens a device with read and write access.
`OpenExclusive();`	Opens a device exclusively with read and write access, the device is blocked for other software tools.
`Close();`	Closes the device.
`IsOpen();`	Opens a device with read only access, no parameters can be changed.
`GetParent();`	This function delivers the superordinate Device object.
`StartStacking (bo_bool bReplaceMode);`	Start command for parameter stacking (parameters will be queued). The parameter control whether all accesses to a register will be transferred to the device or only the last one.

`WriteStack();`	Single write command to transmit all stacked parameters simultaneously.
`GetDataStreams();`	Returns a list of available *DataStreams.*
`GetID();`	Returns a device's unique string ID (such as MAC address on GEV).
`GetVendor();`	Returns the vendor of the camera.
`GetModel();`	Returns the camera model (e.g. MXGC20c).
`GetTLType();`	Returns the device type of the *System* (such as GEV, U3V, ...).
`GetDisplayName();`	Returns an appropriate name for display only.
`GetSerialNumber();`	Returns the camera's serial number

`GetAccessStatus();`	`Returns whether the device is / can be opened:`
	`Open();`
	`OpenReadOnly();`
	`OpenExclusive();`

`GetPayloadSize();`	Returns the payload size.
`GetRemoteNode (String name);`	Returns a pointer to a specified object of the connected device.
`GetRemoteNode-Tree();`	Returns the tree view showing all nodes of the XML description file of the connected device.
`GetRemoteNode-List();`	Returns a one-dimensional list of the nodes of the XML description file of the connected devices.
`GetRemoteConfigura-tionFile();`	Returns the XML description file of the connected device.

3.2.4 DataStream

DataStream

Inherits from:
DataStreamEvent-Control, INode

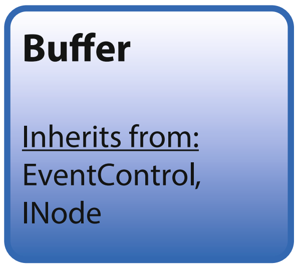
This class represents a data stream from the camera to the PC. It controls the data transfer and is responsible for buffer handling.

Functions of the DataStream class

`BGAPI2::DataStream:: ...`	
`Open();`	Opens a data stream.
`Close();`	Closes the data stream.
`IsOpen();`	This function delivers true if the data stream is opened.
`GetParent();`	This function delivers the superordinate Device object.
`GetBufferList();`	Returns a list of previously created buffers (*BufferList*).
`GetID();`	Returns the unique string identifier of the *data stream.*
`GetPayloadSize();`	Returns the payload size.
`GetDefinesPayload-Size();`	Checks whether the data stream can provide the payload size.
`GetIsGrabbing();`	Returns whether the Data Stream has already been started.
`GetTLType();`	Returns the type of the data stream.

`StartAcquisition-Continuous();`	This function starts the *DataStream* object.
`StartAcquisition (bo_uint64 iNumToAcquire);`	Starts acquisition of a specified number of images.
`StopAcquisition();`	Stops image acquisition.
`AbortAcquisition();`	Aborts image acquisition immediately.

3.2.5 Buffer



The Buffer module encapsulates a single memory buffer. Its purpose is to act as the target for image acquisition. Buffers can be allocated automatically by the producer or by the user.

This class enables data access to the memory. It also contains information about the received data (e.g. image size, pixel format).

Functions of the Buffer class

`BGAPI2::Buffer:: ...`	
`Buffer();`	Creates a buffer object.
`Buffer( void * pUserObj );`	Creates a buffer object including user-specific data.
`Buffer( void *pUserBuffer, bo_uint64 uUserBufferSize, void *pUserObj);`	Creates a buffer object using user-allocated memory.
`GetParent();`	This function delivers the superordinate Device object.
`GetID();`	Returns the unique ID of a buffer (e.g. Buffer_01f94fb), not the memory address of the buffer.
`QueueBuffer();`	Allocates the buffer to the input queue of a *DataStream.*
`GetMemPtr();`	Returns the memory address of the buffer.
`GetMemSize();`	Returns the memory size of the buffer.
`GetUserObj();`	Returns the user specific-data for the buffer.
`GetTimestamp();`	Returns the buffer's timestamp.
`GetNewData();`	Returns "true" if the buffer is filled until GetFilledBuffer is performed.
`GetIsQueued();`	Returns "true" provided the buffer is allocated to the input queue of a *DataStream.*
`GetIsAcquiring();`	Returns "true", if buffer is being filled.
`GetIsIncomplete();`	Returns "true" for defective image data.
`GetTLType();`	Returns the type of buffer (GEV, U3V, ...).
`GetSizeFilled();`	Returns the current fill level of the buffer [bytes].
`GetWidth();`	Returns the image width.
`GetHeight();`	Returns the image height.
`GetXOffset();`	Returns the offset on the x axis.
`GetYOffset();`	Returns the offset on the y axis.
`GetFrameID();`	Returns a hardware generated counter (1...65535).
`GetImagePresent();`	Returns the availability of the image within the buffer.

`GetImageOffset();`	Returns the position of the first byte of image data within the buffer.
`GetPayloadType();`	Returns whether the payload is chunk or image data .
`GetPixelFormat();`	Returns the pixel format of the image.
`GetDeliveredImage-Height();`	Returns the number of lines of a transmitted image.
`GetDelivered-Chunk-PayloadSize();`	Returns the delivered chunk payload size.
`GetContainsChunk();`	Returns true, if the Buffer object includes chunk data.
`GetChunkLayoutID();`	Returns the ID of the layout.
`GetChunkNodeTree();`	Returns the tree view showing all chunk nodes.

3.3 List Classes

These classes enable discovery and listing of the main objects.

The list classes are: *SystemList*, *InterfaceList*, *DeviceList*, *DataStreamList* and *BufferList*.

In any list class, there are three different ways to handle the objects within a list. The first of these is the iterator, which can be used for iterating through the lists using begin and end functions.

The second approach is access via the subscript operator. The third method for object handling is to use the find function.

Each list entry consists of an ID that is used as key value and a pointer to an object within the main classes.

`These functions are available for all List Classes.`

`BGAPI2::[respective List Class]:: ...`	
`iterator`	This is the iterator class.
`begin();`	Retrieves the iterator that points to the first list entry.
`end();`	Retrieves the iterator that points to the last list entry +1.
`Refresh();`	Refresh list.
`size();`	Returns the number of located list entries.
`operator[](const String& val)`	This is the subscript operator for accessing a freely selectable point in a list.
`find(const String& _keyval);`	Returns the iterator for the required list entry that is defined by the unique ...
`clear();`	Clears the list.

3.3.1 SystemList

SystemList	This class is used to list objects of the *System* class. **Notice** This object is a singleton. It is not possible to create other objects of this class. It always accesses the same object. This means that a release function is required.

Functions of the SystemList class

`BGAPI2::SystemList:: ...`	
`GetInstance();`	Creates the *SystemList* instance.
`ReleaseInstance();`	Releases the *SystemList* instance.
`CreateInstanceFromPath (String producerpath);`	Forces only one producer to be used for the listing.
`Add(System * pSystem);`	Adds a producer to the list.
`Refresh();`	Refreshes the *SystemList.*

3.3.2 InterfaceList

InterfaceList	This class is used to list objects of the *Interface* class.

3.3.3 DeviceList

DeviceList	This class is used to list objects of the *Device* class.

3.3.4 DataStreamList

DataStreamList	This class is used to list objects of the *DataStream* class.

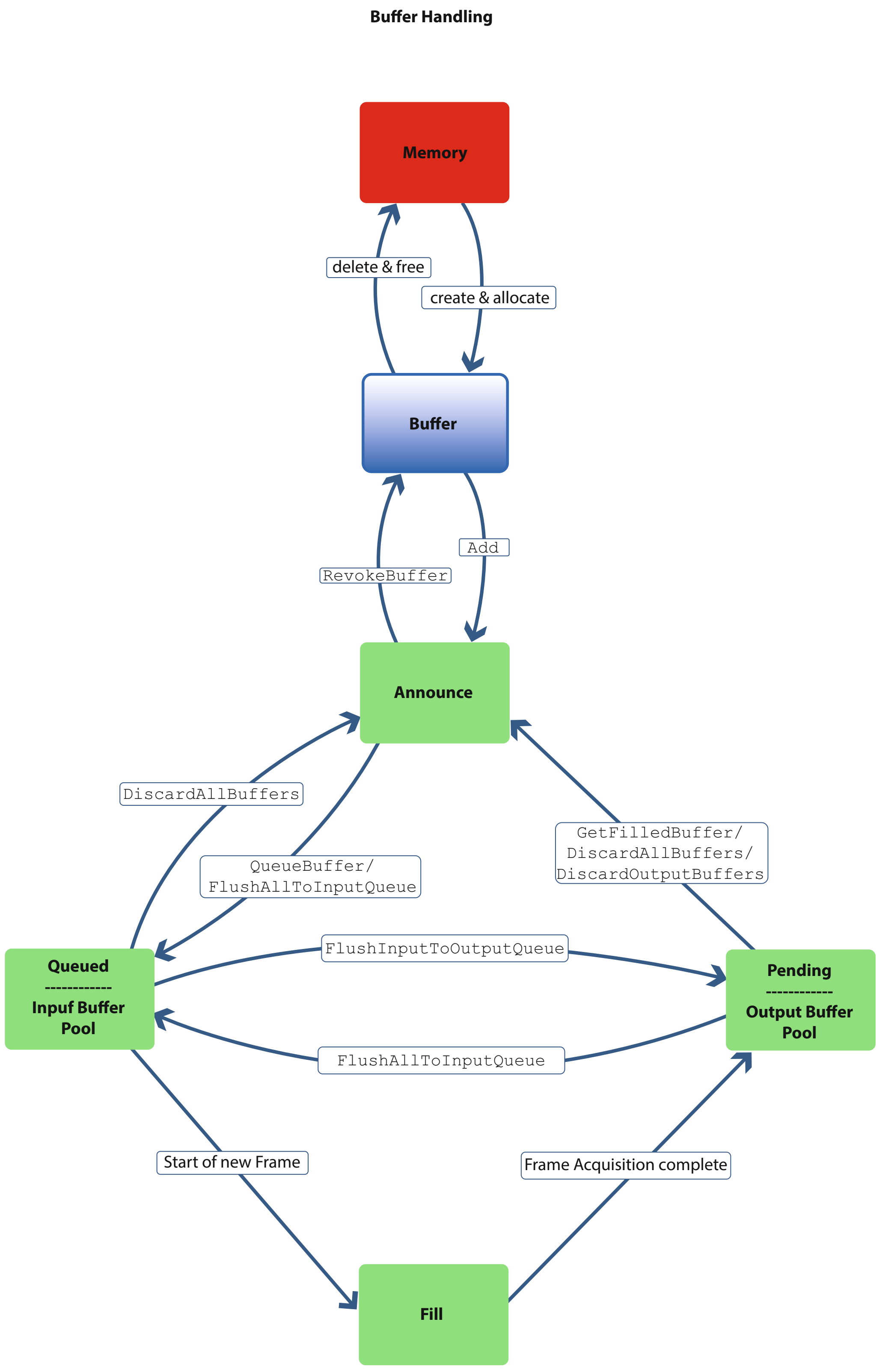
3.3.5 BufferList

BufferList <u>Inherits from:</u> INode	This class is used to list objects of the *Buffer* class. **Special feature of the BufferList:** The programmer can add buffers to the *BufferList* by using this list class. Therefore there is no refresh function.

Functions of the BufferList class

`BGAPI2::BufferList:: ...`	
`Add(Buffer *pBuffer);`	Adds a new buffer to the BufferList.
`RevokeBuffer(Buffer *pBuffer);`	Revokes the buffer from the BufferList.
`FlushInputToOutput-Queue();`	Flushes all buffers from the input queue to the output queue.
`FlushAllToInputQueue();`	Flushes all available buffers to the input queue.
`FlushUnqueuedToInput-Queue();`	Flushes all announced buffers to the input queue.
`DiscardOutputBuffers();`	Discards all buffers from the output queue to announced status.
`DiscardAllBuffers();`	Discards all buffers from any queue to announced status.
`GetDeliveredCount();`	Returns the number of buffers that are retrieved from the output queue via GetFilledBuffer.
`GetUnderrunCount();`	Returns the number of buffer underruns.
`GetAnnouncedCount();`	Returns the number of announced buffers.
`GetQueuedCount();`	Returns the number of queued buffers (input queue).
`GetAwaitDeliveryCount();`	Returns the number of queued buffers (output queue).

The buffer management principle is illustrated in the following diagram.



Buffer Handling
Memory
delete & free
create & allocate
Buffer
RevokeBuffer
Add
Announce
DiscardAllBuffers
QueueBuffer/ FlushAllToInputQueue
GetFilledBuffer/ DiscardAllBuffers/ DiscardOutputBuffers
Queued ------------ Inpuf Buffer Pool
FlushInputToOutputQueue
FlushAllToInputQueue
Pending ------------ Output Buffer Pool
Start of new Frame
Frame Acquisition complete
Fill

3.4 Additional Classes

As well as the classes mentioned above, the Baumer GAPI also contains three additional classes.

The first two of these (*Image* & *ImageProcessor*) are used for image handling and manipulation, and are encapsulated in the bgapi2_img.dll. The third class (*Trace*) provides tracing functionality.

3.4.1 Image

Image	This class provides the ability to transform images.

Functions of the Image class

`BGAPI2::Image:: ...`	
`GetWidth();`	Returns the width of the image.
`GetHeight();`	Returns the height of the image.
`GetPixelformat();`	Returns the pixel format of the image.
`GetBuffer();`	Returns the image buffer.
`GetTransformBufferLength(String sPixelFormat);`	Returns the required buffer length of the image following transformation to a specified pixel format.
`GetHistogram(bo_tHistRecords Histogram);`	Returns the histogram array of the image.
`Init(bo_uint width, bo_uint height, String sPixelFormat, void* pBuffer, bo_uint64 uBufferSize);`	This function reinitialise an Image object.
`Release();`	Releases the occupied resources (excluding the user-allocated buffer).

3.4.2 Image Processor

Image Processor <u>Inherits from:</u> INode	This class creates image objects.

Functions of the Image Processor class

`BGAPI2::ImageProcessor:: ...`	
`new BGAPI2::ImageProcessor();`	Creates a image processor. Multiple image processors possible.
`delete`	Delete the image processor.
`GetVersion();`	Returns the image processor version.
`CreateImage(bo_uint width, bo_uint height, String pixelformat, void* pBuffer, bo_uint64 uBufferSize);`	Creates a Baumer GAPI Image object based on a filled buffer.
`CreateTransformedImage(Image* pInputImage, const char* szDestinationPixelformat);`	Creates an image object and performs a pixel format transformation.
`TransformImageToBuffer(Image* pInputImage, const char* szDestinationPixelformat, void* pBuffer, bo_uint64 uBufferSize);`	This function transformes the pixel format of the Image object and writes the data into the passed destination buffer.

3.4.3 Trace

Trace	Baumer GAPI2 Trace gives you the option to monitor the program flow and detect errors. All functions in this class are static.

Functions of the Trace class

`BGAPI2::Trace:: ...`	
`Enable( bo_bool benable );`	Enables or disables tracing.
`ActivateOutputToFile( bo_bool bactive, String tracefilename );`	Provides the option to print trace outputs to the specified file.
`ActivateOutputToDebugger( bo_bool vactive );`	Provides the option to print trace outputs to the debugger.
`ActivateMaskError( bool vactive );`	Activates error tracing
`ActivateMaskWarning( bo_bool bactive );`	Activates warning tracing.
`ActivateMaskInformation( bo_bool bactive );`	Activates information tracing.
`ActivateOutputOptionTimestamp ( bo_bool bactive );`	Includes the timestamp with the trace output.
`ActivateOutputOptionTimestampDiff ( bo_bool bactive);`	Includes the time lag from the last to the current trace output.
`ActivateOutputOptionPrefix ( bo_bool bactive );`	Includes the module that created the trace output (such as producer, consumer ...)

3.5 IException

IException	This class is responsible for the exception handling and represents the parent class of all exception classes.

IException Functions

BGAPI2::IException:: ...	
`GetErrorDescription();`	Returns a description of the error.
`GetFunctionName();`	Returns the name of the function where the error occurred.
`GetType();`	Returns the type of error.

Exception Class	Description
`NotInitializedException`	The requested object is not initialized.
`NotImplementedException`	The requested function/feature is not implemented.
`ResourceInUseException`	The requested object is already in use.
`AccessDeniedException`	The requested operation is not allowed.
`InvalidHandleException`	The given handle does not support the operation.
`ObjectInvalididException`	The referenced object is not a valid object of BGAPI2.
`NoDataException`	The function has no data to work on.
`InvalidParameterException`	One of the parameters given was invalid or out of range and none of the error codes above fit.
`LowLevelException`	Exception thrown by deeper software layers such as Producer.
`AbortException`	An operation has been aborted before it could be completed.
`InvalidBufferException`	No buffer announced or one or more buffers with invalid buffer size.
`NotAvailableException`	Resource or information is not available at the given time in the current state.
`ErrorException`	General purpose exception

4. Programming Basics in Baumer GAPI2

The following settings for the various operating systems must be made before starting programming.

4.1 Microsoft® Windows®

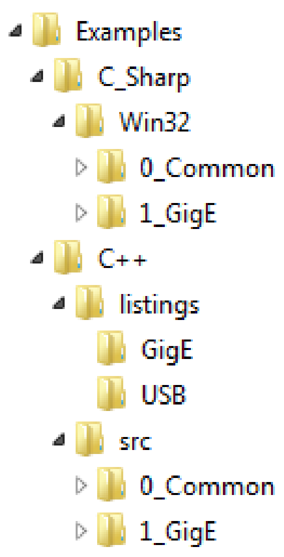
All required files were copied to your system during the installation process.

Default path Windows® 32 bit / 64 bit:

```
C:\Program Files\Baumer\Baumer GAPI SDK
```

Below, these folders are referred to as `<BGAPI2 SDK>`

Folder structure for the examples (Windows® / Linux)



```
Examples
  C_Sharp
    Win32
      0_Common
      1_GigE
  C++
    listings
      GigE
      USB
    src
      0_Common
      1_GigE
```

Notice

Location of the Runtime Modules

All bgapi dll´s must be provided within the same directory as the calling application. This accounts for the fact that the bgapi2_genicam.dll searches for its modules within the directory in which it is stored.

The bgapi2_genicam.dll also searches for producers in the GENTL PATH.

Please ensure you refer to the correct directory (Win32 = 32bit / x64 = 64 bit).

This is particularly important when using Microsoft® Visual Studio – the working directory needs to be adjusted correctly.

4.2 Migration of existing Windows® projects to Linux

Notice

If existing Windows® projects are to be migrated to Linux, the preprocessor switch `D_GNULINUX` must be set for the gcc.

Details from the SDK examples:

```
...
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -D_GNULINUX")
...
```

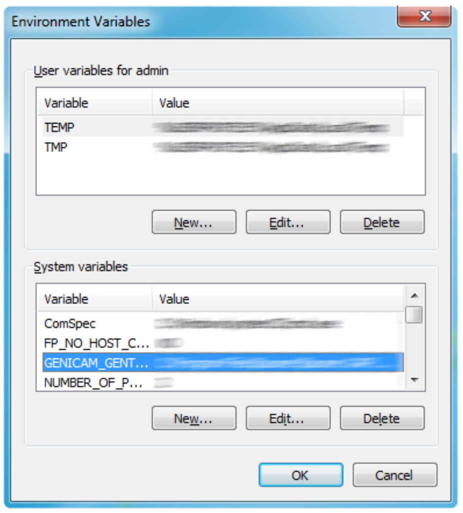
4.3 Setting System variables

4.3.1 Microsoft® Windows®

Notice

The path to the system variable is already set during the installation process of the SDK. Follow the instructions to change the path to the system variables (*bgapi2_xxx.cti).*

1. Click *Start,* then choose *Computer.*



Environment Variables

User variables for admin

Variable	Value
TEMP	[redacted]
TMP	[redacted]

New... Edit... Delete

System variables

Variable	Value
ComSpec	[redacted]
FP_NO_HOST_C...	[redacted]
GENICAM_GENT...	[redacted]
NUMBER_OF_P...	[redacted]

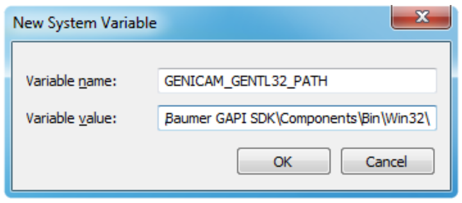
New... Edit... Delete

OK Cancel

2. Click *Properties.*
3. Select the *Advanced* tab.
4. Click *Environment Variables.*
5. Check that the entry GENICAM_ GENTL32_PATH or GENICAM_ GENTL64_PATH is present in the *System variables*.

   If the entry is not present, continue with ***step 6***.

   If the entry is present, continue with ***step 8***.



New System Variable

Variable name: GENICAM_GENTL32_PATH

Variable value: Baumer GAPI SDK\Components\Bin\Win32\

OK Cancel

6. Click *New* within System variables.
7. Adjust the following.

   Variable name:

   32-bit systems

   GENICAM_GENTL32_PATH

   64-bit systems

   GENICAM_GENTL64_PATH

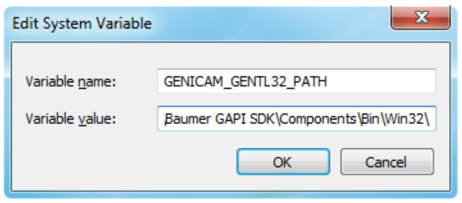
   Variable value:

   32-bit systems

   ```
   <BGAPI2 SDK>\Components\Bin\Win32
   ```

   64-bit systems

   ```
   <BGAPI2 SDK>\Components\Bin\x64
   ```



Edit System Variable

Variable name: GENICAM_GENTL32_PATH

Variable value: Baumer GAPI SDK\Components\Bin\Win32\

OK Cancel

8. Select the GENICAM_GENTL32_ PATH or GENICAM_GENTL64_ PATH variable and click *Edit*.

   Add the Baumer path to the *Variable value*. Separate it from existing entries with a semicolon.

   Variable value:

   32-bit systems

   ```
   <BGAPI2 SDK>\Components\Bin\Win32
   ```

   64-bit systems

   ```
   <BGAPI2 SDK>\Components\Bin\x64
   ```

4.3.2 Linux

If you are using Linux and need to add the file path to the system variable GENICAM_GENTL32_PATH, the following line must be added in the file ~/.bashrc.

32-bit systems

```
export GENICAM_GENTL32_PATH=/usr/local/lib/baumer
```

64-bit systems

```
export GENICAM_GENTL64_PATH=/usr/local/lib/baumer
```

4.4 Implementation

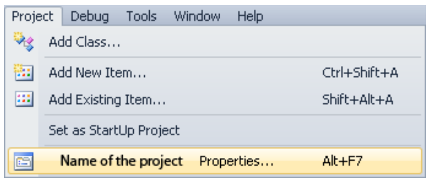
You will need to include different header files and libraries depending on the programming language and operating system. See the respective precautions for your operating system and programming language.

4.4.1 Implementation in C++ (Microsoft® Windows®)

4.4.1.1 Preparations

Create a new project by generating a *.cpp file, then open the project properties from the context menu. Some required information such as the header files directory will need to be set here.

Set the following values in "<Name of the project> Properties":


Project
Debug
Tools
Window
Help
Add Class...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Set as StartUp Project
Name of the project Properties... Alt+F7

34- / 64-bit systems

For 64-bit system, you must reference on the x64 folder instead on the Win32 folder.

- "C++" → "General": "Additional Included Directories":

```
<BGAPI SDK>\Dev\C++\Inc
```

- "Linker" → "General": "Additional Library Directories":

```
<BGAPI SDK>\Dev\C++\Lib\Win32
```

- "Linker" → "Input": "Additional Dependencies":

```
bgapi2_genicam.lib
```

- "Build Events" → "Post-Build-Event": "Command Line":

absolute path:

```
copy "<BGAPI SDK>\Components\Bin\Win32"\*.* .\
```

or the relative path (used in the examples):

```
copy ..\..\..\..\..\Bin\Win32\*.* .\
```

The API is based on the GenICam™ GenAPI reference implementation.

More information can be found on: https://www.emva.org/standards-technology/genicam/

The required files are located in the following folder:

32 Bit:

```
<BGAPI SDK>\Components\Bin\Win32
```

64 Bit:

```
<BGAPI SDK>\Components\Bin\x64
```

Notice

To run the programs on a computer where Baumer GAPI is not installed, these files are absolutely necessary.

The standard parser of GenICam GenAPI reference implemetation includes the following DLLs:

```
GCBase_MD_VC120_v3_0_baumer.dll
GenApi_MD_VC120_v3_0_baumer.dll
Log_MD_VC120_v3_0_baumer.dll
log4cpp_MD_VC120_v3_0_baumer.dll
MathParser_MD_VC120_v3_0_baumer.dll
msvcp120.dll (Microsoft Visual Studio 2013 VC120)
msvcr120.dll (Microsoft Visual Studio 2013 VC120)
NodeMapData_MD_VC120_v3_0_baumer.dll
XmlParser_MD_VC120_v3_0_baumer.dll
```

Baumer GenICam DLL:

```
bgapi2_genicam.dll
```

Baumer GenTL Producer GigE:

```
bgapi2_gige.cti
bopfdrvctl.dll (PacketFilterDriver)
bsysgige.xml
```

Baumer GenTL Producer USB3:

```
bgapi2_usb.cti
bsysusb.xml
```

Baumer ImageProcessor (Bayer to BGR/RGB conversion):

```
bgapi2_img.dll
```

Support for C++ Examples 016/017 Flatfield Correction:

```
bgapi2_ext_sc.dll
```

4.4.1.2 Header Files

```
<BGAPI2 SDK>\Components\Dev\C++\Inc\
```

Header files:
Files required for the compiler and containing forwarded declarations for subroutines, variables and other identifiers.

`bgapi2_genicam\`

`bgapi2_genicam.hpp`	Interface definition for Baumer GAPI, including classes, functions and function pointer for callbacks.
`bgapi2_def.h`	Definitions for Baumer GAPI, including defines, enumerations and typedefs.
`bgapi2_featurenames.h`	Summary of the features from the GenICam Standard Features Naming Convention.

`bgapi2_ext_sc\`

`bgapi2_ext_sc.h`	Summary of the features that are used in programming examples 16 and 17 (Flatfield Correction).

Enter the following code to include the necessary header file for C++:

```
#include "bgapi2_genicam.hpp"
#include "bgapi2_ext_sc\bgapi2_ext_sc.h"
```

4.4.1.3 Libraries

Library:
Collection of recurrent routines.

Notice
The files required for 32 bit and 64 bit operating systems are entirely different. Please ensure you include the correct folder. Including the incorrect files may cause errors!

Default path Windows® 32bit:

```
<BGAPI2 SDK>\Components\Dev\C++\Lib\Win32
```

Default path Windows® 64bit:

```
<BGAPI2 SDK>\Components\Dev\C++\Lib\x64
```

`bgapi2_genicam.lib`	This is where all function names and their implementations are stored.
`bgapi2_ext_sc.lib`	This is where all function names and their implementations for the programming examples 16 and 17 (Flatfield Correction) are stored.

4.4.1.4 Requirements for Cmake

With Baumer GAPI up v2.2 we introduced CMake to generate the project files for C++ on the target system (your PC) according to the used compiler, like VS2005, VS2008, VS2010 or VS2012.

To see just the example source codes, please check

```
../Components/Examples?/C++/src/0_Common
```

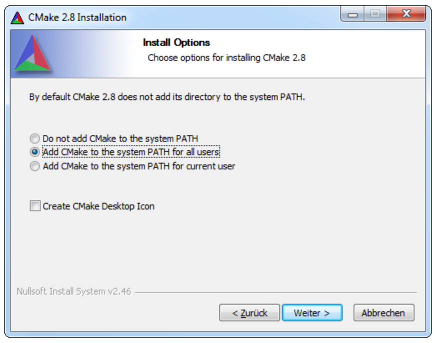
To have a look on the sample console output of each example refer to:

```
../Components/Examples?/C++/listings/USB
```

Instructions to get the project files generated by using CMake

• Install CMake from the Baumer GAPI download file first, if you want to check C++ code (for C# code check the VS2010 projects directly).

```
../Tools/Win32/CMake/cmake-2.8.11-win32-x86.exe
```



CMake 2.8 Installation

Install Options

Choose options for installing CMake 2.8

By default CMake 2.8 does not add its directory to the system PATH.

- ( ) Do not add CMake to the system PATH
- (●) Add CMake to the system PATH for all users
- ( ) Add CMake to the system PATH for current user

☐ Create CMake Desktop Icon

Nullsoft Install System v2.46

< Zurück | Weiter > | Abbrechen

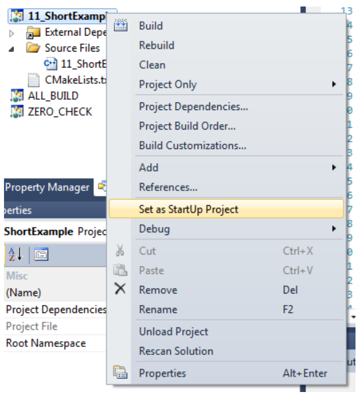
Maybe you need a reboot of the PC here.

• Run the install_example_win.bat to create the project files for your visual studio version (VS2005, VS2008, VS2010 or VS2012) and your system (32-bit or 64-bit).

```
../Components/Examples?/C++/install_example_win.bat
```

• Start the VS2010 solution and select the example you want to run as "Set as StartUpProject?" (e.g. 11_ShortExample).

```
../Components/Examples?/C++/build_vs10_c++_WIN_32/Baumer_GAPI_SDK_Examples_C++.sln
```



4.4.1.5 Using Namespace

To use the Baumer GAPI namespace for C++ by default, please enter the following code:

```
using namespace BGAPI2;
```

4.4.2 Implementation in C++ (Linux)

4.4.2.1 Preparations

Installation location of the examples.

```
/usr/local/src/baumer/sdk_example/C++

    CMakeLists.txt
   install_example_linux.sh
   listings/
   src/
```

4.4.2.2 Header Files

```
/usr/local/src/baumer/inc/
```

`bgapi2_genicam.hpp`	Interface definition for Baumer GAPI, including classes, functions and function pointer for callbacks.
`bgapi2_def.h`	Definitions for Baumer GAPI, including defines, enumerations and typedefs.
`bgapi2_featurenames.h`	Summary of the features from the GenICam Standard Features Naming Convention.

4.4.2.3 Libraries

Notice
The names of the files are the same for 32 bit and 64 bit.

```
/usr/local/lib/baumer/

    libbgapi2_img.so
    libbgapi2_genicam.so
    libsharedlibs.so
    libevisionlib.so
    libbgapi2_gige.cti
    liblibtiff.a
```

4.4.2.4 Requirements for CMake

With Baumer GAPI up v2.2 we introduced CMake to generate the project files for C++ on the target system (your PC) according to the used compiler, like GCC (Compler) in Linux.

Install the following packages. Internet access is required to execute this command.

Input via the console [Terminal Program]:

```
apt-get install cmake build-essential
```

- Query the installed version of CMake

Notice
Version 2.8 or later required!

Input via the console [Terminal Program]:

```
cmake --version
```

- Create Build Directories (preparing to compile / linking):

Input via the console [Terminal Program]:

```
/usr/local/src/baumer/sdk_example/C++# ./install_example_linux.sh
```

The following directories are created:

```
build_linux_debug

build_linux_release
```

- Change the directory.

Input via the console [Terminal Program]:

```
cd build_linux_debug
```

- Run `make` to create the binaries.

Input via the console [Terminal Program]:

```
make
```

- The final binaries are located under e.g.:

```
C++/001_ImageCaptureMode_Polling/001_ImageCaptureMode_Polling#
001_ImageCaptureMode_Polling
```

- Execution of an example file

Input via the console [Terminal Program]:

```
./001_ImageCaptureMode_Polling
```

4.4.2.5 Import the examples into Eclipse IDE

In case of IDE eclipse, do following steps:

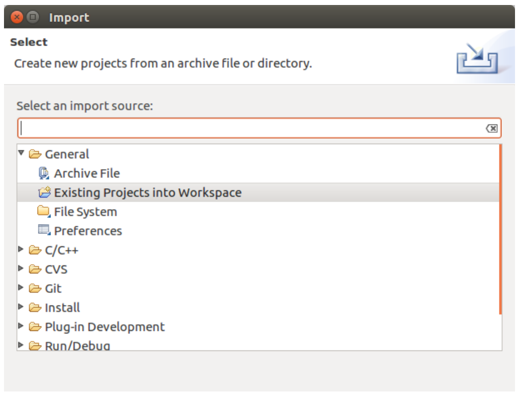
1. Change directory to `cd /usr/local/src/baumer/sdk_example/C++`

2. Execute script `./install_example_linux.sh`

3. Open eclipse.

4. Navigate to *File→Import*

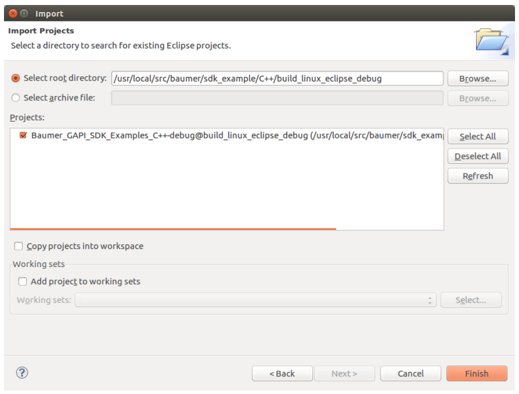
5. Navigate to *General→Existing Projects into Workspace*



6. Select root directory to create build directory from step 2

7. Browse to: `/usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug`

8. Accept with Finish button.



Select root directory: /usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug

Select archive file:

Projects:

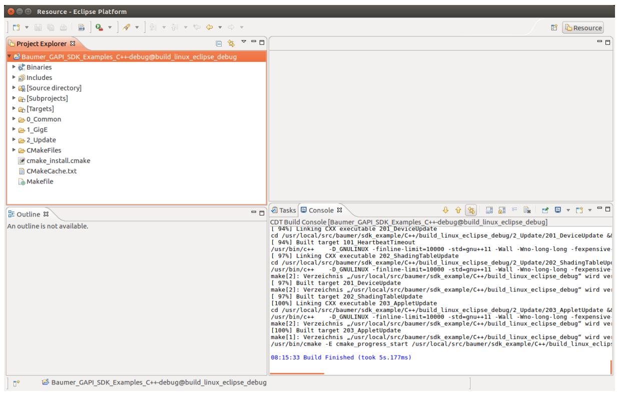
☑ Baumer_GAPI_SDK_Examples_C++-debug@build_linux_eclipse_debug (/usr/local/src/baumer/sdk_exam

☐ Copy projects into workspace

☐ Add project to working sets

Working sets:

9. Now you can see the project workspace.



An outline is not available.

CDT Build Console [Baumer_GAPI_SDK_Examples_C++-debug@build_linux_eclipse_debug]

```
[ 94%] Linking CXX executable 201_DeviceUpdate
cd /usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug/2_Update/201_DeviceUpdate &&
[ 94%] Built target 101_HeartbeatTimeout
/usr/bin/c++    -D_GNULINUX -finline-limit=10000 -std=gnu++11 -Wall -Wno-long-long -fexpensive-
[ 97%] Linking CXX executable 202_ShadingTableUpdate
cd /usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug/2_Update/202_ShadingTableUpd
/usr/bin/c++    -D_GNULINUX -finline-limit=10000 -std=gnu++11 -Wall -Wno-long-long -fexpensive-
make[2]: Verzeichnis „/usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug“ wird ve
[ 97%] Built target 201_DeviceUpdate
make[2]: Verzeichnis „/usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug“ wird ve
[ 97%] Built target 202_ShadingTableUpdate
[100%] Linking CXX executable 203_AppletUpdate
cd /usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug/2_Update/203_AppletUpdate &&
/usr/bin/c++    -D_GNULINUX -finline-limit=10000 -std=gnu++11 -Wall -Wno-long-long -fexpensive-
make[2]: Verzeichnis „/usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug“ wird ve
[100%] Built target 203_AppletUpdate
make[1]: Verzeichnis „/usr/local/src/baumer/sdk_example/C++/build_linux_eclipse_debug“ wird ve
/usr/bin/cmake -E cmake_progress_start /usr/local/src/baumer/sdk_example/C++/build_linux_eclips

08:15:33 Build Finished (took 5s.177ms)
```

Baumer_GAPI_SDK_Examples_C++-debug@build_linux_eclipse_debug

10. Navigate to Project and select Build Project, all example apps are build.

Notice

Please read eclipse documentation for setting different customer configuration like debug settings and so on.

4.4.2.6 Using Namespace

To use the Baumer GAPI namespace for C++ by default, please enter the following code:

```
using namespace BGAPI2;
```

or add BGAPI2 to the head of all calls, e.g.:

```
BGAPI2::Interface
BGAPI2::Device
BGAPI2::Image
BGAPI2::Exception
```

4.4.3 Implementation in C# (Windows®)

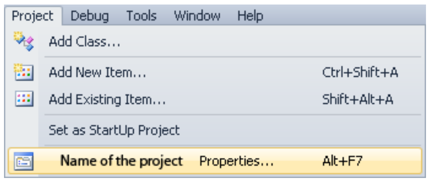
Notice

CMake is not required for C#.

4.4.3.1 Preparations

The first method is applied for the examples provided.

Set the following values under "**Name of the project** Properties...":


Project
Debug
Tools
Window
Help
Add Class...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
Set as StartUp Project
Name of the project Properties... Alt+F7

- "Build Events" → "Post-Build-Event": "Command Line":

absolute path:

```
copy "<BGAPI SDK>\Components\Bin\Win32"\*.* .\
```

or relative path:

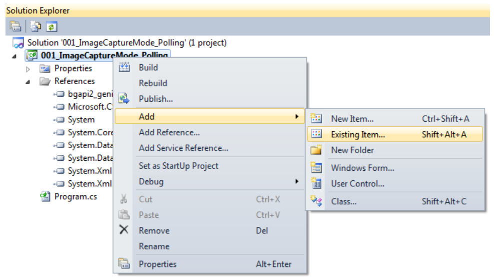
```
relative path: copy "..\..\..\..\..\..\..\..\Bin\Win32"\*.* .\
```

Alternatively, you can use this method:

Build Events (C#):

The reason for the `copy` *command is that the compiled .exe must store all required bgapi2.dll files in the same directory.*

Open the properties via the context menu and select:



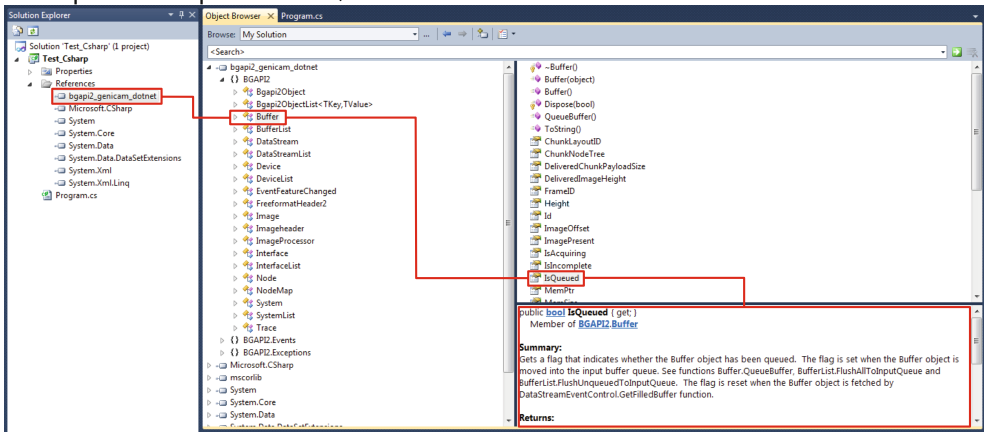
"Add → Existing Item...".

Add the respective DLL files.

Notice

The folder of the `bgapi2_genicam_dotnet.dll` contains the documentation file `bgapi2_genicam_dotnet.xml` as well. After including the .dll as reference the documentation is available to the user within the Visual Studio Object Browser.

Example: Description for *IsQueued*



public bool **IsQueued** { get; }
Member of BGAPI2.Buffer

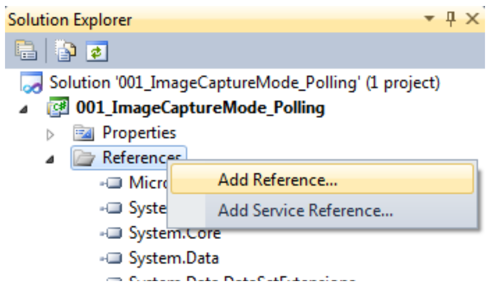
Summary:
Gets a flag that indicates whether the Buffer object has been queued. The flag is set when the Buffer object is moved into the input buffer queue. See functions Buffer.QueueBuffer, BufferList.FlushAllToInputQueue and BufferList.FlushUnqueuedToInputQueue. The flag is reset when the Buffer object is fetched by DataStreamEventControl.GetFilledBuffer function.

Returns:

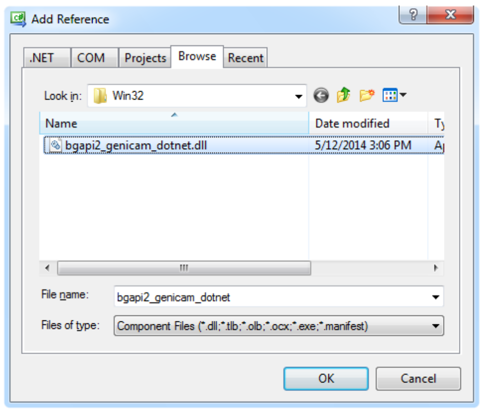
4.4.3.2 .NET Component

Create a new project, open its properties via the context menu and select:

"Add Reference...".



Integrate the respective bgapi2_genicam_ dotnet.dll.



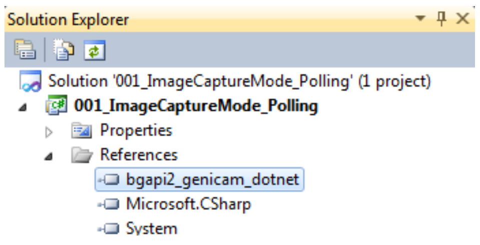
Look in: Win32

Name	Date modified
bgapi2_genicam_dotnet.dll	5/12/2014 3:06 PM

File name: bgapi2_genicam_dotnet

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

Now you can see the integrated bgapi2_geni-cam_dotnet.dll under *References*.



Notice

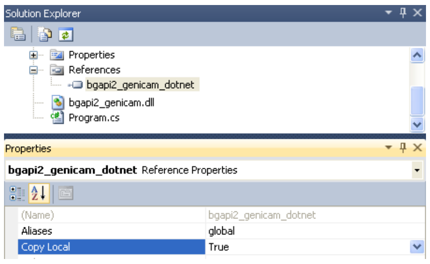
32-Bit:

```
<BGAPI SDK>\Components\Dev\C_Sharp\Win32
bgapi2_genicam_dotnet.dll
```

64-Bit

```
<BGAPI SDK>\Components\Dev\C_Sharp\x64
bgapi2_genicam_dotnet.dll
```

Then go to the added bgapi2_genicam_dotnet. dll Properties and set "Copy Local" to "True".



(Name)	bgapi2_genicam_dotnet
Aliases	global
Copy Local	True

The API is based on the GenICam™ GenAPI reference implementation.

More information can be found on: https://www.emva.org/standards-technology/genicam/

The required files are located in the following folder:

32 Bit:

```
<BGAPI SDK>\Components\Bin\Win32
```

64 Bit:

```
<BGAPI SDK>\Components\Bin\x64
```

Notice

To run the programs on a computer where Baumer GAPI is not installed, these files are absolutely necessary.

The standard parser of GenICam GenAPI reference implemetation includes the following DLLs:

```
GCBase_MD_VC120_v3_0_baumer.dll
GenApi_MD_VC120_v3_0_baumer.dll
Log_MD_VC120_v3_0_baumer.dll
log4cpp_MD_VC120_v3_0_baumer.dll
MathParser_MD_VC120_v3_0_baumer.dll
msvcp120.dll (Microsoft Visual Studio 2013 VC120)
msvcr120.dll (Microsoft Visual Studio 2013 VC120)
NodeMapData_MD_VC120_v3_0_baumer.dll
XmlParser_MD_VC120_v3_0_baumer.dll
```

Baumer GenICam DLL:

```
bgapi2_genicam.dll
```

Baumer GenTL Producer GigE:

```
bgapi2_gige.cti
bopfdrvctl.dll (PacketFilterDriver)
bsysgige.xml
```

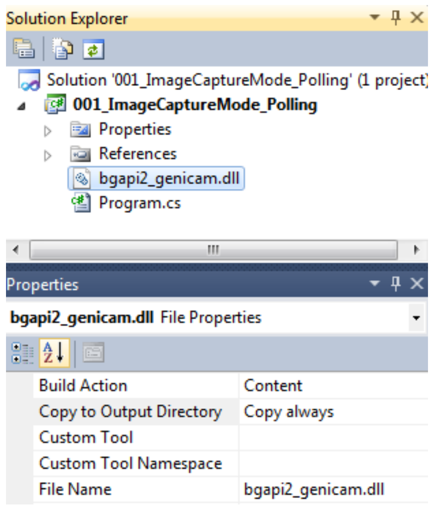
Baumer GenTL Producer USB3:

```
bgapi2_usb.cti
bsysusb.xml
```

Baumer ImageProcessor (Bayer to BGR/RGB conversion):

```
bgapi2_img.dll
```

You must set "Copy to Output Directory" to "Copy always" for each of the included files.


Solution Explorer
Solution '001_ImageCaptureMode_Polling' (1 project)
001_ImageCaptureMode_Polling
Properties
References
bgapi2_genicam.dll
Program.cs
Properties
bgapi2_genicam.dll File Properties

Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
File Name	bgapi2_genicam.dll

4.4.3.3 Using Namespace

To use the Baumer GAPI namespace for C# by default, please enter the following code:

```
using BGAPI2;
```

or add BGAPI2 to the head of all calls, e.g.:

```
BGAPI2.Interface
BGAPI2.Device
BGAPI2.Image
BGAPI2.Exception
```

5. Application development with Baumer GAPI

In this chapter, we will guide you on your way to your first image.

This tutorial assumes you are working with one interface and one camera in **C++** and **C#**.

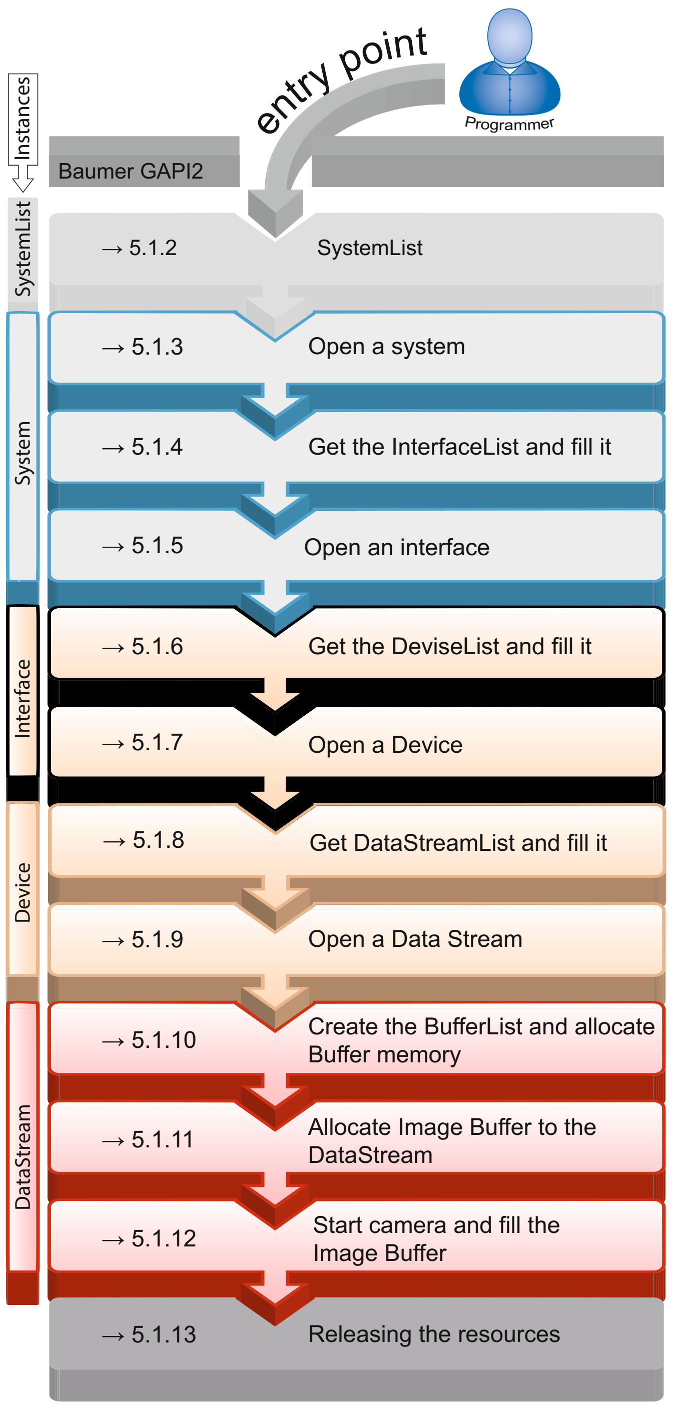
All code extracts in the following sub-chapter are provided in C++ and C#.

Notice

In order to ensure your application works smoothly, we recommend that you do not perform function calls during image acquisition at high system load. If parallel processing is required, Baumer suggests processing these function calls in a separate thread.

5.1 First steps

The program sequence is displayed schematically below, with reference to the corresponding main class and chapter.


entry point
Programmer
Baumer GAPI2
Instances
SystemList
System
Interface
Device
DataStream
→ 5.1.2 SystemList
→ 5.1.3 Open a system
→ 5.1.4 Get the InterfaceList and fill it
→ 5.1.5 Open an interface
→ 5.1.6 Get the DeviseList and fill it
→ 5.1.7 Open a Device
→ 5.1.8 Get DataStreamList and fill it
→ 5.1.9 Open a Data Stream
→ 5.1.10 Create the BufferList and allocate Buffer memory
→ 5.1.11 Allocate Image Buffer to the DataStream
→ 5.1.12 Start camera and fill the Image Buffer
→ 5.1.13 Releasing the resources

5.1.1 Preparation

5.1.1.1 Includes

First, create a new project and set the "Configuration Properties".

Include the necessary header files.

C++

```
#include " bgapi2_genicam/bgapi2_genicam.hpp"
using namespace BGAPI2;
```

The standard library and the standard input/output header also need to be included:

```
#include <stdio.h>
#include <iostream>
#include <iomanip>
```

Header file C++ (stdio.h)

Header file of the default input/output library: This library includes macro definitions, constants, declarations of functions and types.

For more information, please refer to the documentation for the standard library.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BGAPI2;
```

5.1.1.2 Declaration of Variables

We also need to declare several variables and pointers for the main function:

C++

```
int main()
{
```

C#

```
static void Main(string[] args)
{
```

Variable for *SystemList* and *System.*

C++

```
BGAPI2::SystemList * systemList = NULL;

BGAPI2::System * pSystem = NULL;

BGAPI2::String sSystemID;
```

C#

```
BGAPI2.SystemList systemList = null;

BGAPI2.System mSystem = null;

string sSystemID = "";
```

Variables for *InterfaceList* and *Interface*.

C++

```
BGAPI2::InterfaceList * interfaceList = NULL;

BGAPI2::Interface * pInterface = NULL;

BGAPI2::String sInterfaceID;
```

C#

```
BGAPI2.InterfaceList interfaceList = null;

BGAPI2.Interface mInterface = null;

string sInterfaceID = "";
```

Variable for *DeviceList* and *Device.*

C++

```
BGAPI2::DeviceList * deviceList = NULL;

BGAPI2::Device * pDevice = NULL;

BGAPI2::String sDeviceID;
```

C#

```
BGAPI2.DeviceList deviceList = null;

BGAPI2.Device mDevice = null;

string sDeviceID = "";
```

Variables for *DataStreamList* and *DataStream.*

C++

```
BGAPI2::DataStreamList * datastreamList = NULL;

BGAPI2::DataStream * pDataStream = NULL;

BGAPI2::String sDataStreamID;
```

C#

```
DataStreamList datastreamList = null;

BGAPI2.DataStream mDataStream = null;

string sDataStreamID = "";
```

Variables for *BufferList* and *Buffer.*

C++

```
BGAPI2::BufferList * bufferList = NULL;
BGAPI2::Buffer * pBuffer = NULL;
```

C#

```
BufferList bufferList = null;

BGAPI2.Buffer mBuffer = null;
```

5.1.2 SystemList

Notice

Try and catch is used for error handling. They are omitted in the following code, but should always be used.

Instantiating and updating *SystemList.*

C++

```
try
{
    systemList = SystemList::GetInstance();
    systemList->Refresh();
    std::cout << "5.1.2 Detected systems: " << systemList->size() << std::endl;
}
catch (BGAPI2::Exceptions::IException& ex)
{
    std::cout << "ExceptionType: " << ex.GetType() << std::endl;
    std::cout << "ErrorDescription: " << ex.GetErrorDescription() << std::endl;
    std::cout << "in function: " << ex.GetFunctionName() << std::endl;
}
```

C#

```
try
{
    systemList = SystemList.Instance;
    systemList.Refresh();
    System.Console.Write("5.1.2 Detected systems: {0}\n", systemList.Count);
}
catch (BGAPI2.Exceptions.IException ex)
{
    System.Console.Write("ErrorType: {0}.\n", ex.GetType());
    System.Console.Write("ErrorException {0}.\n", ex.GetErrorDescription());
    System.Console.Write("in function: {0}.\n", ex.GetFunctionName());
}
```

Output C++ /C#

```
SYSTEM LIST
###########

5.1.2  Detected systems:  2
```

5.1.3 Open a

First you need to search for the system in the list.

C++

```
for (SystemList::iterator sys = systemList->begin(); sys != systemList->end(); sys++)
{
    sys->second->Open();
    sSystemID = sys->first;
    break;
}
```

C#

```
foreach (KeyValuePair<string, BGAPI2.System> sys_pair in BGAPI2.SystemList.Instance)
{
    sys_pair.Value.Open();
    sSystemID = sys_pair.Key;
    break;
}
```

Check whether the SystemID is available and assign the system pointer.

C++

```
if (sSystemID == "")
    return 0; // no system found
else
    pSystem = (*systemList)[sSystemID];
```

C#

```
if (sSystemID == "")
    return; // no system found
else
    mSystem = systemList[sSystemID];
```

Output C++ /C#

```
SYSTEM
######

5.1.3  Open next system
```

5.1.4 Get the InterfaceList and fill it

Get the InterfaceList for the selected System and update that list. For this example we used a timeout of 100 ms.

C++

```
interfaceList = pSystem->GetInterfaces();
interfaceList->Refresh(100);
std::cout << "5.1.4 Detected interfaces: " << interfaceList->size() << std::endl;
```

C#

```
interfaceList = mSystem.Interfaces;
interfaceList.Refresh(100);
System.Console.Write("5.1.4 Detected interfaces: {0}\n", interfaceList.Count);
```

Now the *InterfaceList* is filled with InterfaceIDs (key) (example: "{9EF543BB-B4FB-4817-9D61-DAE4988C8498}") and interface pointers.

Output C++ /C#

```
INTERFACE LIST
##############

5.1.4  Detected interfaces: 5
```

5.1.5 Open an Interface

Now you can search for the interface in the list (in this case, open the first interface on the list):

C++

```
for(InterfaceList::iterator ifc = interfaceList->begin(); ifc != interfaceList->end(); ifc++)
{
    ifc->second->Open();
    sInterfaceID = ifc->first;
    break;
}
```

C#

```
foreach (KeyValuePair<string, BGAPI2.Interface> ifc_pair in interfaceList)
{
    ifc_pair.Value.Open();
    sInterfaceID = ifc_pair.Key;
    break;
}
```

Check whether the InterfaceID is filled and assign the interface pointer:

C++

```
if (sInterfaceID == "")
     return 0; // no interface found
else
     pInterface = (*interfaceList)[sInterfaceID];
```

C#

```
if (sInterfaceID == "")
     return; // no interface found
else
     mInterface = interfaceList[sInterfaceID];
```

Output C++ /C#

```
INTERFACE
#########

5.1.5  Open interface
```

5.1.6 Get the DeviceList and fill it

Get the *DeviceList* for the selected *Interface* and update that list. Using a timeout of 100 ms.

C++

```
deviceList = pInterface->GetDevices();
deviceList->Refresh(100);
std::cout << "5.1.6 Detected devices: " << deviceList->size() << std::endl;
```

C#

```
deviceList = mInterface.Devices;
deviceList.Refresh(100);
System.Console.Write("5.1.6 Detected devices: {0}\n", deviceList.Count);
```

Now the *DeviceList* is filled with DeviceIDs (example: "00_06_be_00_27_16") and Device pointers.

Output C++ /C#

```
DEVICE LIST
###########

5.1.6  Detected devices: 1
```

5.1.7 Open a Device

Notice

During the operation with Baumer Gigabit Ethernet cameras, the application requires control access to the device. This is implemented by a so called HeartbeatTimeout. This allows the software to open a message channel. Using this channel, the camera is reserved for the application and the exclusive write access to the device registers is guaranteed. The HeartbeatTimeout displays a cyclical read access to a special register of the camera hardware, with the duration of this cycle being defined by the Heartbeat-Timeout.

If this time is exceeded without a read access, the camera disconnects itself to be ready for the next connection of another application, or reconnection of the restarted PC application. The exceedance can be caused, for example, by a crashed software or a CPU overload of the PC.

In debugging mode, this timeout can cause problems. If the HeartbeatTimeout is too short, the camera will be released while the project is being debugged. On the other hand, if the HeartbeatTimeout is too long and the software or the operating system crashes during debugging (before the camera is released by the program code), the camera will be accessible once the HeartbeatTimeout has elapsed.

For further information see the example source code:

`../src/1_GigE/101_HeartbeatTimeout`

Now you can search for the device in the list. In this case, open the first device on the list.

C++

```
for (DeviceList::iterator dev = deviceList->begin(); dev != deviceList->end(); dev++)
{
    dev->second->Open();
    sDeviceID = dev->first;
    break;
}
```

C#

```
foreach (KeyValuePair<string, BGAPI2.Device> dev_pair in deviceList)
{
    dev_pair.Value.Open();
    sDeviceID = dev_pair.Key;
    break;
}
```

Check the DeviceID is filled and assign the device pointer:

C++

```
if (sDeviceID == "")
    return 0; // no device found
else
    pDevice = (*deviceList)[sDeviceID];
```

C#

```
if (sDeviceID == "")
     return; // no device found
else
     mDevice = deviceList[sDeviceID];
```

Output C++ /C#

```
DEVICE
######

5.1.7  Open first device
```

5.1.8 Get DataStreamList and fill it

Get the *DataStreamList* for the selected *Device* and update that list.

C++

```
datastreamList = pDevice->GetDataStreams();
datastreamList->Refresh();
std::cout << "5.1.8 Detected datastreams: " << datastreamList->size() << std::endl;
```

C#

```
datastreamList = mDevice.DataStreams;
datastreamList.Refresh();
System.Console.Write("5.1.8 Detected datastreams: {0}\n", datastreamList.Count);
```

Now the *DataStreamList* is filled with DataStreamIDs (example: "Stream0") and data stream pointers.

Output C++ /C#

```
DATA STREAM LIST
################

5.1.8  Detected datastreams: 1
```

5.1.9 Open a DataStream

Now you can search for the data stream in the list (in this case, open the first data stream on the list):

C++

```
for(DataStreamList::iterator dst= datastreamList->begin();dst != datastreamList->end();dst++)
{
     dst->second->Open();
     sDataStreamID = dst->first;
     break;
}
```

C#

```
foreach (KeyValuePair<string, BGAPI2.DataStream> dst_pair in datastreamList)
{
     dst_pair.Value.Open();
     sDataStreamID = dst_pair.Key;
     break;
}
```

Check the DataStreamID is filled and assign the data stream pointer:

C++

```
if (sDataStreamID == "")
     return 0; // no datastream found
else
     pDataStream = (*datastreamList)[sDataStreamID];
```

C#

```
if (sDataStreamID == "")
     return; // no datastream found
else
     mDataStream = datastreamList[sDataStreamID];
```

Now the data stream is opened.

Output C++ /C#

```
DATA STREAM
###########

5.1.9  Open first datastream
```

5.1.10 Create the BufferList and allocate Buffer memory

The *BufferList* created is empty and needs to be filled with buffers.

The `Add()` function adds a buffer to the *BufferList* and allocates the necessary storage automatically if the default constructor of the BGAPI.Buffer was used.

C++

```
bufferList = pDataStream->GetBufferList();
for(int i=0; i<4; i++)  // 4 buffers using internal buffers
{
    pBuffer = new BGAPI2::Buffer();
    bufferList->Add(pBuffer);
}
std::cout << "5.1.10 Announced buffers: " << bufferList->size() << std::endl;
```

C#

```
bufferList = mDataStream.BufferList;
for(int i=0; i<4; i++)  // 4 buffers using internal buffers
{
    mBuffer = new BGAPI2.Buffer();
    bufferList.Add(mBuffer);
}
System.Console.Write("5.1.10 Announced buffers: {0}\n", bufferList.Count);
```

Alternatively the user may allocate the buffer memory on its own. The responsability to release the allocated memory then remains with the user.
(see example: *004_PartialScan_ExternalBuffer*).

Notice

Changing the payload-size during aquisition requires a Buffer of different size. If the Buffer size does not meet the required payload size **image data will be discarded**.

Using the maximum payload-size assures to never lose image data.

Output C++ /C#

```
BUFFER LIST
###########

5.1.10  Announced buffers:        4
```

5.1.11 Allocate Buffer to the DataStream

C++

```
for (BufferList::iterator buf = bufferList->begin(); buf != bufferList->end(); buf++)
{
    buf->second->QueueBuffer();
}
std::cout << "5.1.11 Queued buffers: " << bufferList->GetQueuedCount() << std::endl;
```

C#

```
foreach (KeyValuePair<string, BGAPI2.Buffer> buf_pair in bufferList)
{
    buf_pair.Value.QueueBuffer();
}
System.Console.Write("5.1.11. Queued buffers: {0}\n", bufferList.Count);
```

Output C++ /C#

```
BUFFER LIST
###########

5.1.11  Queued buffers:        4
```

5.1.12 Start Camera and fill the Buffer

Start the data stream.

C++

```
pDataStream->StartAcquisitionContinuous();
```

C#

```
mDataStream.StartAcquisition();
```

A significant factor for data stream performance is the buffer size used for camera payload. To improve this performance there is the `TLParamsLocked mechanism`. This mechanism prevents changes in payload size during data streaming.

Starting data streaming locks all of the camera features that influence payload size. These features can only be unlocked by stopping the data stream.

Examples of locked features:

Locked features (e.g. LXG) [Category: ImageFormatControl]	
BinningHorizontal	OffsetX
BinningVertical	OffsetY
DecimationHorizontal	PixelFormat
DecimationHorizontal	RegionMode
DecimationVertical	Width
Height	

C++

```
// if camera supports TLParamsLocked it is set automatically now
pDataStream->StartAcquisitionContinuous();

// CANNOT change payloadsize relevant features now
if( pDevice->GetRemoteNode("Height")->IsWriteable() == true )
{
    // safe way to check if feature is locked by TLParamsLocked
}

// if camera supports TLParamsLocked it is reset automatically now
pDataStream->StopAcquisition();

// can change payloadsize relevant features now
if( pDevice->GetRemoteNode("Height")->IsWriteable() == true )
{
    // safe way to check if feature is locked by TLParamsLocked
}
```

C#

```
// if camera supports TLParamsLocked it is set automatically now
pDataStream.StartAcquisition();

// CANNOT change payloadsize relevant features now
if (mDevice.RemoteNodeList["Height"].IsWriteable == true)
{
     // safe way to check if feature is locked by TLParamsLocked
}

// if camera supports TLParamsLocked it is reset automatically now
pDataStream.StopAcquisition();

// can change payloadsize relevant features now
if (mDevice.RemoteNodeList["Height"].IsWriteable == true)
{
     // safe way to check if feature is locked by TLParamsLocked
}
```

Start the camera.

C++

```
pDevice->GetRemoteNode("AcquisitionStart")->Execute();
```

C#

```
mDevice.RemoteNodeList["AcquisitionStart"].Execute();
```

Output C++ /C#

```
CAMERA START
#############

5.1.12  DataStream started
5.1.12  HXG20c started
```

Fill the image buffer using a timeout of 1000 milliseconds. The memory pointer to the image data is displayed.

C++

```
BGAPI2::Buffer * pBufferFilled = NULL;
try
{
    pBufferFilled = pDataStream->GetFilledBuffer(1000);
    if(pBufferFilled == NULL)
{
    std::cout << "Error: Buffer Timeout after 1000 msec" << std::endl;
}
else
    {
            std::cout << " Image " << pBufferFilled->GetFrameID();
            std::cout << " received in memory address " << pBufferFilled->GetMemPtr();
            std::cout << std::endl;
    }
}
catch (BGAPI2::Exceptions::IException& ex)
{
    std::cout << "ExceptionType: " << ex.GetType() << std::endl;
    std::cout << "ErrorDescription: " << ex.GetErrorDescription() << std::endl;
    std::cout << "in function: " << ex.GetFunctionName() << std::endl;
}
```

Buffer::GetFrameID()

GigE-Vision 1.2: *The function delivers a 16bit counter, which starts at 1 and restarts at 1 once it reaches 65535 (0 is skipped).*

USB3 Vision 1.0: *The function delivers a 64bit counter, which starts at 0 and restarts at 0 once it reaches 18446744073709551615.*

In both cases, the counter increases when the interface transfers an image on the camera side.

The maximum for the FrameID can be queried. See documentation: Main Class Features.

BufferList::GetDeliveredCount()

This function delivers a 64bit unsigned int value, which starts at 1. It is increased when the GigE Producer receives an image and transfers it to the user.

Buffer::GetChunkNodeTree()::GetNode("ChunkFrameID")::GetInt()

The function delivers a camera-generated counter. Size and start value depend on the camera. The value is increased when the camera takes a picture.

C#

```
BGAPI2.Buffer mBufferFilled = null;
try
{
     mBufferFilled = mDataStream.GetFilledBuffer(1000);
     if(mBufferFilled == null)
     {
             System.Console.Write("Error: Buffer Timeout after 1000 msec\n");
     }
     else
     {
             System.Console.Write(" Image{0} ", mBufferFilled.FrameID);
             System.Console.Write("received in memory address {0:X}\n",
             (ulong)mBufferFilled.MemPtr);
     }
}
catch (BGAPI2.Exceptions.IException ex)
{
     System.Console.Write("ExceptionType: {0}.\n", ex.GetType());
     System.Console.Write("ErrorDescription: {0}.\n", ex.GetErrorDescription());
     System.Console.Write("in function: {0}\n", ex.GetFunctionName());
}
```

Output C++ /C#

```
CAPTURE 12 IMAGES BY IMAGE POLLING
################################

 Image     1 received in memory address 03000020
 ...
 ...
 Image     n received in memory address 03000020
```

Following image processing, put the *Buffer* object into the queue:

C++

```
pBufferFilled->QueueBuffer();
```

C#

```
mBufferFilled.QueueBuffer();
```

Stop the camera:

C++

```
pDevice->GetRemoteNode("AcquisitionStop")->Execute();
```

C#

```
mDevice.RemoteNodeList["AcquisitionStop"].Execute();
```

Stop the data stream and delete the buffers:

C++

```
pDataStream->StopAcquisition();
bufferList->DiscardAllBuffers();
while( bufferList->size() > 0)
{
     pBuffer = bufferList->begin()->second;
     bufferList->RevokeBuffer(pBuffer);
     delete pBuffer;
}
```

C#

```
mDataStream.StopAcquisition();
bufferList.DiscardAllBuffers();
while (bufferList.Count > 0)
{
     mBuffer = bufferList.Values.First();
     bufferList.RevokeBuffer(mBuffer);
}
```

Output C++ /C#

```
CAMERA STOP
###########

5.1.12   HXG20c aborted
5.1.12   HXG20c stopped

5.1.12   DataStream stopped
```

5.1.13 Releasing the resources

C++

```
pDataStream->Close();
pDevice->Close();
pInterface->Close();
pSystem->Close();
BGAPI2::SystemList::ReleaseInstance();
```

C#

```
mDataStream.Close();
mDevice.Close();
mInterface.Close();
mSystem.Close();
```

Use the following code to prevent the application from closing.

C++

```
  int endKey = 0;
  std::cout << "Input any number to close the program:";
  std::cin >> endKey;
  return 0;
} // end of main
```

C#

```
System.Console.Write("Input any number to close the program:\n");
Console.Read();
return;
} // end of Main
```

Output C++ /C#

```
RELEASE
#######

5.1.13   Releasing the resources
         buffers after revoke:    0

End

Input any number to close the program:
```

5.2 Information about System, Interface, Device and DataStream

In this chapter, we will show you how to get information about *System* (Producer), *Interface*, *Device* and *DataStream*.

This information is necessary to distinguish between multiple located objects and make the correct choice. (e.g. correctly selecting the interface card to which the camera is connected).

Notice

To increase clarity, the code for initialization and image grabbing is removed. Only the changes to the respective parameter are shown.

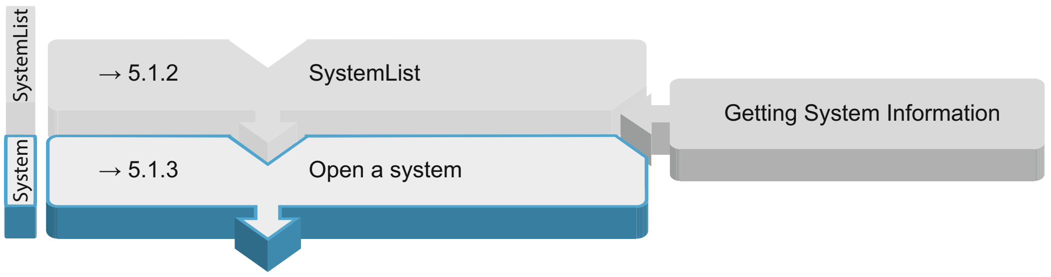
5.2.1 Getting System (Producer) Information

You can request information about the *System* here.

Notice

Not all available system information is shown in this example.

You can find further queryable parameters in chapter 3.2.1.



Querying the *System* information from the *SystemList* before opening a *System* (e.g. Name, TLType & Version).

C++

```cpp
for (SystemList::iterator sys = systemList->begin(); sys != systemList->end(); sys++)
{
    std::cout << "System Name:" << sys->second->GetFileName() << std::endl;
    std::cout << "System Type:" << sys->second->GetTLType() << std::endl;
    std::cout << "System Version:" << sys->second->GetVersion() << std::endl;
}
```

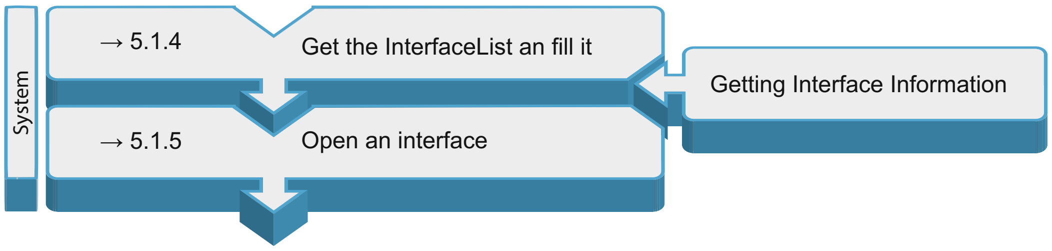
C#

```csharp
foreach (KeyValuePair<string, BGAPI2.System> sys_pair in BGAPI2.SystemList.Instance)
{
    System.Console.Write("System Name:          {0}\n", sys_pair.Value.FileName);
    System.Console.Write("System Type:          {0}\n", sys_pair.Value.TLType);
    System.Console.Write("System Version:       {0}\n", sys_pair.Value.Version);
}
```

Output C++ /C#

```
SYSTEM LIST
###########

5.1.2   Detected systems:  2
  5.2.1   System Name:     bgapi2_gige.cti
          System Type:     GEV
          System Version:  2.2.3170.3184

  5.2.1   System Name:     bgapi2_usb.cti
          System Type:     U3V
          System Version:  2.2.3184.3184
```

5.2.2 Getting Interface Information


System
→ 5.1.4
Get the InterfaceList an fill it
→ 5.1.5
Open an interface
Getting Interface Information

Querying the *Interface* information from the *InterfaceList* before opening an *Interface* (e.g. ID, TLType & DisplayName).

The InterfaceID is also the key in the list (first parameter) and therefore the Producer wide unique identifier for the selected interface (e.g.: "{9EF543BB-B4FB-4817-9D61-DAE4988C8498}").

Notice

Not all available interface information is shown in this example.

You can find further queryable parameters in chapter 3.2.2.

C++

```
for(InterfaceList::iterator ifc = interfaceList->begin(); ifc != interfaceList->end(); ifc++)
{
    std::cout << "Interface ID:   " << ifc->first << std::endl;
    std::cout << "Interface Type: " << ifc->second->GetTLType() << std::endl;
    std::cout << "Interface Name: " << ifc->second->GetDisplayName() << std::endl;
}
```

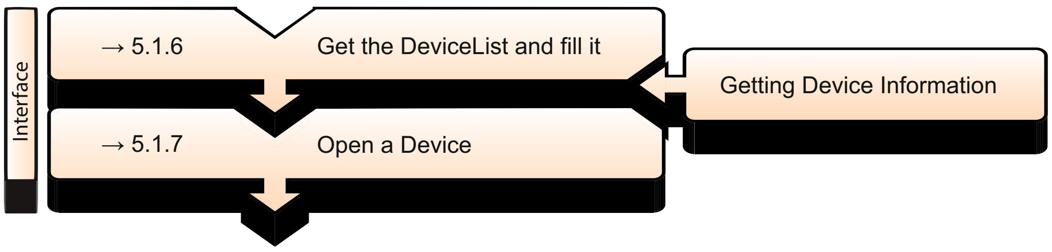
C#

```
foreach (KeyValuePair<string, BGAPI2.Interface> ifc_pair in interfaceList)
{
    System.Console.Write("Interface ID:   {0}\n", ifc_pair.Value.Id);
    System.Console.Write("Interface Type: {0}\n", ifc_pair.Value.TLType);
    System.Console.Write("Interface Name: {0}\n", ifc_pair.Value.DisplayName);
}
```

Output C++ /C#

```
INTERFACE LIST
##############

5.1.4   Detected interfaces: 5
  5.2.2   Interface ID:      {20423325-F0B6-46E6-A323-04388A222A09}
          Interface Type:    GEV
          Interface Name:    Intel(R) Ethernet Server Adapter I350-T4 #2

  5.2.2   Interface ID:      {45D70EFC-D91C-46E4-98FB-70D872BB8EEB}
          Interface Type:    GEV
          Interface Name:    Intel(R) Ethernet Server Adapter I350-T4 #4

  5.2.2   Interface ID:      {9EF543BB-B4FB-4817-9D61-DAE4988C8498}
          Interface Type:    GEV
          Interface Name:    Broadcom NetXtreme 57xx-Gigabit-Controller

  5.2.2   Interface ID:      {B19BC4E9-F020-408C-AC6E-343C8908559A}
          Interface Type:    GEV
          Interface Name:    Intel(R) Ethernet Server Adapter I350-T4

  5.2.2   Interface ID:      {C11F5EA3-E476-437A-A710-C8B1389478D6}
          Interface Type:    GEV
          Interface Name:    Intel(R) Ethernet Server Adapter I350-T4 #3
```

5.2.3 Getting Device Information



Querying the device information from the *DeviceList* before opening a device (e.g. ID, Model, Vendor, TLType & DisplayName).

The DeviceID is also the key in the list (first parameter) and therefore the Interface wide unique identifier for the selected device. (e.g.: "00_06_be_00_27_16").

The display name is the camera's arbitrary user ID (GigE - 15 characters; USB - 63 characters).

Notice

Not all available device information is shown in this example.

You can find further queryable parameters in chapter 3.2.3.

C++

```
for (DeviceList::iterator dev = deviceList->begin(); dev != deviceList->end(); dev++)
{
    std::cout << "Device DeviceID:" << dev->first << std::endl;
    std::cout << "Device Model:" << dev->second->GetModel() << std::endl;
    std::cout << "Device SerialNumber:" << dev->second->GetSerialNumber() << std::endl;
    std::cout << "Device Vendor:" << dev->second->GetVendor() << std::endl;
    std::cout << "Device TLType:" << dev->second->GetTLType() << std::endl;
    std::cout << "Device UserID:" << dev->second->GetDisplayName() << std::endl;
}
```

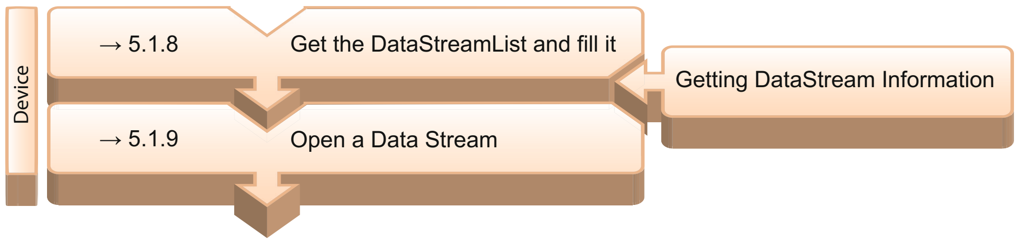
C#

```
foreach (KeyValuePair<string, BGAPI2.Device> dev_pair in deviceList)
{
    System.Console.Write("Device DeviceID: {0}\n", dev_pair.Key);
    System.Console.Write("Device Model: {0}\n", dev_pair.Value.Model);
    System.Console.Write("Device SerialNumber: {0}\n", dev_pair.Value.SerialNumber);
    System.Console.Write("Device Vendor: {0}\n", dev_pair.Value.Vendor);
    System.Console.Write("Device TLType: {0}\n", dev_pair.Value.TLType);
    System.Console.Write("Device UserID: {0}\n", dev_pair.Value.DisplayName);
}
```

Output C++ /C#

```
DEVICE LIST
###########

5.1.6   Detected devices:         1
  5.2.3   Device DeviceID:        00_06_be_00_44_10
          Device Model:           HXG20c
          Device SerialNumber:    0174240712
          Device Vendor:          Baumer Optronic
          Device TLType:          GEV
          Device AccessStatus:    RW
          Device UserID:          Camera0
```

5.2.4 Getting DataStream Information


Device
→ 5.1.8
Get the DataStreamList and fill it
→ 5.1.9
Open a Data Stream
Getting DataStream Information

Querying the *DataStream* information from the *DataStreamList* before opening a *DataStream*.

DataStreamID: Device wide unique identifier for the selected *DataStream*, example: "Stream0"**.**

Notice

Not all available DataStream information is shown in this example.

You can find further queryable parameters in chapter 3.2.4.

C++

```
for(DataStreamList::iterator dst = datastreamList->begin(); dst!= datastreamList->end();
dst++)
{
     std::cout << "DataStream ID: " << dst->first << std::endl;
}
```

C#

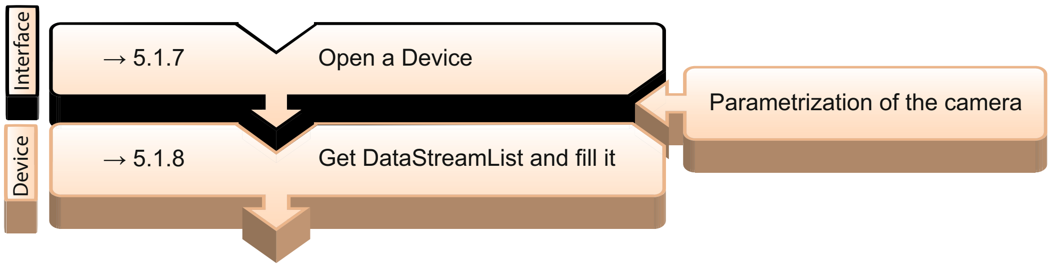
```
foreach (KeyValuePair<string, BGAPI2.DataStream> dst_pair in datastreamList)
{
     System.Console.Write("DataStream ID: {0}\n", dst_pair.Key);
}
```

Output C++ /C#

```
DATA STREAM LIST
################

5.1.8   Detected datastreams:     1
  5.2.4   DataStream ID:          Stream0
```

5.3 Parametrization of the camera

This section describes the general approach for handling certain camera features.



5.3.1 Camera feature categories

The camera features are divided into several categories. These categories are described in the XML description file nodes with the ICategory interface type.

The following table shows the most common categories in the XML description file.

ICategory	
AcquisitionControl	DigitalIOControl
ActionControl	EventControl
AnalogControl	HDRControl
BoSequencerControl	ImageFormatControl
ChunkDataControl	LUTControl
CounterAndTimerControl	TransportLayerControl
DeviceControl	UserSetControl

Each category has sub-nodes describing the features.

Notice

Features which have a "-" for their possible values have no possible values. These are wo (write only) features.

Example:

ICategory	Interface type	Feature	Possible Value
AcquisitionControl	[ICommand]	AcquisitionAbort	-
	[IFloat]	AcquisitionFrameRate	50
	[IBoolean]	AcquisitionFrameRateEnable	1
	[IEnumeration]	AcquisitionMode	Continuous
	[ICommand]	AcquisitionStart	-
	[ICommand]	AcquisitionStop	-
	[IEnumeration]	ExposureMode	Timed
	[IFloat]	ExposureTime	4000
	[IEnumeration]	ReadoutMode	Sequential
	[IEnumeration]	TriggerActivation	RisingEdge
	[IFloat]	TriggerDelay	0
	[IEnumeration]	TriggerMode	Off
	[IEnumeration]	TriggerOverlap	ReadOut
	[IEnumeration]	TriggerSelector	FrameStart
	[ICommand]	TriggerSoftware	-
	[IEnumeration]	TriggerSource	Line0

Interface types:

Interface types	Description
[IBoolean]	For features in the *IBoolean* interface type, you can choose between *False* or *True*.
[ICommand]	An *ICommand* interface type feature is used to perform a single action.
[IEnumeration]	*IEnumeration* interface types are a collection of values as strings and integers.
[IFloat]	*IFloat* interface types include double values (floating point).
[IInteger]	*IInteger* interface types include 64 bit integer values.
[IString]	*IString* interface types are represented as a simple string value.

To list the camera's categories, use RemoteNodeTree (not RemoteNodeList):

C++

```
BGAPI2::NodeMap * nmNodeTree = pDevice->GetRemoteNodeTree();
for( bo_uint64 i = 0; i < nmNodeTree->GetNodeCount(); i++)
{
    BGAPI2::Node * nNode = nmNodeTree->GetNodeByIndex(i);
    std::cout << nNode->GetInterface() << " " << nNode->GetName() << std::endl;
}
```

C#

```
for (ulong i = 0; i < mDevice.RemoteNodeTree.Count; i++)
{
    System.Console.Write("{0} ", mDevice.RemoteNodeTree[i].Interface);
    System.Console.Write("{0}\n", mDevice.RemoteNodeTree[i].Name);
}
```

Refer to the example code of "002_CameraParameterTree.cpp" or "002_CameraParameterTree.cs" to learn how to display all categories, feature nodes and, where available, their values.

5.3.2 IBoolean Interface Type Example "AcquisitionFrameRateEnable"

IBoolean interface types are used to switch features on and off. For example, *Acquisition-FrameRateEnable* or *LUTEnable*.

They can also be used for digital inputs and outputs where values can be "0" or "1", e.g. *LineStatus* and *LineInverter*.

Short Reference

C++

Read: `pDevice->GetRemoteNode("AcquisitionFrameRateEnable")->GetBool();`

Write: `pDevice->GetRemoteNode("AcquisitionFrameRateEnable")->SetBool(true);`

C#

Read: `(bool)mDevice.RemoteNodeList["AcquisitionFrameRateEnable"].Value;`

Write: `mDevice.RemoteNodeList["AcquisitionFrameRateEnable"].Value = true;`

5.3.3 IString Interface Type Example "DeviceUserID"

The IString interface type can be used to set a string of your choice. Unfortunately, there is no simple example for this.

Short Reference

C++

Read: `pDevice->GetRemoteNode("NodeName")->GetString();`

Write: `pDevice->GetRemoteNode("NodeName")->SetString("Camera0");`

C#

Read: `pDevice->GetRemoteNode("DeviceUserID")->GetString();`

Write: `pDevice->GetRemoteNode("DeviceUserID")->SetString("Camera0");`

Detailed code

C++

```
std::cout << "DeviceUserID" << std::endl;
std::cout << "description: " << pDevice->GetRemoteNode("DeviceUserID")->GetDescription() <<
std::endl;
std::cout << "interface type: " << pDevice->GetRemoteNode("DeviceUserID")->GetInterface() <<
std::endl;
std::cout << "current value: " << pDevice->GetRemoteNode("DeviceUserID")->GetString() <<
std::endl;

// SET A NEW USER ID LIKE "Camera0"
BGAPI2::String newDeviceUserID = "Camera0";

//CONFIRM STRING LENGTH IS MATCHING
if( newDeviceUserID.size() <= pDevice->GetRemoteNode("DeviceUserID")->GetMaxStringLength() )
{
        pDevice->GetRemoteNode("DeviceUserID")->SetString(newDeviceUserID);
}
else
{
        std::cout << "Error: newDeviceUserID string length (" << newDeviceUserID.size() <<
        ")
        ") is longer than MaxStringthLength (" << pDevice->GetRemoteNode("DeviceUserID")
        ->GetMaxStringLength() << ")" << std::endl;
}

//RECHECK CHANGES
std::cout << "set value to: " << pDevice->GetRemoteNode("DeviceUserID")->GetString() <<
std::endl;
std::cout << std::endl;
```

C#

```
System.Console.Write("5.3.6   DeviceUserID\n");
System.Console.Write("description: {0}\n",
                            (string)mDevice.RemoteNodeList["DeviceUserID"].Description);
System.Console.Write("interface type: {0}\n",
                            (string)mDevice.RemoteNodeList["DeviceUserID"].Interface);
System.Console.Write("current value: {0}\n",
                            mDevice.RemoteNodeList["DeviceUserID"].Value);

//SET A NEW USER ID LIKE "Camera0"
string newDeviceUserID = "Camera0";

//CONFIRM STRING LENGTH IS MATCHING
if (newDeviceUserID.Length <= 15)
{
         mDevice.RemoteNodeList["DeviceUserID"].Value = newDeviceUserID;
}
else
{
         System.Console.Write("Error: newDeviceUserID string length ({0})",
                  newDeviceUserID.Length);
         System.Console.Write("is longer than MaxStringthLength ({0})",
                  15);
}

//RECHECK CHANGES
System.Console.Write("set value to: {0}\n",
           mDevice.RemoteNodeList["DeviceUserID"].Value);
System.Console.Write(" \n");
```

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################

5.3.6   DeviceUserID
          description:            User-programmable device identifier.
          interface type:         IString
          current value:          Camera0
          set value to:           Camera0
```

5.3.4 IFloat Interface Type Example "ExposureTime"

Read the current value of "ExposureTime" and the max/min values to check whether the new value is inside the range.

Then set the "ExposureTime" to 20000 µsec, for example.

Short Reference

C++

Read: `pDevice->GetRemoteNode("ExposureTime")->GetDouble();`

Write: `pDevice->GetRemoteNode("ExposureTime")->SetDouble(20000);`

C#

Read: `(double)mDevice.RemoteNodeList["ExposureTime"].Value;`

Write: `mDevice.RemoteNodeList["ExposureTime"].Value = (double)20000;`

Detailed code with max/min values request

C++

```
bo_double fExposureTime = pDevice->GetRemoteNode("ExposureTime")->GetDouble();
bo_double fExposureTimeMin = pDevice->GetRemoteNode("ExposureTime")->GetDoubleMin();
bo_double fExposureTimeMax = pDevice->GetRemoteNode("ExposureTime")->GetDoubleMax();

//set new exposure value to 20000 usec
bo_double exposurevalue = 20000;

// check new value is within range
if( exposurevalue < fExposureTimeMin)
     exposurevalue = fExposureTimeMin;

if( exposurevalue > fExposureTimeMax)
     exposurevalue = fExposureTimeMax;

pDevice->GetRemoteNode("ExposureTime")->SetDouble(exposurevalue);

std::cout << "Set Exposure to: " << pDevice->GetRemoteNode("ExposureTime")->GetDouble();
std::cout << std::endl;
```

C#

```
double fExposureTime = (double)mDevice.RemoteNodeList["ExposureTime"].Value;
double fExposureTimeMin = (double)mDevice.RemoteNodeList["ExposureTime"].Min;
double fExposureTimeMax = (double)mDevice.RemoteNodeList["ExposureTime"].Max;

//set new exposure value to 20000 usec
double exposurevalue = 20000;

// check new value is within range
if (exposurevalue < fExposureTimeMin)
    exposurevalue = fExposureTimeMin;

if (exposurevalue > fExposureTimeMax)
    exposurevalue = fExposureTimeMax;

mDevice.RemoteNodeList["ExposureTime"].Value = exposurevalue;

System.Console.Write("Set Exposure to: {0}\n\n",
                     (double)mDevice.RemoteNodeList["ExposureTime"].Value);
```

Notice

With C#, the cast (double) is needed to assign the correct type to the object .Value.

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################

5.3.2 ExposureTime
        description:        Sets the exposure time (in microseconds) when ExposureMode is timed.
        interface type:         IFloat
        current value:          4000
        possible value range:   4 to 100000
        set value to:           20000
```

5.3.5 IInteger Interface Type Example "Width"

Read the current value of "Width" and the max/min values to check whether the new value is inside the range.

Then set the "Width" of the image to the same value as the image height, for example.

Short Reference

C++

Read: `pDevice->GetRemoteNode("Width")->GetInt();`

Write: `pDevice->GetRemoteNode("Width")->SetInt(480);`

C#

Read: `(long)mDevice.RemoteNodeList["Width"].Value;`

Write: `mDevice.RemoteNodeList["Width"].Value = (long)480;`

Detailed code with max/min values request

C++

```
bo_int64 iImageWidth = pDevice->GetRemoteNode("Width")->GetInt();
bo_int64 iImageWidthMin = pDevice->GetRemoteNode("Width")->GetIntMin();
bo_int64 iImageWidthMax = pDevice->GetRemoteNode("Width")->GetIntMax();
bo_int64 iImageWidthInc = pDevice->GetRemoteNode("Width")->GetIntInc();

//set new width value same to the value of height
bo_int64 widthvalue = pDevice->GetRemoteNode("Height")->GetInt();
// find number to match the increment
widthvalue = widthvalue / iImageWidthInc * iImageWidthInc;

// check new value is within range
if( widthvalue < iImageWidthMin)
     widthvalue = iImageWidthMin;

if( widthvalue > iImageWidthMax)
     widthvalue = iImageWidthMax;

pDevice->GetRemoteNode("Width")->SetInt(widthvalue);

std::cout << "Set Width to: " << pDevice->GetRemoteNode("Width")->GetInt() << std::endl;
```

C#

```
long iImageWidth = (long)mDevice.RemoteNodeList["Width"].Value;
long iImageWidthMin = (long)mDevice.RemoteNodeList["Width"].Min;
long iImageWidthMax = (long)mDevice.RemoteNodeList["Width"].Max;
long iImageWidthInc = (long)mDevice.RemoteNodeList["Width"].Inc;

//set new width value same to the value of height
long widthvalue = (long)mDevice.RemoteNodeList["Height"].Value;
// find number to match the increment
widthvalue = widthvalue / iImageWidthInc * iImageWidthInc;

// check new value is within range
if (widthvalue < iImageWidthMin)
     widthvalue = iImageWidthMin;

if (widthvalue > iImageWidthMax)
     widthvalue = iImageWidthMax;

mDevice.RemoteNodeList["Width"].Value = widthvalue;

System.Console.Write("Set Width to: {0}\n\n", (long)mDevice.RemoteNodeList["Width"].Value);
```

Notice

With C#, the cast (long) is needed to assign the correct type to the object .Value.

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################

5.3.3   Width
          description:            Width of the image provided by the device (in pixels).
          interface type:         IInteger
          current value:          2048
          possible value range:   32 to 2048 in increments of 32
          set value to:           1088 is about the same as the height: 1088
```

5.3.6 IEnumeration Interface Type Example "TriggerSource"

List all available values of the String type and note the current value of "TriggerSource".

Then set the "`TriggerSource`" to "`Software`", for example.

Short Reference with string access

C++

Read: `pDevice->GetRemoteNode("TriggerSource")->GetString();`

Write: `pDevice->GetRemoteNode("TriggerSource")->SetString("Software");`

C#

Read: `(string)mDevice.RemoteNodeList["TriggerSource"].Value);`

Write: `mDevice.RemoteNodeList["TriggerSource"].Value = "Software";`

Short Reference with integer access

Notice

The following example is based on the "TriggerSource" feature of the MXG camera. Values for other cameras may differ.

Trigger Source

Enumeration String	Enumeration Integer
Software	3
Line0	4
Action1	2
Off	0

Pixel Format

Enumeration String	Enumeration Integer
BayerRG8	0x1080009
BayerRG12	0x1100011
Mono8	0x1080001

C++

Read: `pDevice->GetRemoteNode("TriggerSource")->GetInt();`

Write: `pDevice->GetRemoteNode("TriggerSource")->SetInt(3);`

C#

Read: `(long)mDevice.RemoteNodeList["TriggerSource"].EnumerationValue;`

Write: `mDevice.RemoteNodeList["TriggerSource"].EnumerationValue = (long)3;`

To get the list of available values, use:

C++

```
pDevice->GetRemoteNode("TriggerSource")->GetEnumNodeList()->GetNodeCount();
pDevice->GetRemoteNode("TriggerSource")->GetEnumNodeList()->GetNodeByIndex(i)->GetValue();
```

C#

```
mDevice.RemoteNodeList["TriggerSource"].EnumNodeList.Count;
mDevice.RemoteNodeList["TriggerSource"].EnumNodeList[i].Value;
```

Detailed code with list of the possible values

C++

```
bo_uint64 count = pDevice->GetRemoteNode("TriggerSource")->GetEnumNodeList()->GetNodeCount();
for(bo_uint64 i = 0; i < count; i++)
{
    BGAPI2::Node * pNode = NULL;
    pNode = pDevice->GetRemoteNode("TriggerSource")->GetEnumNodeList()->GetNodeByIndex(i);

    //check each node if it is available for this camera
    if( pNode->IsReadable() == true )

    {
            std::cout << i << ": ";
            if(pNode->GetValue() == pDevice->GetRemoteNode("TriggerSource")->GetValue())
            {
                    std::cout << "*";  // current value marked with "*"
            }
            else
            {
                    std::cout << " ";
            }
            std::cout << pNode->GetValue() << std::endl;
    }
}

//set TriggerSource "Software"
pDevice->GetRemoteNode("TriggerSource")->SetValue("Software");

std::cout << "Set TriggerSource to: " << pDevice->GetRemoteNode("TriggerSource")->GetValue();
std::cout << std::endl;
```

C#

```
ulong count = mDevice.RemoteNodeList["TriggerSource"].EnumNodeList.Count;
for (ulong i = 0; i < count; i++)
{
     BGAPI2.Node mNode = mDevice.RemoteNodeList["TriggerSource"].EnumNodeList[i];

     //check each node if it is available for this camera
     if (mNode.IsReadable == true)
     {


             System.Console.Write("{0 }: ", i);
             string currentValue = (string)mDevice.RemoteNodeList["TriggerSource"].Value);
             if ((string)mNode.Value == currentValue)
             {
                     System.Console.Write("*");  // current value marked with "*"
             }
             else
             {
                     System.Console.Write(" ");
             }
             System.Console.Write("{0}\n", mNode.Value);
     }
}

//set TriggerSource "Software"
mDevice.RemoteNodeList["TriggerSource"].Value = "Software";

System.Console.Write("Set TriggerSource to:  {0}\n",
                     mDevice.RemoteNodeList["TriggerSource"].Value);
```

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################

5.3.4   TriggerSource
          description:            Specifies the source for the trigger (input line signal, soft-
ware, ...).
          interface type:         IEnumeration
          enumeration list count: 6 (current marked with *)
                              0:  Action1
                              1: *Line0
                              2:  Line1
                              3:  Line2
                              4:  Off
                              5:  Software
          set value to:           Software

        TriggerMode
          description:            Controls whether the selected trigger is active.
          interface type:         IEnumeration
          current value:          Off
          set value to:           On
```

5.3.7 Selectors

A selector is used to index which instance of the feature is accessed in situations where multiple instances of a feature exist.

A selector is a separate feature that is typically an IEnumeration or an IInteger interface. Selectors must only be used to select the target features for subsequent changes. You may not change the behavior of a Producer in response to a change to a selector value.

5.3.7.1 IEnumeration Interface Type Selector Example "LineSelector"

The "LineSelector" feature needs to be set to a particular I/O line from the enumeration list before you can access the dependant features such as "LineInverter" or "LineStatus".

Short Reference

C++

Read: `pDevice->GetRemoteNode("LineSelector")->GetString();`

Write: `pDevice->GetRemoteNode("LineSelector")->SetString("Line0");`

C#

Read: `(string)mDevice.RemoteNodeList["LineSelector"].Value;`

Write: `mDevice.RemoteNodeList["LineSelector"].Value = "Line0";`

Detailed code

C++

```
//select an I/O line first
pDevice->GetRemoteNode("LineSelector")->SetString("Line0");
std::cout << "Set LineSelector to: " << pDevice->GetRemoteNode("LineSelector")->GetString();
std::cout << std::endl;

std::cout << "Current LineInverter: " << pDevice->GetRemoteNode("LineInverter")->GetBool();
std::cout << std::endl;
std::cout << "Current LineStatus: " << pDevice->GetRemoteNode("LineStatus")->GetBool();
std::cout << std::endl;
//set LineInverter of Line0 to true
pDevice->GetRemoteNode("LineInverter")->SetBool(true);
//recheck new value
std::cout << "Set LineInverter to: " << pDevice->GetRemoteNode("LineInverter")->GetBool();
std::cout << std::endl;
std::cout << "Current LineStatus: " << pDevice->GetRemoteNode("LineStatus")->GetBool();
std::cout << std::endl;
```

C#

```
//select an I/O line first
mDevice.RemoteNodeList["LineSelector"].Value = "Line0";
System.Console.Write("Set LineSelector to: {0}\n",
(string)mDevice.RemoteNodeList["LineSelector"].Value);

System.Console.Write("Current LineInverter: {0}\n",
(bool)mDevice.RemoteNodeList["LineInverter"].Value);
System.Console.Write("Current LineStatus: {0}\n",
(bool)mDevice.RemoteNodeList["LineStatus"].Value);
//set LineInverter of Line0 to true
mDevice.RemoteNodeList["LineInverter"].Value = true;
//recheck new value
System.Console.Write("Set LineInverter to: {0}\n",
(bool)mDevice.RemoteNodeList["LineInverter"].Value);
System.Console.Write("Current LineStatus: {0}\n",
(bool)mDevice.RemoteNodeList["LineStatus"].Value);
```

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################

Set LineSelector to:  Line0
Current LineInverter: 0
Current LineStatus:   0
Set LineInverter to:  1
Current LineStatus:   1
```

5.3.7.2 IInteger Interface Type Selector Example "HDRIndex"

Selectors from the IInteger interface type are selected in order to index various parameter sets (e.g. HDR Parameter Set).

Notice

The HDR feature is supported by Baumer cameras such as the HXG, MXGC, VLG-22M, VLG-22C, VLG-40M, VLG-40C, LXG and others.

Short Reference

C++

Read: `pDevice->GetRemoteNode("HDRIndex")->GetInt();`

Write: `pDevice->GetRemoteNode("HDRIndex")->SetInt(0);`

C#

Read: `(long)mDevice.RemoteNodeList["HDRIndex"].Value;`

Write: `mDevice.RemoteNodeList["HDRIndex"].Value = (long)0;`

Detailed code

C++

```cpp
std::cout << "HDR parameter change" << std::endl;

pDevice->GetRemoteNode("HDREnable")->SetBool(true);

std::cout << "HDREnable : "
          << pDevice->GetRemoteNode("HDREnable")->GetBool()
          << std::endl;



pDevice->GetRemoteNode("HDRIndex")->SetInt(0);

std::cout << "HDRIndex: "
          << pDevice->GetRemoteNode("HDRIndex")->GetInt()
          << std::endl;

pDevice->GetRemoteNode("HDRExposureRatio")->SetInt(185); //t_Exp_0

std::cout << "HDRExposureRatio: "
          << pDevice->GetRemoteNode("HDRExposureRatio")->GetInt()
          << std::endl;

std::cout << "HDRExposureRatioPercent : "
          << pDevice->GetRemoteNode("HDRExposureRatioPercent")->GetDouble()
          << std::endl;

pDevice->GetRemoteNode("HDRPotentialAbs")->SetInt(40); //Pot_0

std::cout << "HDRPotentialAbs : "
          << pDevice->GetRemoteNode("HDRPotentialAbs")->GetInt()
          << std::endl;

pDevice->GetRemoteNode("HDRIndex")->SetInt(1);

std::cout << "HDRIndex: "
          << pDevice->GetRemoteNode("HDRIndex")->GetInt()
          << std::endl;

pDevice->GetRemoteNode("HDRExposureRatio")->SetInt(45); //t_Exp_1

std::cout << "HDRExposureRatio: "
          << pDevice->GetRemoteNode("HDRExposureRatio")->GetInt()
          << std::endl;

std::cout << "HDRExposureRatioPercent: "
          << pDevice->GetRemoteNode("HDRExposureRatioPercent")->GetDouble()
          << std::endl;

pDevice->GetRemoteNode("HDRPotentialAbs")->SetInt(20); //Pot_1

std::cout << "HDRPotentialAbs: "
          << pDevice->GetRemoteNode("HDRPotentialAbs")->GetInt()
          << std::endl;
```

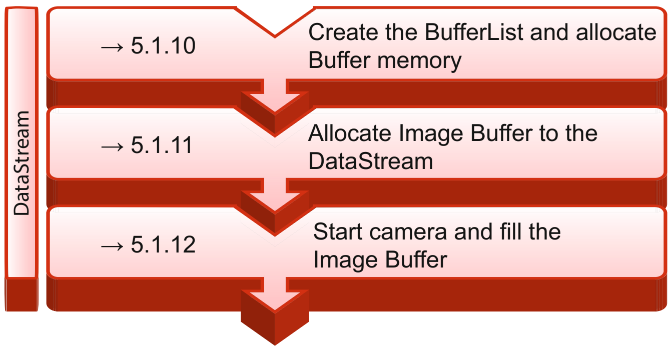
C#

```
System.Console.Write("HDR parameter change\n");

mDevice.RemoteNodeList["HDREnable"].Value = true;

System.Console.Write("  HDREnable : {0}\n",
   (bool)mDevice.RemoteNodeList["HDREnable"].Value);



mDevice.RemoteNodeList["HDRIndex"].Value = (long)0;

System.Console.Write("HDRIndex : {0}\n",
   (long)mDevice.RemoteNodeList["HDRIndex"].Value);

mDevice.RemoteNodeList["HDRExposureRatio"].Value = (long)185; //t_Exp_0

System.Console.Write("HDRExposureRatio : {0}\n",
   (long)mDevice.RemoteNodeList["HDRExposureRatio"].Value);

System.Console.Write("HDRExposureRatioPercent : {0}\n",
   (double)mDevice.RemoteNodeList["HDRExposureRatioPercent"].Value);

mDevice.RemoteNodeList["HDRExposureRatio"].Value = (long)40; //Pot_0

System.Console.Write("HDRPotentialAbs : {0}\n",
   (long)mDevice.RemoteNodeList["HDRPotentialAbs"].Value);



mDevice.RemoteNodeList["HDRIndex"].Value = (long)1;

System.Console.Write("HDRIndex : {0}\n",
   (long)mDevice.RemoteNodeList["HDRIndex"].Value);

mDevice.RemoteNodeList["HDRExposureRatio"].Value = (long)45; //t_Exp_1

System.Console.Write("HDRExposureRatio : {0}\n",
   (long)mDevice.RemoteNodeList["HDRExposureRatio"].Value);

System.Console.Write("HDRExposureRatioPercent : {0}\n",
   (double)mDevice.RemoteNodeList["HDRExposureRatioPercent"].Value);

mDevice.RemoteNodeList["HDRPotentialAbs"].Value = (long)20; //Pot_1

System.Console.Write("HDRPotentialAbs : {0}\n",
   (long)mDevice.RemoteNodeList["HDRPotentialAbs"].Value);

System.Console.Write("\n");
```

Output C++ /C#

```
DEVICE PARAMETER SETUP
######################
 ExposureTime                    : 10000

HDR parameter change
          HDREnable              : 1


HDRIndex                         : 0
          HDRExposureRatio       : 185
          HDRExposureRatioPercent   : 72.54
          HDRPotentialAbs        : 40

HDRIndex                         : 1
          HDRExposureRatio       : 45
          HDRExposureRatioPercent   : 17.64
          HDRPotentialAbs        : 20
```

5.4 Buffer Allocation


DataStream
→ 5.1.10
Create the BufferList and allocate Buffer memory
→ 5.1.11
Allocate Image Buffer to the DataStream
→ 5.1.12
Start camera and fill the Image Buffer

Baumer GAPI SDK offers you four options for influencing the buffer. These are described in greater detail the next sections.

- Buffer internal allocation
- Buffer internal allocation with additional user object
- Buffer external AllocationF
- Buffer external allocation with additional user object

5.4.1 Buffer internal allocation

The function `BGAPI2::Buffer::Buffer()` creates a buffer the size of the maximum payload of the camera that is connected. This enables switching between the pixel formats and various image details without the need for size checking or recreating the buffer, since all formats will fit within the buffer.

If a lot of buffer space is required for small image details (ROI); this creates a high memory requirement, although only a small portion of this is actually used.

The following table shows the memory requirement for an internal buffer of a camera with a resolution of 1288 x 960 pixels and pixel formats "BayerRG8", "BayerRG12" and "BGR8":

Pixelformat	BayerRG8	BayerRG12	BGR8
Byte per pixel	1	2	3
PixelCount	1236480	1236480	1236480
PixelBytes	1236480	2472960	3709440
ChunkBytes	272	272	272
TotalBytes	1236752	2473232	3709712
BufferSizeBytes	3709712	3709712	3709712
Utilization internal Buffer [%]	33	66	100

The maximum payload of 3709712 is the result of the memory requirement of the largest pixel format (here BGR8 with 3 bytes per pixel) plus the memory requirement for the optional chunk data (here fixed 272 bytes).

Refer to the example code of *001_ImageCaptureMode_Polling (see chapter „5.1.10 Create the BufferList and allocate Buffer memory“ on page 56).*

5.4.2 Buffer internal allocation with additional user object

The function `BGAPI2::Buffer::Buffer(void * pUserObj)` also creates a buffer the size of the maximum payload of the camera that is connected. Although in addition you can specify a pointer to an individual object.

This makes it possible to couple the internal buffer with an external user buffer.

For example, the internal buffer can receive the RAW images from a camera that only supports pixel formats "BayerRG8", "BayerRG10" and "BayerRG12".

Pixelformat	BayerRG8	BayerRG10	BayerRG12
Byte per pixel	1	2	2
PixelCount	1236480	1236480	1236480
PixelBytes	1236480	2472960	2472960
ChunkBytes	272	272	272
TotalBytes	1236752	2473232	2473232
BufferSizeBytes	2473232	2473232	2473232
Utilization internal Buffer [%]	50	100	100

At a resolution of 1288 x 960 the maximum payload size here would be 2473232 (derived from "BayerRG12" with 2 bytes per pixel plus chunk data) and derived from that the size of an internal buffer also 2473232 bytes.

Since RAW formats often have to be converted into other target formats, such as BGR8 or BGR12, you can couple an external user buffer to this internal buffer, which then accepts the transformed data.

Using `pBuffer->GetMemPtr()` accesses the RAW image and `pBuffer ->GetUserObj()` accesses the memory area in the user buffer. In the example, the external user buffer is of a size that equates to the target pixel format "BGR8", which requires 3 bytes per pixel.

C++

```cpp
#define BUFFERCOUNT 4
unsigned char * pUserBufferPixeltransformation[BUFFERCOUNT];
BGAPI2::Buffer * pBuffer = NULL;
BGAPI2::ImageProcessor * imgProcessor = NULL;
imgProcessor = BGAPI2::ImageProcessor::GetInstance();

// TransformationBuffer BGR8
for(int i=0; i<BUFFERCOUNT; i++)
{
    pUserBufferPixeltransformation[i] = (unsigned char*)new char[(unsigned int)(pDevice
    ->GetRemoteNode("Width")->GetInt() * pDevice->GetRemoteNode("Height")->GetInt() * 3)];
}

bufferList = pDataStream->GetBufferList();
for(int i=0; i<BUFFERCOUNT; i++)
{
    pBuffer = new BGAPI2::Buffer(pUserBufferPixeltransformation[i]);
    bufferList->Add(pBuffer);
}
// Image capture and transformation
pBufferFilled = pDataStream->GetFilledBuffer(1000);
if(pBufferFilled == NULL)
{
     std::cout << "Error: Buffer Timeout after 1000 msec" << std::endl;
}
else
{
     std::cout << " Image " << pBufferFilled->GetFrameID();
     std::cout << " PixelFormat " << pBufferFilled->GetPixelFormat();
     std::cout << " received in memory address " << pBufferFilled->GetMemPtr();
     std::cout << std::endl;

     BGAPI2::Image * pImage = imgProcessor->CreateImage( (bo_uint)pBufferFilled->GetWidth(),
                               (bo_uint)(int)pBufferFilled->GetHeight(),
                               pBufferFilled->GetPixelFormat(),
                               (void*)((unsigned char*)pBufferFilled
                               ->GetMemPtr() + pBufferFilled
                               ->GetImageOffset()),
                               pBufferFilled->GetMemSize());
     bo_uint64 iTransformBuffersize = pImage->GetTransformBufferLength("RGB8");
     pImage->TransformImageToBuffer("RGB8",
     (void*)pBufferFilled->GetUserObj(),
     iTransformBuffersize);
     std::cout << " transformed to RGB8";
     std::cout << " in memory address " << pBufferFilled->GetUserObj();
     std::cout << std::endl;
     pImage->Release();
     pBufferFilled->QueueBuffer();
}
```

```
// Release internal buffers
while( bufferList->size() > 0)
{
    pBuffer = bufferList->begin()->second;
    bufferList->RevokeBuffer(pBuffer);
    delete pBuffer;
}

// Release external user buffers
for(int i=0; i<BUFFERCOUNT; i++)
{
    delete [] pUserBufferPixeltransformation[i];
}
BGAPI2::ImageProcessor::ReleaseInstance();
```

Notice

The `mImage.TransformToBuffer()` function is not available in C#.

5.4.3 Buffer external Allocation

Alternatively the user may allocate the buffer memory on its own using the function `BGAPI2::Buffer:: Buffer( void *pUserBuffer, bo_uint64 uUserBufferSize, void *pUserObj);`

The responsibility to release the allocated memory then remains with the user.

(see code example: "004_PartialScan_ExternalBuffer").

The buffer size can be adjusted to the required payload size. This enables many buffers with small image details (ROI).

The payload size for the current camera parameter settings (ROI or pixel format) can be determined using the query `pDevice->GetRemoteNode(„PayloadSize")->GetInt()`.

Pixelformat	BayerRG8	BayerRG12	BGR8
Byte per pixel	1	2	3
PixelCount	1236480	1236480	1236480
PixelBytes	1236480	2472960	3709440
ChunkBytes	272	272	272
TotalBytes	1236752	2473232	3709712
BufferSizeBytes	1236752	2473232	3709712
Utilization external Buffer [%]	100	100	100

Notice

Changing the payload-size during aquisition requires a Buffer of different size. If the Buffer size does not meet the required payload size image data will be discarded.

Using the maximum payload-size assures to never lose image data.

Detailed code

C++

```
// create external buffers
char * pMemoryBlock[4];
bufferList = pDataStream->GetBufferList();
bo_uint64 payloadsize = pDataStream->GetDefinesPayloadSize() ?
    pDataStream->GetPayloadSize() : pDevice->GetPayloadSize();
for(int i=0; i<4; i++) // 4 buffers using external allocated memory
{
    pMemoryBlock[i] = (char*)new char[(unsigned int)payloadsize];
    pBuffer = new BGAPI2::Buffer(pMemoryBlock[i], payloadsize, NULL);
    bufferList->Add(pBuffer);
    std::cout << " add external buffer ["
              << pBuffer->GetMemSize()
              << "bytes]"<< std::endl;
}

// Release buffers
while( bufferList->size() > 0)
{
    pBuffer = bufferList->begin()->second;
    bufferList->RevokeBuffer(pBuffer);
    delete pBuffer;
}

// Release external buffer memory
for(int i=0; i<4; i++)
{
    delete [] pMemoryBlock[i];
}
```

C#

```
bufferList = mDataStream.BufferList;
long iDevicePayloadSize = (long)mDevice.RemoteNodeList["PayloadSize"].Value;
ulong uPayloadSize = mDataStream.IsDefinedPayloadSize ?
    mDataStream.PayloadSize : (ulong)iDevicePayloadSize;
for (int i = 0; i < 4; i++) // 4 buffers using external allocated memory
{
    IntPtr mUserBuffer = Marshal.AllocHGlobal((int)uPayloadSize);
    LUserBuffer.Add(mUserBuffer);
    IntPtr pUserObj = new IntPtr(0); // NULL pointer, not used
    mBuffer = new BGAPI2.Buffer(mUserBuffer, uPayloadSize, pUserObj);
    bufferList.Add(mBuffer);
    System.Console.Write(" add external buffer [");
    System.Console.Write("{0} bytes]\n", mBuffer.MemSize);
}

// release external buffers
while (bufferList.Count > 0)
{
    mBuffer = (BGAPI2.Buffer)bufferList.Values.First();
    IntPtr mBufferPointer = mBuffer.MemPtr;
    if (LUserBuffer.Contains(mBufferPointer))
    {
        Marshal.FreeHGlobal(mBufferPointer);
    }
    bufferList.RevokeBuffer(mBuffer);
}
```

Output C++ / C#

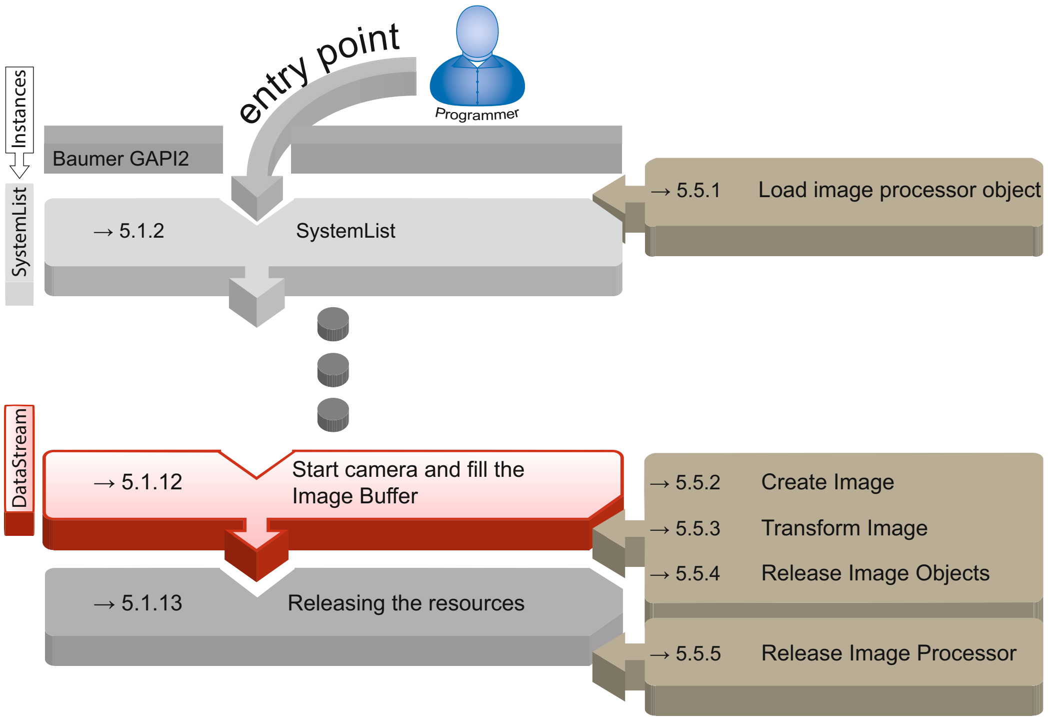
```
BUFFER LIST
###########

5.1.10  Announced buffers:       4
```

5.4.4 Buffer external allocation with additional user object

The function `BGAPI2::Buffer:: Buffer( void *pUserBuffer, bo_uint64 uUserBufferSize, void *pUserObj)` can also be coupled for external buffer allocation with a user object `pUserObj` as a pointer to the buffer.

5.5 Image transformation

The Baumer GAPI Image Library offers a image transformation option. The pixel transformation proceeds in the following steps.



List of important pixel transformations of Baumer image processor:

	BayerBG10	BayerBG12	BayerBG8	BayerGB10	BayerGB12	BayerGB8	BayerGR10	BayerGR12	BayerGR8	BayerRG10	BayerRG12	BayerRG8	BGR10	BGR12	BGR16	BGR8	Mono10	Mono14	Mono16	Mono8	RGB10_Planar	RGB12	RGB12_Planar	RGB16	RGB16_Planar	RGB8	RGB8_Planar
BayerBG10	*		*										*			*	*			*	*					*	*
BayerBG10p	*		*										*			*	*			*	*					*	*
BayerBG10Packed	*		*										*			*	*			*	*					*	*
BayerBG12	*	*	*										*	*		*	*			*	*	*	*			*	*
BayerBG12p	*	*	*										*	*		*	*			*	*	*	*			*	*
BayerBG12Packed	*	*	*										*	*		*	*			*	*	*	*			*	*
BayerBG16	*	*	*																								
BayerBG8			*													*				*						*	*
BayerGB10				*		*							*			*	*			*	*					*	*
BayerGB10p				*		*							*			*	*			*	*					*	*
BayerGB10Packed				*		*							*			*	*			*	*					*	*
BayerGB12				*	*	*							*	*		*	*			*	*	*	*			*	*
BayerGB12p				*	*	*							*	*		*	*			*	*	*	*			*	*
BayerGB12Packed				*	*	*							*	*		*	*			*	*	*	*			*	*
BayerGB16				*	*	*																					
BayerGB8						*										*				*						*	*
BayerGR10							*		*				*			*	*			*	*					*	*
BayerGR10p							*		*				*			*	*			*	*					*	*
BayerGR10Packed							*		*				*			*	*			*	*					*	*
BayerGR12							*	*	*				*	*		*	*			*	*	*	*			*	*
BayerGR12p							*	*	*				*	*		*	*			*	*	*	*			*	*
BayerGR12Packed							*	*	*				*	*		*	*			*	*	*	*			*	*
BayerGR16							*	*	*																		
BayerGR8									*							*				*						*	*
BayerRG10										*		*	*			*	*			*	*					*	*
BayerRG10p										*		*	*			*	*			*	*					*	*
BayerRG10Packed										*		*	*			*	*			*	*					*	*
BayerRG12										*	*	*	*	*		*	*			*	*	*	*			*	*
BayerRG12p										*	*	*	*	*		*	*			*	*	*	*			*	*
BayerRG12Packed										*	*	*	*	*		*	*			*	*	*	*			*	*
BayerRG16										*	*	*															
BayerRG8												*				*				*						*	*
BGR10													*			*	*			*	*					*	*
BGR12													*	*		*	*			*	*	*	*			*	*
BGR16													*	*	*	*	*	*	*	*	*	*	*	*		*	*
BGR8																*				*						*	*
Mono10													*			*	*			*	*					*	*
Mono10p													*			*	*			*	*					*	*
Mono10Packed													*			*	*			*	*					*	*
Mono14													*	*		*	*	*		*	*	*	*			*	*
Mono16													*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Mono8																*				*						*	*
RGB10_Planar													*			*	*			*	*					*	*
RGB12													*	*		*	*			*	*	*	*			*	*
RGB12_Planar													*	*		*	*			*	*	*	*			*	*
RGB16													*	*	*	*	*	*	*	*	*	*	*	*		*	*
RGB16_Planar													*	*		*	*	*	*	*	*	*	*		*	*	*
RGB8																*				*						*	*
RGB8_Planar																*				*						*	*
YUV422_8_UYVY																*				*						*	*
YUV8_UYV																*				*						*	*

⬜ Transforming not possible

* Transforming possible

5.5.1 Load Image Processor object

Pixel transformation is done by the image library. The image processor must first be load in the program.

Detailed code

C++

```
// Variables for image processor
BGAPI2::ImageProcessor * imgProcessor = NULL;
BGAPI2::Image * pImage = NULL;
BGAPI2::Image * pTransformImage = NULL;

// Load the image processor in the beginning of the program.
try
{
     imgProcessor = new BGAPI2::ImageProcessor();
     std::cout << "Version: " << imgProcessor->GetVersion() << std::endl;
}
catch (BGAPI2::Exceptions::IException& ex)
{
     std::cout << "ExceptionType: " << ex.GetType() << std::endl;
     std::cout << "ErrorDescription: " << ex.GetErrorDescription() << std::endl;
     std::cout << "in function: " << ex.GetFunctionName() << std::endl;
}
```

C#

```
// Variables for image processor
BGAPI2.ImageProcessor imgProcessor = null;
BGAPI2.Image mImage = null;
BGAPI2.Image mTransformImage = null;

// Load the image processor in the beginning of the program.
try
{
     img.Processor = new BGAPI2.ImageProcessor();
     System.Console.Write("Version: {0}\n", imgProcessor.GetVersion());
}
catch (BGAPI2.Exceptions.IException ex)
{
     System.Console.Write("ExceptionType: {0}\n", ex.GetType());
     System.Console.Write("ErrorDescription: {0}\n", ex.GetErrorDescription());
     System.Console.Write("in function: {0}\n",ex.GetFunctionName());
}
```

Output C++ / C#

```
IMAGE PROCESSOR
###############

Version:        2.3.4861.5273
```

5.5.2 Create Image

If the buffer object is filled with image data then an image object can be created using the information from the buffer about the image details, like width, height, pixelformat, image-buffer pointer, and memory size.

Detailed code

C++

```
BGAPI2::Buffer * pBufferFilled = NULL;
pBufferFilled = pDataStream->GetFilledBuffer(1000);

pImage = imgProcessor->CreateImage( (bo_uint)pBufferFilled->GetWidth(),
                                   (bo_uint)(int)pBufferFilled->GetHeight(),
                                   pBufferFilled->GetPixelFormat(),
                                   pBufferFilled->GetMemPtr(),
                                   pBufferFilled->GetMemSize() );


// Get the information of the image object.
std::cout << "pImage.Pixelformat: " << pImage->GetPixelformat() << std::endl;
std::cout << "pImage.Width:       " << pImage->GetWidth() << std::endl;
std::cout << "pImage.Height:      " << pImage->GetHeight() << std::endl;
std::cout << "pImage.Buffer:      " << pImage->GetBuffer() << std::endl;
```

C#

```
BGAPI2.Buffer mBufferFilled = null;
mBufferFilled = mDataStream.GetFilledBuffer(1000);

mImage = imgProcessor.CreateImage((uint)mBufferFilled.Width,
                                 (uint)mBufferFilled.Height,
                                 (string)mBufferFilled.PixelFormat,
                                 mBufferFilled.MemPtr,
                                 (ulong)mBufferFilled.MemSize);


// Get the information of the image object.
System.Console.Write("mImage.Pixelformat: {0}\n", mImage.PixelFormat);
System.Console.Write("mImage.Width:       {0}\n", mImage.Width);
System.Console.Write("mImage.Height:      {0}\n", mImage.Height);
System.Console.Write("mImage.Buffer:      {0:X8}\n", (ulong)mImage.Buffer);
```

Output C++ / C#

```
IMAGE
#####

pImage.Pixelformat:          BayerRG8
pImage.Width:                2040
pImage.Height:               1084
pImage.Buffer:              0000000004B00040
```

5.5.3 Transform Image

The image object can be transformed to another pixelformat (e.g. to display the image on a monitor the image need to be transformed to Mono8 in case of monochrome image).

Detailed Code

C++

```
pTransformImage = imgProcessor->CreateTransformedImage(pImage, "BGR8");


// Get the information of the transformed image object.
std::cout << "pTransformImage.Pixelformat: " << pTransformImage->GetPixelformat();
std::cout << std::endl;
std::cout << "pTransformImage.Width:       " << pTransformImage->GetWidth() << std::endl;
std::cout << "pTransformImage.Height:      " << pTransformImage->GetHeight() << std::endl;
std::cout << "pTransformImage.Buffer:      " << pTransformImage->GetBuffer() << std::endl;

// The transformed image data can be accessed by a new buffer pointer.
```

C#

```
mTransformImage = imgProcessor.CreateTransformedImage(mImage, "BGR8");


// Get the information of the transformed image object.
System.Console.Write("mTransformImage.Pixelformat: {0}\n", mTransformImage.PixelFormat);
System.Console.Write("mTransformImage.Width:       {0}\n", mTransformImage.Width);
System.Console.Write("mTransformImage.Height:      {0}\n", mTransformImage.Height);
System.Console.Write("mTransformImage.Buffer:      {0}\n", (ulong)mTransformImage.Buffer);

// The transformed image data can be accessed by a new buffer pointer.
```

Output C++ / C#

```
TRANSFORMED IMAGE
#################

pTransformImage.Pixelformat:        BGR8
pTransformImage.Width:              2040
pTransformImage.Height:             1084
pTransformImage.Buffer:            00000000066D0040
```

Notice

In C# create a bitmap using the `mTransformImage.Buffer` pointer to display or save the image. This function is not available with C++.

C#

```
// need to add the reference System.Drawing
using System.Drawing;
System.Drawing.Bitmap bitmap;

if (mTransformImage.Pixelformat == "Mono8")

{
     bitmap = new System.Drawing.Bitmap( (int)mTransformImage.Width,
                              (int)mTransformImage.Height,
                              (int)mTransformImage.Width * 1,
                              System.Drawing.Imaging.PixelFormat.Format8bppIndexed,
                              mTransformImage.Buffer);
     System.Drawing.Imaging.ColorPalette palette = bitmap.Palette;
     int nColors = 256;
     for (int i = 0; i < nColors; i++)
     {
             uint Alpha = 0xFF;
             uint Intensity = (uint)(i * 0xFF / (nColors - 1));
             palette.Entries[i] = Color.FromArgb(
             (int)Alpha,
             (int)Intensity,
             (int)Intensity,
             (int)Intensity);
     }
     bitmap.Palette = palette;
     //to display use: this.pictureBoxImageDisplay.Image = bitmap;
     bitmap.Save("image.bmp");
}


if (mTransformImage.Pixelformat == "BGR8")
{
     bitmap = new System.Drawing.Bitmap((int)mTransformImage.Width,
                              (int)mTransformImage.Height,
                              (int)mTransformImage.Width * 3,
                              System.Drawing.Imaging.PixelFormat.Format24bppRgb,
                              mTransformImage.Buffer);
     //to display use: this.pictureBoxImageDisplay.Image = bitmap;
     bitmap.Save("image.bmp");
}
```

5.5.4 Release Image Objects

Release the image and transformed image objects after using.

C++

```
pImage->Release();
pTransformImage->Release();
```

C#

```
mImage.Release();
mTransformImage.Release();
```

5.5.5 Release Image Processor

At the end of the C++ program release the image processor instance.

C++

```
delete imgProcessor;
```

Notice

In the case of C#, the image processor is deleted when the program is closed.

5.6 Device Events

Device Events are used to transmit events that occur (e.g. ExposureStart, ExposureEnd) to the software.

5.6.1 Device Events in Polling Mode

The asynchronous message channel is described in the respecitve standard (GigE Vision® / USB 3.0 Vision™) and offers the possibility of event signaling. There is a timestamp (64 bits) for each announced event, which contains the accurate time at which the event occurred. Each event can be activated and deactivated separately.

For example, set the device events "ExposureStart" and "ExposureEnd" in the camera from "Off" to "On".

Notice
On some cameras use the outdated term "GigEVisionEvent" instead of "On".

Detailed code

C++

```
pDevice->GetRemoteNode("EventSelector")->SetValue("ExposureStart");
pDevice->GetRemoteNode("EventNotification")->SetValue("On");

pDevice->GetRemoteNode("EventSelector")->SetValue("ExposureEnd");
pDevice->GetRemoteNode("EventNotification")->SetValue("On");


// List up all events and their status.
BGAPI2::NodeMap * nmEventList;
nmEventList = pDevice->GetRemoteNode("EventSelector")->GetEnumNodeList();
std::cout << "EventListCount: " << nmEventList->GetNodeCount() << std::endl;

BGAPI2::String sEventItem;
for(bo_uint64 i = 0; i < nmEventList->GetNodeCount(); i++)
{
    std::cout << i << ": ";
    sEventItem = nmEventList->GetNodeByIndex(i)->GetValue();
    std::cout << sEventItem << " ";

    // check message is ON or OFF
    pDevice->GetRemoteNode("EventSelector")->SetValue(sEventItem);
    std::cout << pDevice->GetRemoteNode("EventNotification")->GetValue();
    std::cout << std::endl;
}
```

```
// Set the event mode to EVENTMODE_POLLING.
pDevice->RegisterDeviceEvent(BGAPI2::Events::EVENTMODE_POLLING);
BGAPI2::Events::EventMode currentEventMode = pDevice->GetEventMode();
BGAPI2::String sCurrentEventMode = "";
switch(currentEventMode)
{
     case BGAPI2::Events::EVENTMODE_POLLING:
               sCurrentEventMode = "EVENTMODE_POLLING";
               break;
     case BGAPI2::Events::EVENTMODE_DISABLED:
               sCurrentEventMode = "EVENTMODE_DISABLED";
               break;
     case BGAPI2::Events::EVENTMODE_EVENT_HANDLER:
               sCurrentEventMode = "EVENTMODE_EVENT_HANDLER";
               break;
     default:
               sCurrentEventMode = "EVENTMODE_UNKNOWN";
}
std::cout << "Register Event Mode to: " << sCurrentEventMode << std::endl;



TriggerSource = Software
TriggerMode = On

// Enable SoftwareTrigger
pDevice->GetRemoteNode("TriggerSource")->SetString("Software");
std::cout << "TriggerSource: "
          << pDevice->GetRemoteNode("TriggerSource")->GetString()
          << std::endl;
pDevice->GetRemoteNode("TriggerMode")->SetString("On");
std::cout<<"TriggerMode:    "
         <<pDevice->GetRemoteNode("TriggerMode)->GetString()
         << std::endl;
```

C#

```
mDevice.RemoteNodeList["EventSelector"].Value = "ExposureStart";
mDevice.RemoteNodeList["EventNotification"].Value = "On";

mDevice.RemoteNodeList["EventSelector"].Value = "ExposureEnd";
mDevice.RemoteNodeList["EventNotification"].Value = "On";

// List up all events and their status.
BGAPI2.NodeMap nmEventList;
nmEventList = mDevice.RemoteNodeList["EventSelector"].EnumNodeList;
System.Console.Write("EventListCount: {0}\n", nmEventList.Count);

string sEventItem;
for (ulong i = 0; i < nmEventList.Count; i++)
{
     System.Console.Write("{0}: ", i);
     sEventItem = (string)nmEventList[i].Value;
     System.Console.Write("{0} ", sEventItem);

     // check message is ON or OFF
     mDevice.RemoteNodeList["EventSelector"].Value = sEventItem;
     System.Console.Write("{0}\n",(string)mDevice.RemoteNodeList["EventNotification"].Value);
}
// Set the event mode to EVENTMODE_POLLING.
mDevice.RegisterDeviceEvent(BGAPI2.Events.EventMode.POLLING);
BGAPI2.Events.EventMode currentEventMode = mDevice.EventMode;
string sCurrentEventMode = "";
switch (currentEventMode)
{
     case BGAPI2.Events.EventMode.POLLING:
             sCurrentEventMode = "EVENTMODE_POLLING";
             break;
     case BGAPI2.Events.EventMode.DISABLED:
             sCurrentEventMode = "EVENTMODE_DISABLED";
             break;
     case BGAPI2.Events.EventMode.EVENT_HANDLER:
             sCurrentEventMode = "EVENTMODE_EVENT_HANDLER";
             break;
     default:
             sCurrentEventMode = "EVENTMODE_UNKNOWN";
             break;
}
```

```
System.Console.Write("Register Event Mode to: {0}\n", sCurrentEventMode);

TriggerSource = Software
TriggerMode = On

// Enable SoftwareTrigger
mDevice.RemoteNodeList["TriggerSource"].Value = "Software";
        System.Console.Write("TriggerSource: {0}\n",
(string)mDevice.RemoteNodeList["TriggerSource"].Value);
        mDevice.RemoteNodeList["TriggerMode"].Value = "On";
System.Console.Write("TriggerMode:   {0}\n",
       (string)mDevice.RemoteNodeList["TriggerMode"].Value);
```

Output C++ / C#

```
Device Events
#############

EventSelector:           ExposureStart
EventNotification:       On

EventSelector:           ExposureEnd
EventNotification:       On

EventSelector list count: 29
                      0:  Action1                       Off
                      1:  EventDiscarded                Off
                      2:  EventLost                     Off
                      3:  ExposureEnd                   On
                      4:  ExposureStart                 On
                      5:  FrameEnd                      Off
                      6:  FrameStart                    Off
                      7:  GigEVisionError               Off
                      8:  GigEVisionHeartbeatTimeOut    Off
                      9:  Line0FallingEdge              Off
                      10:  Line0RisingEdge              Off
                      11:  Line1FallingEdge             Off
                      12:  Line1RisingEdge              Off
                      13:  Line2FallingEdge             Off
                      14:  Line2RisingEdge              Off
                      15:  Line3FallingEdge             Off
                      16:  Line3RisingEdge              Off
                      17:  PrimaryApplicationSwitch     Off
                      26:  TriggerOverlapped            Off
                      27:  TriggerReady                 Off
                      28:  TriggerSkipped               Off

Register Event Mode to:   EVENTMODE_POLLING
```

The following sequence is used to demonstrate the device events. Refer to example code *006_DeviceEventMode_Polling.*

1. Get TimestampTickFrequency of the camera for reference of EventTimestamp.
2. Set ExposureTime.
3. Do SoftwareTrigger.
4. Wait for Event ExposureStart.
5. Wait for Event ExposureEnd.
6. Calculate the Timestamp difference of ExposureEnd - ExposureStart.
7. Wait for ImageBuffer filled.

Detailed code

C++

```cpp
BGAPI2::Buffer * pBufferFilled = NULL;
bo_double fExposureValue = 20000;
bo_double fTimestampTickFrequency = 1.0;
bo_double fTimestampExposureStart = 0.0;
bo_double fTimestampExposureEnd = 0.0;
bo_double fTimestampDiff = 0.0;
BGAPI2::Events::DeviceEvent* pdEvent = new BGAPI2::Events::DeviceEvent();

// 1. TIMESTAMP TICK FREQUENCY
if(pDevice->GetRemoteNodeList()->GetNodePresent("GevTimestampTickFrequency"))
    {
            fTimestampTickFrequency = (bo_double)pDevice->GetRemoteNode
            ("GevTimestampTickFrequency")->GetInt();
    }
    else
    {
            fTimestampTickFrequency = 1000000000.0;
    }

// 2. SET EXPOSURE TIME
pDevice->GetRemoteNode("ExposureTime")->SetDouble(fExposureValue);
std::cout << "Set Exposure to: ";
std::cout << pDevice->GetRemoteNode("ExposureTime")->GetDouble();
std::cout << " [usec]" << std::endl;

// 3. DO SOFTWARE TRIGGER
std::cout << "Execute TriggerSoftware " << std::endl;
pDevice->GetRemoteNode("TriggerSoftware")->Execute();

// 4. WAIT FOR EVENT EXPOSURE START
if(pDevice->GetDeviceEvent(pdEvent, 1000) == true)
{
    fTimestampExposureStart = (bo_double)pdEvent->GetTimeStamp() / fTimestampTickFrequency;
    fTimestampExposureStart = fTimestampExposureStart * 1000000.0;
    std::cout << "EventID " << pdEvent->GetId() << " " << pdEvent->GetName();
    std::cout << " Timestamp " << fTimestampExposureStart << " [usec]" << std::endl;
}
else
{
    std::cout << "Error: GetDeviceEvent Timeout after 1000 msec" << std::endl;
}
```

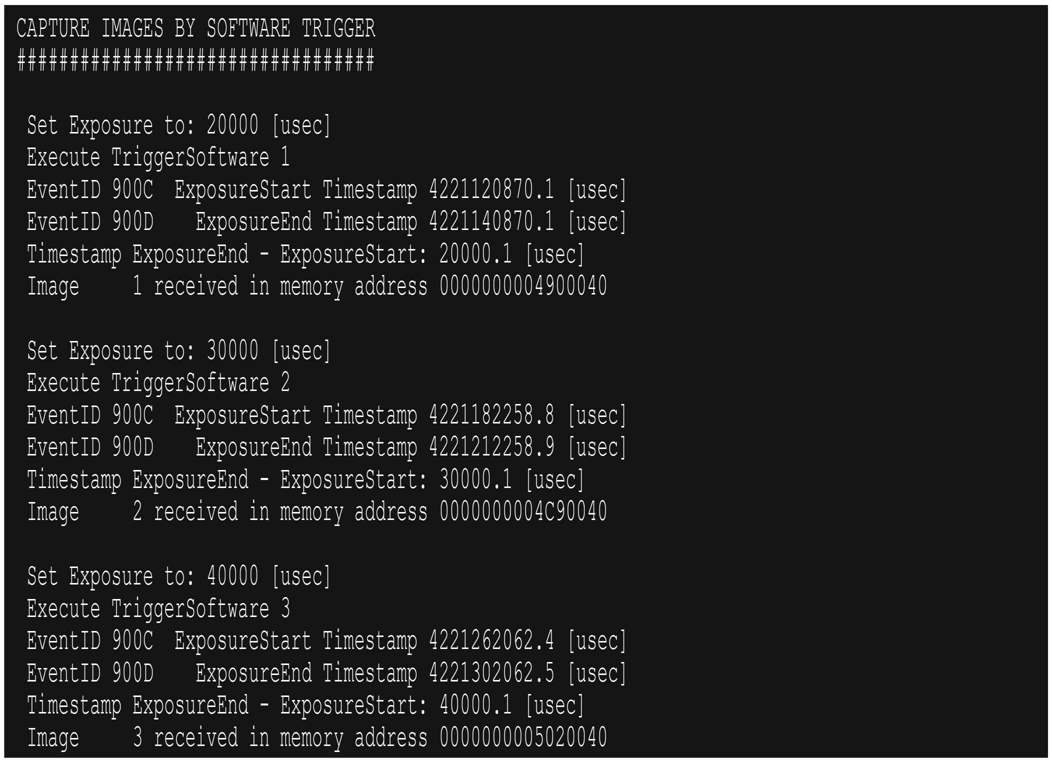
```
// 5. WAIT FOR EVENT EXPOSURE END
if(pDevice->GetDeviceEvent(pdEvent, 1000) == true)
{
     fTimestampExposureEnd = (bo_double)pdEvent->GetTimeStamp() / fTimestampTickFrequency;
     fTimestampExposureEnd = fTimestampExposureEnd * 1000000.0;
     std::cout << "EventID " << pdEvent->GetId() << " " << pdEvent->GetName();
     std::cout << " Timestamp " << fTimestampExposureEnd << " [usec]" << std::endl;
}
else
{
     std::cout << "Error: GetDeviceEvent Timeout after 1000 msec" << std::endl;
}

// 6. CALCULATE TIME DIFFERENCE EXPOSURE END - START
fTimestampDiff = fTimestampExposureEnd - fTimestampExposureStart;
std::cout << "Timestamp ExposureEnd - ExposureStart: " << fTimestampDiff << " [usec]";
std::cout << std::endl;

// 7. WAIT FOR IMAGE BUFFER IS FILLED
pBufferFilled = pDataStream->GetFilledBuffer(1000);
if(pBufferFilled == NULL)
{
     std::cout << "Error: Buffer Timeout after 1000 msec" << std::endl;
}
else
{
     std::cout << "Image " << pBufferFilled->GetFrameID() << " received" << std::endl;
     pBufferFilled->QueueBuffer();
}

// Reset the event mode to EventMode_Disabled.
pDevice->UnregisterDeviceEvent();
currentEventMode = pDevice->GetEventMode();
delete pdEvent;

// Set the device events "ExposureStart" and "ExposureEnd" back to "Off" in the camera.
pDevice->GetRemoteNode("EventSelector")->SetValue("ExposureStart");
pDevice->GetRemoteNode("EventNotification")->SetValue("Off");

pDevice->GetRemoteNode("EventSelector")->SetValue("ExposureEnd");
pDevice->GetRemoteNode("EventNotification")->SetValue("Off");
pDevice->GetRemoteNode("TriggerMode")->SetValue("Off");
```

C#

```
BGAPI2.Events.DeviceEvent mdEvent = new BGAPI2.Events.DeviceEvent();
BGAPI2.Buffer mBufferFilled = null;
double fExposureValue = 20000;
double fTimestampTickFrequency = 1.0;
double fTimestampExposureStart = 0.0;
double fTimestampExposureEnd = 0.0;
double fTimestampDiff = 0.0;

// 1. TIMESTAMP TICK FREQUENCY
if (mDevice.RemoteNodeList.GetNodePresent("GevTimestampTickFrequency") == true)
     {
             fTimestampTickFrequency =
             (double)((long)mDevice.RemoteNodeList["GevTimestampTickFrequency"].Value);
     }
     else
     {
             fTimestampTickFrequency = 1000000000.0;
     }

// 2. SET EXPOSURE TIME
mDevice.RemoteNodeList["ExposureTime"].Value = fExposureValue;
System.Console.Write("Set Exposure to: {0} [usec] \n",
                     (double)mDevice.RemoteNodeList["ExposureTime"].Value);

// 3. DO SOFTWARE TRIGGER
System.Console.Write("Execute TriggerSoftware\n");
mDevice.RemoteNodeList["TriggerSoftware"].Execute();


// 4. WAIT FOR EVENT EXPOSURE START
if (mDevice.GetDeviceEvent(ref mdEvent, 1000) == true)
{
     fTimestampExposureStart = (double)((ulong)mdEvent.Timestamp) / fTimestampTickFrequency;
     fTimestampExposureStart = fTimestampExposureStart * 1000000.0; // in usec
     System.Console.Write(" EventID {0} {1} ", mdEvent.ID, mdEvent.Name);
     System.Console.Write(" Timestamp {0:F1} [usec]\n", fTimestampExposureStart);
}
else
{
     System.Console.Write("Error: GetDeviceEvent Timeout after 1000 msec\n");
}
```

```
// 5. WAIT FOR EVENT EXPOSURE END
if (mDevice.GetDeviceEvent(ref mdEvent, 1000) == true)
{
     fTimestampExposureEnd = (double)((ulong)mdEvent.Timestamp) / fTimestampTickFrequency;
     fTimestampExposureEnd = fTimestampExposureEnd * 1000000.0; // in usec
     System.Console.Write(" EventID {0} {1} ", mdEvent.ID, mdEvent.Name);
     System.Console.Write(" Timestamp {0:F1} [usec]\n", fTimestampExposureEnd);
}


// 6. CALCULATE TIME DIFFERENCE EXPOSURE END - START
fTimestampDiff = fTimestampExposureEnd - fTimestampExposureStart;
System.Console.Write("Timestamp ExposureEnd - ExposureStart: {0:F1}", fTimestampDiff);
System.Console.Write(" [usec]\n");

// 7. WAIT FOR IMAGE BUFFER IS FILLED
mBufferFilled = mDataStream.GetFilledBuffer(1000); // timeout 1000 msec
if (mBufferFilled == null)
{
     System.Console.Write("Error: Buffer Timeout after 1000 msec\n");
}
else
{
     System.Console.Write("Image {0} received\n", mBufferFilled.FrameID);
     mBufferFilled.QueueBuffer();
}

// Reset the event mode to EventMode_Disabled.
mDevice.UnregisterDeviceEvent();
mDevice.UnregisterDeviceEvent();
currentEventMode = mDevice.EventMode;

// Set the device events "ExposureStart" and "ExposureEnd" back to "Off" in the camera.
mDevice.RemoteNodeList["EventSelector"].Value = "ExposureStart";
mDevice.RemoteNodeList["EventNotification"].Value = "Off";

mDevice.RemoteNodeList["EventSelector"].Value = "ExposureEnd";
mDevice.RemoteNodeList["EventNotification"].Value = "Off";
mDevice.RemoteNodeList["TriggerMode"].Value = "Off";
```

Output C++ / C#

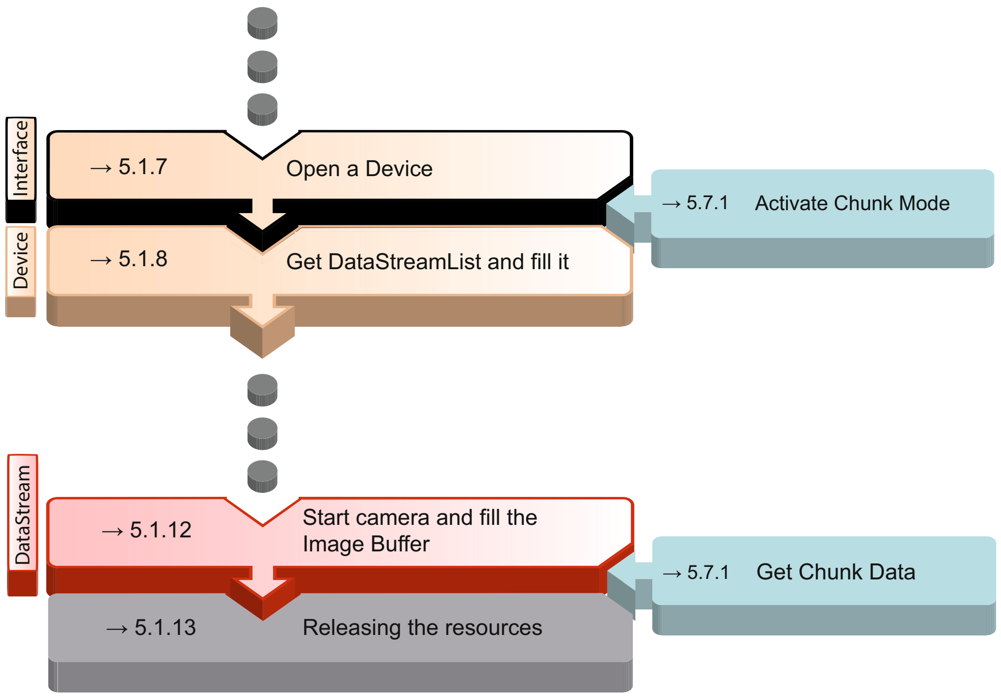


```
CAPTURE IMAGES BY SOFTWARE TRIGGER
##################################

 Set Exposure to: 20000 [usec]
 Execute TriggerSoftware 1
 EventID 900C  ExposureStart Timestamp 4221120870.1 [usec]
 EventID 900D    ExposureEnd Timestamp 4221140870.1 [usec]
 Timestamp ExposureEnd - ExposureStart: 20000.1 [usec]
 Image     1 received in memory address 0000000004900040

 Set Exposure to: 30000 [usec]
 Execute TriggerSoftware 2
 EventID 900C  ExposureStart Timestamp 4221182258.8 [usec]
 EventID 900D    ExposureEnd Timestamp 4221212258.9 [usec]
 Timestamp ExposureEnd - ExposureStart: 30000.1 [usec]
 Image     2 received in memory address 0000000004C90040

 Set Exposure to: 40000 [usec]
 Execute TriggerSoftware 3
 EventID 900C  ExposureStart Timestamp 4221262062.4 [usec]
 EventID 900D    ExposureEnd Timestamp 4221302062.5 [usec]
 Timestamp ExposureEnd - ExposureStart: 40000.1 [usec]
 Image     3 received in memory address 0000000005020040
```

5.7 Chunk

The Chunk Mode is a special kind to structure the transferred data of the camera. With the chunk mode, it is possible to transfer additional meta information with an image.

This chapter describes the structure and the access to the chunk data.



Other information besides the actual image data, so-called metadata, is also transmitted by default. This information is located in the leader and could for instance be:

Leader	
Block ID (FrameID)	Offset X
Timestamp	Offset Y
PixelFormat	Padding X
Width	Padding Y
Height	

Further information can be found in the chunk. This depends on the camera and can for instance contain:

Chunk	
The same information as in the leader	
Timestamp	Height
PixelFormat	Offset X
Width	Offset Y
Additional information	
FrameID (hardware counter)	RegionID
ExposureTime	ReverseX
Gain	ReverseY
BinningHorizontal	DeviceTemperature
BinningVertical	LineStatus (Inputs only)
BlackLevel	BoSequencerEnable
FixedPatternNoiseCorrection	BrightnessCorrection

For the control of the Chunk-Modes there is in the XML file the category *ChunkDataControl*. This category contains the features *ChunkModeActive*, *ChunkSelector*, and *ChunkEnable*.

The feature *ChunkModeActive* activates the Chunk-Mode in the camera and ensures that the transmitted data corresponds to the defined chunk structure.

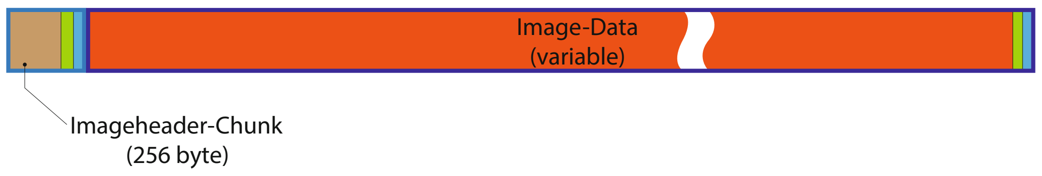
With the features *ChunkSelector* and *ChunkEnable* can seperate Chunk-Blocks activated or deactivated.

TXG-Series

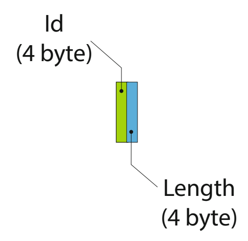
These cameras supports the Image-Data and the Imageheader-Chunk, whereat these cameras transfer first the Imageheader-Chunk and afterwards the image data.

The following figure shows the structure of a chunk of these camera.

Chunk TXG



Notice

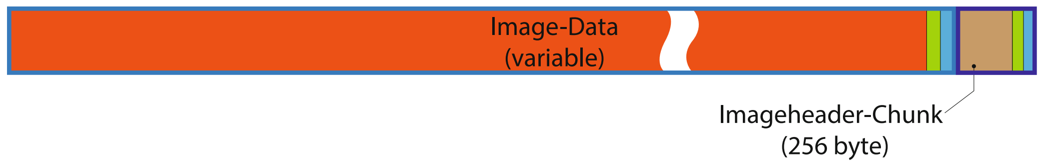


HXG/SXG-Series, MX-Series/VisiLine-Series, PXU and VCX-Series

These cameras supports the Image-Data and the Imageheader-Chunk too. Here first the Image-Data and afterwards the Imageheader-Chunk will transferered.

The following figure shows the structure of the chunk of these cameras.

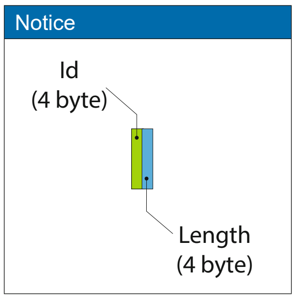
HXG/SXG-Series, MX-Series/VisiLine-Series and PXU-Series

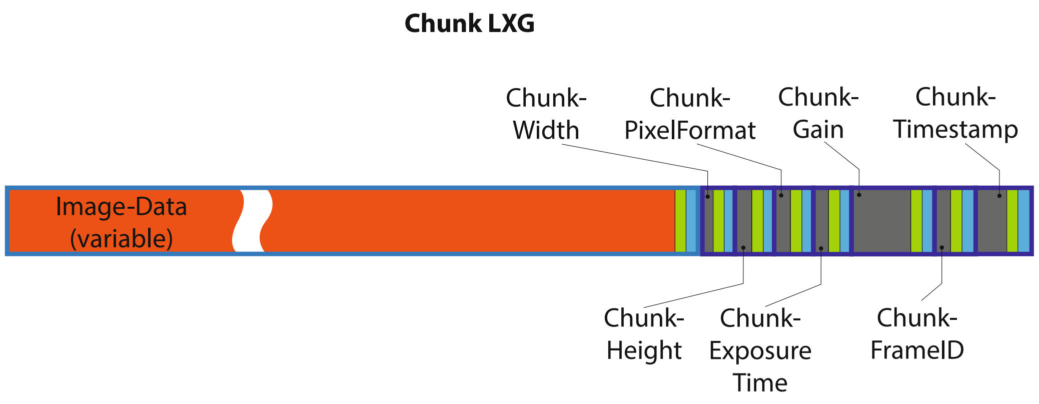


LXG-Series

Baumer LXG supports Extended Chunk Mode. All chunk data are transferred individually as Extended Chunk according to the image data. The Extended Chunk data can be selected individually and also transferred without image data.

The following figure shows the structure of the Extended Chunk.





LXG has available the chunk data shown in the following table. These can be set individually. The output sequence for chunk data is according to ChunkID.

ChunkID	**ChunkData**
0x000	Image
0x100	ChunkOffsetX
0x101	ChunkOffsetY
0x102	ChunkWidth
0x103	ChunkHeight
0x104	ChunkPixelFormat
0x105	ChunkBinningHorizontal
0x106	ChunkBinningVertical
0x107	ChunkImageControl ChunkBrightnessCorrection ChunkDefectPixelCorrection ChunkLUTEnable ChunkReverseX ChunkReverseY ChunkFixedPatternNoiseCorrection
0x200	ChunkExposureTime
0x300	ChunkBlackLevel
0x301	ChunkGain
0x400	ChunkBOSequencerEnable
0x500	ChunkFrameID
0x501	ChunkTimestamp
0x502	ChunkRegionID

5.7.1 Activate Chunk Mode

The following example "007_Chunk" is based on the simple example "001_ImageCaptureModePolling".

See the following overview of changes in the sequence, especially in "Device Parameter Setup" and "Image Capture Loop".

The necessary changes in "Device Parameter Setup" are following in detail here.

5.7.1.1 Activate Imageheader-Chunk

Detailed code (HXG, MXG, MXU, Visiline, PXU)

Notice

In case of TXG series use `FreeformatHeader2` instead of `BaumerImageHeader3`.

In case of SXG series use `FreeformatHeader3`.

C++

```cpp
if(pDevice->GetRemoteNode("ChunkSelector")->GetEnumNodeList()
          ->GetNodePresent("BaumerImageHeader3"))
{
     // ACTIVATE CHUNK
     pDevice->GetRemoteNode("ChunkModeActive")->SetBool(true);
     std::cout << "ChunkModeActive:"
               << pDevice->GetRemoteNode("ChunkModeActive")->GetValue()
               << std::endl;
     // ENABLE ChunkSelector "BaumerImageHeader3"
     pDevice->GetRemoteNode("ChunkSelector")->SetString("BaumerImageHeader3");
     std::cout << "ChunkSelector:"
               << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
               << std::endl;
     if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
     {
         pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
     }
     std::cout << "ChunkEnable:"
               << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
               << std::endl;
     // ENABLE ChunkSelector "Image"
     pDevice->GetRemoteNode("ChunkSelector")->SetString("Image");
     std::cout << "ChunkSelector:"
               << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
               << std::endl;
     if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
     {
         pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
     }
```

```
        std::cout << "ChunkEnable:"
                  << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
                  << std::endl;
    }
```

C#

```
    if (mDevice.RemoteNodeList["ChunkSelector"].EnumNodeList.GetNodePresent("BaumerImageHeader3")
        == true)
    {
        // ACTIVATE CHUNK
        mDevice.RemoteNodeList["ChunkModeActive"].Value = true;
        System.Console.Write("ChunkModeActive:{0}\n",
                             (bool)mDevice.RemoteNodeList["ChunkModeActive"].Value);

        // ENABLE ChunkSelector "BaumerImageHeader3"
        mDevice.RemoteNodeList["ChunkSelector"].Value = "BaumerImageHeader3";
        System.Console.Write("ChunkSelector:{0}\n",
                             (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
        if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
        {
            mDevice.RemoteNodeList["ChunkEnable"].Value = true;
        }
        System.Console.Write("ChunkEnable:{0}\n",
                             (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);
        // ENABLE ChunkSelector "Image"
        mDevice.RemoteNodeList["ChunkSelector"].Value = "Image";
        System.Console.Write("ChunkSelector:{0}\n",
                             (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
        if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
        {
            mDevice.RemoteNodeList["ChunkEnable"].Value = true;
        }
        System.Console.Write("ChunkEnable:{0}\n",
                             (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);
        System.Console.Write("\n");
    }
```

5.7.1.2 Activate Extended Chunk (setting all available chunk data)

This sample code sets all available chunk data of the extended chunk.

C++

```cpp
pDevice->GetRemoteNode("ChunkModeActive")->SetBool(true);
std::cout << "ChunkModeActive:  "
          << pDevice->GetRemoteNode("ChunkModeActive")->GetValue()
          << std::endl;

//Enable all available chunk data
for(bo_uint64 i = 0;
    i < pDevice->GetRemoteNode("ChunkSelector")->GetEnumNodeList()->GetNodeCount();
    i++)
{
    if (pDevice->GetRemoteNode("ChunkSelector")->GetEnumNodeList()->GetNodeByIndex(i)
               ->IsReadable() == true)
    {
        pDevice->GetRemoteNode("ChunkSelector")->SetString(
                 pDevice->GetRemoteNode("ChunkSelector")->GetEnumNodeList()
                 ->GetNodeByIndex(i)->GetString());
        std::cout << "ChunkSelector: "
                  << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
                  << std::endl;
        if(pDevice->GetRemoteNode("ChunkEnable")->IsWriteable() == true)
        {
            pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
        }
        std::cout << " ChunkEnable: "
                  << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
                  << std::endl;
    }
}
```

C#

```
mDevice.RemoteNodeList["ChunkModeActive"].Value = true;
System.Console.Write(" ChunkModeActive: {0}\n",
                     (bool)mDevice.RemoteNodeList["ChunkModeActive"].Value);
for (ulong i = 0; i < mDevice.RemoteNodeList["ChunkSelector"].EnumNodeList.Count; i++)
{
    if (mDevice.RemoteNodeList["ChunkSelector"].EnumNodeList[i].IsReadable == true)
    {
        mDevice.RemoteNodeList["ChunkSelector"].Value =
                (string)mDevice.RemoteNodeList["ChunkSelector"].EnumNodeList[i].Value;
        System.Console.Write(" ChunkSelector: {0}\n",
                             (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
        if (mDevice.RemoteNodeList["ChunkEnable"].IsWriteable == true)
        {
            mDevice.RemoteNodeList["ChunkEnable"].Value = true;
        }
        System.Console.Write(" ChunkEnable: {0}\n",
                             (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);
    }
}
```

5.7.1.3 Activate Extended Chunk (setting individual pieces of chunk data)

This sample code sets individual pieces of chunk data (Image, FrameID, Timestamp, PixelFormat) of the extended chunk.

Detailed code

C++

```cpp
// Chunk Image
pDevice->GetRemoteNode("ChunkSelector")->SetString("Image");
std::cout << "ChunkSelector:"
          << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
          << std::endl;
if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
{
    pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
}
std::cout << "ChunkEnable:"
          << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
          << std::endl;

// Chunk FrameID
pDevice->GetRemoteNode("ChunkSelector")->SetString("FrameID");
std::cout << "ChunkSelector:"
          << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
          << std::endl;
if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
{
    pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
}
std::cout << "ChunkEnable:"
          << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
          << std::endl;

// Chunk Timestamp
pDevice->GetRemoteNode("ChunkSelector")->SetString("Timestamp");
std::cout << "ChunkSelector:"
          << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
          << std::endl;
if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
{
    pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
}
std::cout << "ChunkEnable:"
          << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
          << std::endl;
```

```
// Chunk PixelFormat
pDevice->GetRemoteNode("ChunkSelector")->SetString("PixelFormat");
std::cout << "ChunkSelector:"
          << pDevice->GetRemoteNode("ChunkSelector")->GetValue()
          << std::endl;
if(pDevice->GetRemoteNode("ChunkEnable")->GetCurrentAccessMode() == "RW")
{
    pDevice->GetRemoteNode("ChunkEnable")->SetBool(true);
}
std::cout << "ChunkEnable:"
          << pDevice->GetRemoteNode("ChunkEnable")->GetValue()
          << std::endl;
```

C#

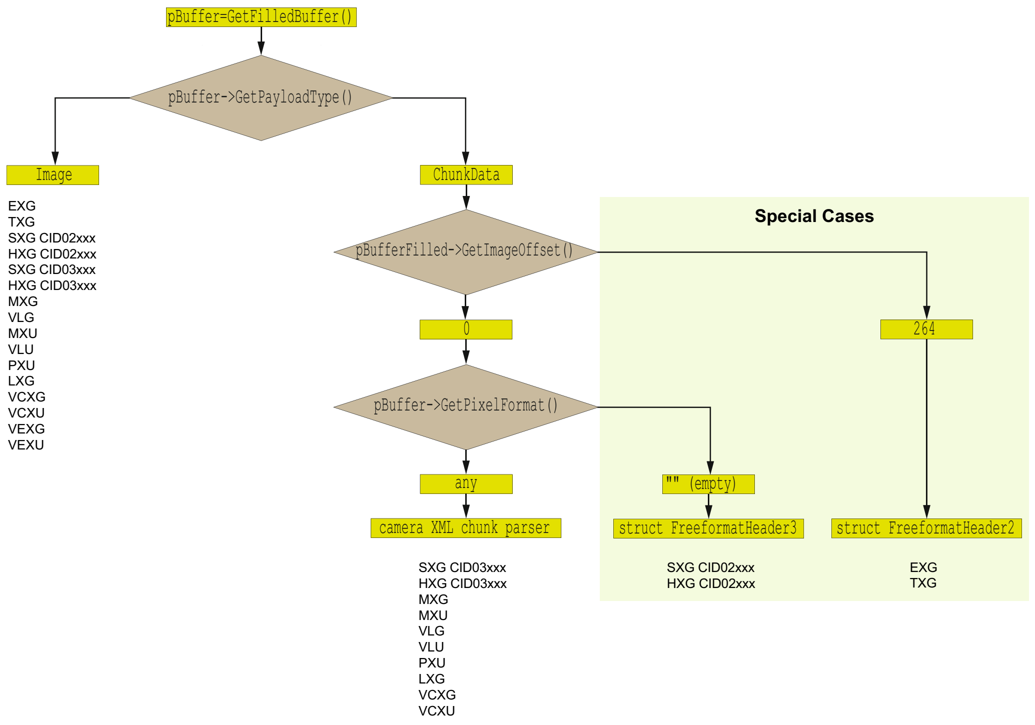
```
// Chunk Image
mDevice.RemoteNodeList["ChunkSelector"].Value = "Image";
System.Console.Write("ChunkSelector:{0}\n",
                     (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
{
    mDevice.RemoteNodeList["ChunkEnable"].Value = true;
}
System.Console.Write("ChunkEnable:{0}\n",
                     (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);

// Chunk FrameID
mDevice.RemoteNodeList["ChunkSelector"].Value = "FrameID";
System.Console.Write("ChunkSelector:{0}\n",
                     (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
{
    mDevice.RemoteNodeList["ChunkEnable"].Value = true;
}
System.Console.Write("ChunkEnable:{0}\n",
                     (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);

// Chunk Timestamp
mDevice.RemoteNodeList["ChunkSelector"].Value = "Timestamp";
System.Console.Write("ChunkSelector:{0}\n",
                     (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
{
    mDevice.RemoteNodeList["ChunkEnable"].Value = true;
}
System.Console.Write("ChunkEnable:{0}\n",
                     (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);
```

```
// Chunk PixelFormat
mDevice.RemoteNodeList["ChunkSelector"].Value = "PixelFormat";
System.Console.Write("ChunkSelector:{0}\n",
                     (string)mDevice.RemoteNodeList["ChunkSelector"].Value);
if (mDevice.RemoteNodeList["ChunkEnable"].CurrentAccessMode == "RW")
{
    mDevice.RemoteNodeList["ChunkEnable"].Value = true;
}
System.Console.Write("ChunkEnable:{0}\n",
                     (bool)mDevice.RemoteNodeList["ChunkEnable"].Value);
```

5.7.2 Get Chunk Data

The data are available in the filled buffer after exposure and transmission of the image or the image header chunk. Query of the `Payloadtype` allows you to determine if only the image was transmited (`Payloadtype = Image`) or the receiving buffer also contains chunk data (`ChunkData`).

Some cameras do not support chunk parsers. In this case `struct FreeformatHead-er2` or `FreeformatHeader3` is used to access the chunk data block.



5.7.2.1 Payloadtype "ChunkData"

This `Payloadtype` is supported by HXG, MXG, MXU, VLG, LXG, PXU, VCXG, VCXU and SXG with firmware level CID03xxx.

The chunk parser enables access to individual pieces of information from the chunk data via the feature name, e.g. ChunkFrameID (equates to the HardwareCounter).

Short Reference

C++

```
std::cout << "ChunkFrameID:  "
          << pBufferFilled->GetChunkNodeTree()->GetNode("ChunkFrameID")->GetInt()
          << std::endl;
```

C#

```
System.Console.Write("ChunkFrameID: {0}\n",
                     (long)mBufferFilled.ChunkNodeTree["ChunkFrameID"].Value);
```

The following code reads all available chunk data in a picture:

Detailed code

C++

```
if((pBufferFilled->GetPayloadType() == "ChunkData") &&
   (pBufferFilled->GetImageOffset() == 0) &&
   (pBufferFilled->GetPixelFormat() != ""))
{
    std::cout << "Buffer.FrameID:       " << pBufferFilled->GetFrameID() << std::endl;
    std::cout << "Buffer.Timestamp:     " << pBufferFilled->GetTimestamp() << std::endl;
    std::cout << "Buffer.PixelFormat:   " << pBufferFilled->GetPixelFormat() << std::endl;
    std::cout << "Buffer.Width:         " << pBufferFilled->GetWidth() << std::endl;
    std::cout << "Buffer.Height:        " << pBufferFilled->GetHeight() << std::endl;
    std::cout << "Buffer.XOffset:      " << pBufferFilled->GetXOffset() << std::endl;
    std::cout << "Buffer.YOffset:      " << pBufferFilled->GetYOffset() << std::endl;

    bo_uint img_offset = pBufferFilled->GetImageOffset();
    std::cout << "Buffer.ImageOffset: " << img_offset << std::endl;

    //CHECK THE NUMBER OF ITEMS IN THE CHUNK
    std::cout << " ChunkNodeTree.Count: "
              << pBufferFilled->GetChunkNodeTree()->GetNodeCount()
              << std::endl;
    for(bo_uint64 j = 0; j < pBufferFilled->GetChunkNodeTree()->GetNodeCount(); j++)
    {
        BGAPI2::Node * pNode = pBufferFilled->GetChunkNodeTree()->GetNodeByIndex(j);
        //CHECK EACH ITEM IS ACCESSIBLE
        if( pNode->IsReadable() == true )
        {
            //PRINT ITEM NAME AND VALUE DEPENDING ON THE INTERFACE TYPE
            if(pNode->GetInterface() == "IBoolean")
            {
                std::cout << "[" << pNode->GetInterface() << "] ";
        std::cout << pNode->GetName();
        std::cout << ": " << pNode->GetBool() << std::endl;
    }
    if(pNode->GetInterface() == "IInteger")
    {
        std::cout << "[" << pNode->GetInterface() << "] ";
        std::cout << pNode->GetName();
        std::cout << ": " << pNode->GetInt() << std::endl;
    }
    if(pNode->GetInterface() == "IEnumeration")
    {
        //IF ITEM IS SELECTOR
        //THEN SWITCH THROUGH ALL POSSIBLE SELECTIONS AND GET THEIR VALUES
        if(pNode->IsSelector() == true)
        {
            //SAVE ORIGINAL SETTING OF SELECTOR
            BGAPI2::String sSavedSelector = pNode->GetString();

            //GET A LIST OF ITEMS THAT ARE RELATED TO THIS SELECTOR
            BGAPI2::NodeMap * nSelectedNodeList = pNode->GetSelectedNodeList();

            bool IsFirstAvailableNodeInList = true;
            for( bo_uint64 l=0; l < pNode->GetEnumNodeList()->GetNodeCount(); l++)
            {
                BGAPI2::Node * nEnumNode = pNode->GetEnumNodeList()
                                                ->GetNodeByIndex(l);
                if( nEnumNode->IsReadable() == true )
                {
                    if(IsFirstAvailableNodeInList == true)
                    {
                        std::cout << "  [" << pNode->GetInterface() << "] ";
                        IsFirstAvailableNodeInList = false;
                    }
                    else
                    {
                        std::cout << "  ";
                    }
                    std::cout << pNode->GetName() << ": "
                              << pNode->GetEnumNodeList()->GetNodeByIndex(l)
                                      ->GetValue()
                              << std::endl;
                    //SET THE NEXT SELECTION
                    pNode->SetString(pNode->GetEnumNodeList()
                                          ->GetNodeByIndex(l)->GetValue());
                    for(bo_uint64 s=0; s < nSelectedNodeList->GetNodeCount(); s++)
                    {
                        BGAPI2::Node * nSelectedNode = nSelectedNodeList
                                                           ->GetNodeByIndex(s);
```

```
                                    if( nSelectedNode->IsReadable() == true )
                                    {
                                        std::cout << " " << nSelectedNode->GetName()
                                                  << ": " << nSelectedNode->GetValue()
                                                  << std::endl;
                                    }
                                }
                            }
                        }
                        //RESTORE ORIGINAL SETTING OF SELECTOR
                        pNode->SetString(sSavedSelector);
                    }
                    else //IF ITEM IS NOT A SELECTOR THEN OUTPUT VALUE
                    {
                        std::cout << " [" << pNode->GetInterface() << "] ";
                        std::cout << pNode->GetName();
                        std::cout << ": " << pNode->GetString() << std::endl;
                    }
                }
                if(pNode->GetInterface() == "IFloat")
                {
                    std::cout << " [" << pNode->GetInterface() << "] ";
                    std::cout << pNode->GetName();
                    std::cout << ": " << pNode->GetDouble() << std::endl;
                }
                if(pNode->GetInterface() == "IString")
                {
                    std::cout << " [" << pNode->GetInterface() << "] ";
                    std::cout << " " << pNode->GetName();
                    std::cout << ": " << pNode->GetString() << std::endl;
                }
            }
        }
    }
```

C#

```
if ((mBufferFilled.PayloadType == "ChunkData") &&
    (mBufferFilled.ImageOffset == 0) &&
    (mBufferFilled.PixelFormat != ""))
{
    System.Console.Write("Buffer.FrameID:      {0}\n", mBufferFilled.FrameID);
    System.Console.Write("Buffer.Timestamp:    {0}\n", mBufferFilled.Timestamp);
    System.Console.Write("Buffer.PixelFormat:  {0}\n", mBufferFilled.PixelFormat);
    System.Console.Write("Buffer.Width:        {0}\n", mBufferFilled.Width);
    System.Console.Write("Buffer.Height:       {0}\n", mBufferFilled.Height);
    System.Console.Write("Buffer.XOffset:      {0}\n", mBufferFilled.XOffset);
    System.Console.Write("Buffer.YOffset:      {0}\n", mBufferFilled.YOffset);

    ulong img_offset = (ulong)mBufferFilled.ImageOffset;
    System.Console.Write("Buffer.ImageOffset:  {0}\n", img_offset);

    //CHECK THE NUMBER OF ITEMS IN THE CHUNK
    System.Console.Write("ChunkNodeTree.Count: {0}\n",
                         mBufferFilled.ChunkNodeTree.Count);
    for (ulong j = 0; j < mBufferFilled.ChunkNodeTree.Count; j++)
    {
        BGAPI2.Node mNode = mBufferFilled.ChunkNodeTree[j];
        //CHECK EACH ITEM IS ACCESSIBLE
        if (mNode.IsReadable == true)
        {
            //PRINT ITEM NAME AND VALUE DEPENDING ON THE INTERFACE TYPE
            if ((string)mNode.Interface == "IBoolean")
            {
                System.Console.Write(" [{0}]", mNode.Interface);
                System.Console.Write(" {0}", mNode.Name);
                System.Console.Write(": {0}\n", (bool)mNode.Value);
            }
            if ((string)mNode.Interface == "IInteger")
            {
                System.Console.Write(" [{0}]", mNode.Interface);
                System.Console.Write(" {0}", mNode.Name);
                System.Console.Write(": {0}\n", (long)mNode.Value);
            }
            if ((string)mNode.Interface == "IEnumeration")
            {
                //IF ITEM IS SELECTOR
                //THEN SWITCH THROUGH ALL POSSIBLE SELECTIONS AND GET THEIR VALUES
                if (mNode.IsSelector == true)
    {
                //SAVE ORIGINAL SETTING OF SELECTOR
                string sSavedSelector = (string)mNode.Value;

                //GET A LIST OF ITEMS THAT ARE RELATED TO THIS SELECTOR
                BGAPI2.NodeMap nSelectedNodeList = mNode.SelectedNodeList;

                bool IsFirstAvailableNodeInList = true;
                for (ulong l = 0; l < mNode.EnumNodeList.Count; l++)
                {
                    BGAPI2.Node nEnumNode = mNode.EnumNodeList[l];
                    if (nEnumNode.IsReadable == true)
                    {
                        if (IsFirstAvailableNodeInList == true)
                        {
                            System.Console.Write(" [{0}]", mNode.Interface);
                            IsFirstAvailableNodeInList = false;
                        }
                        else
                        {
                            System.Console.Write("  ");
                        }
                        System.Console.Write(" {0}", mNode.Name);
                        System.Console.Write(": {0}\n", (string)nEnumNode.Value);
                        //SET THE NEXT SELECTION
                        mNode.Value = (string)nEnumNode.Value;
                        for (ulong s = 0; s < nSelectedNodeList.Count; s++)
                        {
                            BGAPI2.Node nSelectedNode = nSelectedNodeList[s];
                            if (nSelectedNode.IsReadable == true)
                            {
                                System.Console.Write("    {0}", nSelectedNode.Name);
                                if ((string)nSelectedNode.Interface == "IBoolean")
                                {
                                    System.Console.Write(": {0}\n",
                                     (bool)nSelectedNode.Value);
                                }
```

```
                                    if ((string)nSelectedNode.Interface == "IInteger")
                                    {
                                        System.Console.Write(": {0}\n",
                                                             (long)nSelectedNode.Value);
                                    }
                                    if ((string)nSelectedNode.Interface == "IEnumeration")
                                    {
                                        System.Console.Write(": {0}\n",
                                                             (string)nSelectedNode.Value);
                                    }
                                    if ((string)nSelectedNode.Interface == "IFloat")
                                    {
                                        System.Console.Write(": {0}\n",
                                                             (double)nSelectedNode.Value);
                                    }
                                    if ((string)nSelectedNode.Interface == "IString")
                                    {
                                        System.Console.Write(": {0}\n",
                                                             (string)nSelectedNode.Value);
                                    }
                                }
                            }
                        }
                    }
                    //RESTORE ORIGINAL SETTING OF SELECTOR
                    mNode.Value = sSavedSelector;
                }
                else //IF ITEM IS NOT A SELECTOR THEN OUTPUT VALUE
                {
                    System.Console.Write(" [{0}]", mNode.Interface);
                    System.Console.Write(" {0}", mNode.Name);
                    System.Console.Write(": {0}\n", (string)mNode.Value);
                }
            }
            if ((string)mNode.Interface == "IFloat")
            {
                System.Console.Write(" [{0}]", mNode.Interface);
                System.Console.Write(" {0}", mNode.Name);
                System.Console.Write(": {0}\n", (double)mNode.Value);
            }
            if ((string)mNode.Interface == "IString")
            {
                System.Console.Write("[ {0}]", mNode.Interface);
                System.Console.Write(" {0}", mNode.Name);
                System.Console.Write(": {0}\n", (string)mNode.Value);
            }
        }
    }
}
```

5.7.2.2 Special case struct FreeformatHeader3

SXG cameras with firmware level CID02xxx do not support the chunk parser, so `struct FreeformatHeader3` has to be used to access the chunk data.

SXG cameras with firmware level CID03xxxx do support the chunk parser, see „5.7.2.1 Payloadtype "ChunkData"“ on page 118

C++

```
//Start pattern of BGAPI image header V3
#define BGAPI2_IMAGEHEADER_STARTPATTERN_V3   0x1B2B3B4B

// BGAPI Image Header V3 of SXG cameras
struct FreeformatHeader3
{
    unsigned int uiStartPattern;          // 00 Start pattern of BGAPI Image Header V3
                                          // "0x1B2B3B4B"
    unsigned int uiReserved0;             // 01 reserved
    unsigned int uiReserved1;             // 02 reserved
    unsigned int uiReserved2;             // 03 reserved
    unsigned int uiReserved3;             // 04 reserved
    unsigned short usPartialScanStartX;   // 05_0 partial scan start X
    unsigned short usPartialScanStartY;   // 05_1 partial scan start Y
    unsigned short usPartialScanWidth;    // 06_0 partial scan width
    unsigned short usPartialScanHeight;   // 06_1 partial scan height
    unsigned int uiPixelFormat;           // 07 GEV pixelformat
    unsigned int uiReserved4;             // 08 reserved
    unsigned char ucBinningX;             // 09_0 binning x factor
    unsigned char ucBinningY;             // 09_1 binning y factor
    unsigned char ucSubsamplingX;         // 09_2 subsampling x factor
    unsigned char ucSubsamplingY;         // 09_3 subsampling y factor
    unsigned int uiMaster0;               // 10 features: brightness corrcetion, defectpixel
                                          //    correction, gainautobalance, enable LUT,
                                          //    h-mirror, v-mirror, enable sequencer
    unsigned int uiDigitizationTaps;      // 11 sensor digitization taps
    unsigned int uiSensorClock;           // 12 device clock - sensor
    unsigned int uiExposure1;             // 13 exposure
    unsigned int uiMeasuredExposure1;     // 14 measured exposure
    unsigned int uiReserved5;             // 15 reserved
    unsigned int uiReserved6;             // 16 reserved
    unsigned int uiReserved7;             // 17 reserved
    unsigned int uiReserved8;             // 18 reserved
    unsigned int uiReserved9;             // 19 reserved
    unsigned int uiReserved10;            // 20 reserved
//SXG: Gain from 0 dB up to 20 dB, step size 0.0359 dB
//RegisterValueDEZ=20*lg(gain factor)/0.0359+128;
    unsigned short usAnalogGain;          // 21_0 analog gain
    unsigned short usAnalogOffset;       // 21_1 analog offset
    unsigned int uiReserved11;            // 22 reserved
    unsigned int uiReserved12;            // 23 reserved
    unsigned int uiReserved13;            // 24 reserved
//SXG: gain factor * 4096
    unsigned short usPixelGainRed;        // 25_0 pixel gain red
```

```
//SXG: gain factor * 4096
    unsigned short usPixelGainGreenRed;   // 25_1 pixel gain green red

//SXG: gain factor * 4096
    unsigned short usPixelGainBlue;       // 26_0 pixel gain blue

//SXG: gain factor * 4096
    unsigned short usPixelGainGreenBlue;  // 26_1 pixel gain green blue

    unsigned int uiSequencerConfig0;      // 27 sequencer config 0
    unsigned int uiSequencerConfig1;      // 28 sequencer config 1
    unsigned int uiFrameCounter;          // 29 frame counter
    unsigned int uiTriggerCounter;        // 30 trigger counter
    unsigned int uiLostImageCounter;      // 31 lost image counter
    unsigned int uiNotDefined0;           // 32 not used
    unsigned int uiTriggerInfo;           // 33 trigger info
    unsigned int uiNotDefined1;           // 34 not used
    unsigned int uiReserved14;            // 35 reserved
    unsigned int uiNotDefined2;           // 36 not used
    unsigned int uiRequestID;             // 37 request ID
    unsigned int uiIPAddress;             // 38 IP address
    unsigned int uiTemperature;           // 39 temperature
    unsigned int uiNotDefined[23];        // 40-62 not used
    unsigned int uiStopPattern;           // 63 Stop pattern of BGAPI Image Header V3
};


//WORKAROUND SXG CID02xxx because chunk not supported in XML
if( (pBufferFilled->GetPayloadType() == "ChunkData") &&
    (pBufferFilled->GetImageOffset() == 0 ) &&
    (pBufferFilled->GetPixelFormat() == "") )
{

    std::cout << "Buffer.FrameID:     " << pBufferFilled->GetFrameID() << std::endl;
    std::cout << "Buffer.Timestamp:   " << pBufferFilled->GetTimestamp() << std::endl;
    std::cout << "Buffer.PixelFormat: " << pBufferFilled->GetPixelFormat()
              << "  - not available in chunk mode " << std::endl;
    std::cout << "Buffer.Width:       " << pBufferFilled->GetWidth()
              << " - not available in chunk mode " << std::endl;
    std::cout << "Buffer.Height:      " << pBufferFilled->GetHeight()
              << " - not available in chunk mode " << std::endl;
    std::cout << "Buffer.XOffset:     " << pBufferFilled->GetXOffset()
              << " - not available in chunk mode " << std::endl;
    std::cout << "Buffer.YOffset:     " << pBufferFilled->GetYOffset()
              << " - not available in chunk mode " << std::endl;

    bo_uint64 img_offset = pBufferFilled->GetImageOffset();
    std::cout << "Buffer.ImageOffset: " << img_offset << std::endl;
```

Notice

Please note that the following buffer information is not available if the chunk is activated:

```
Buffer::GetPixelFormat()
Buffer::GetWidth()
Buffer::GetHeight()
Buffer::GetXOffset()
Buffer::GetYOffset()
```

```
FreeformatHeader3 * pFreeformatHeader3 =
                                (FreeformatHeader3*)((char*)pBufferFilled->GetMemPtr() +
                                pBufferFilled->GetSizeFilled() - 264);

if( pFreeformatHeader3->uiStartPattern == BGAPI2_IMAGEHEADER_STARTPATTERN_V3 )
{
    std::cout << " Chunk Type FreeformatHeader3 of SXG CID02xxx " << std::endl;
    std::cout << " Chunk.ExposureTime:    "
              << pFreeformatHeader3->uiExposure1
              << std::endl;
    std::cout << " Chunk.Gain (register): "
              << pFreeformatHeader3->usAnalogGain
              << std::endl;
    std::cout << " Chunk.OffsetX:         "
              << pFreeformatHeader3->usPartialScanStartX
              << std::endl;
    std::cout << " Chunk.OffsetY:         "
              << pFreeformatHeader3->usPartialScanStartY
              << std::endl;
    std::cout << " Chunk.Width:           "
              << pFreeformatHeader3->usPartialScanWidth
              << std::endl;
    std::cout << " Chunk.Height:          "
              << pFreeformatHeader3->usPartialScanHeight
              << std::endl;
    std::cout << " Chunk.FrameCounter:    "
              << pFreeformatHeader3->uiFrameCounter
              << std::endl;
    std::cout << " Chunk.PixelFormatHex:  0x"
              << std::hex
              << pFreeformatHeader3->uiPixelFormat
              << std::dec
              << std::endl;

    BGAPI2::String sPixelFormat = "";
    switch(pFreeformatHeader3->uiPixelFormat)
    {
        case 0x01080001: sPixelFormat = "Mono8"; break;
        case 0x01100003: sPixelFormat = "Mono10"; break;
        case 0x01100005: sPixelFormat = "Mono12"; break;
        case 0x010C0006: sPixelFormat = "Mono12Packed"; break;
        case 0x0108000A: sPixelFormat = "BayerGB8"; break;
        case 0x0110000E: sPixelFormat = "BayerGB10"; break;
        case 0x01100012: sPixelFormat = "BayerGB12"; break;
        case 0x01080009: sPixelFormat = "BayerRG8"; break;
```

```
            case 0x0110000D: sPixelFormat = "BayerRG10"; break;
            case 0x01100011: sPixelFormat = "BayerRG12"; break;
            default:         sPixelFormat = "unknown";
        }
        std::cout << " Chunk.PixelFormat:      " << sPixelFormat << std::endl;
    }
}
```

Notice

Please note that the following buffer information is not available if the chunk is activated:

```
Buffer::GetPixelFormat()
Buffer::GetWidth()
Buffer::GetHeight()
Buffer::GetXOffset()
Buffer::GetYOffset()
```

C#

```
using System.Runtime.InteropService; // Marshal.Copy()

if ((mBufferFilled.PayloadType == "ChunkData") &&
    (mBufferFilled.ImageOffset == 0) &&
    (mBufferFilled.PixelFormat == ""))
{
    System.Console.Write("Buffer.FrameID:      {0}\n", mBufferFilled.FrameID);
    System.Console.Write("Buffer.Timestamp:    {0}\n", mBufferFilled.Timestamp);
    //mBufferFilled.PixelFormat not valid
    System.Console.Write("Buffer.PixelFormat:  {0}  - not available in chunk mode \n",
                         mBufferFilled.PixelFormat);
    //mBufferFilled.Width not valid
    System.Console.Write("Buffer.Width:        {0} - not available in chunk mode \n",
                         mBufferFilled.Width);
    //mBufferFilled.Height not valid
    System.Console.Write("Buffer.Height:       {0} - not available in chunk mode \n",
                         mBufferFilled.Height);
    //mBufferFilled.XOffset not valid
    System.Console.Write("Buffer.XOffset:     {0} - not available in chunk mode \n",
                         mBufferFilled.XOffset);
    //mBufferFilled.YOffset not valid
    System.Console.Write("Buffer.YOffset:     {0} - not available in chunk mode \n",
                         mBufferFilled.YOffset);

    ulong img_offset = (ulong)mBufferFilled.ImageOffset;
    System.Console.Write("Buffer.ImageOffset:  {0}\n", img_offset);

    //COPY UNMANAGED IMAGEHEADERBUFFER TO A MANAGED BYTE ARRAY
    //256 / 4 size of header data in bytes converted to 32-Bit int
    int[] imageHeaderBufferCopy = new int[256 / 4];
    Marshal.Copy((IntPtr)((ulong)mBufferFilled.MemPtr + mBufferFilled.SizeFilled - 264),
                 imageHeaderBufferCopy,
                 0,
                 (int)(256 / 4)); // 264 = 256 bytes chunk + 8 bytes chunk id

    //  00 Start pattern of BGAPI Image Header V3
    if (imageHeaderBufferCopy[0] == 0x1B2B3B4B)
    {
        System.Console.Write("Chunk Type FreeformatHeader3 of SXG CID02xxx \n");

        // 13 exposure
        System.Console.Write(" Chunk.ExposureTime:     {0}\n",
                             imageHeaderBufferCopy[13]);
```

```
// 21_0 analog gain
//SXG: Gain from 0 dB up to 20 dB, step size 0.0359 dB;
//RegisterValueDEZ=20*lg(gain factor)/0.0359+128
System.Console.Write(" Chunk.Gain:            {0}\n",
                     (ushort)((imageHeaderBufferCopy[21] & 0x0000ffff)));

// 21_1 analog offset
System.Console.Write(" Chunk.BlackLevelOffset: {0}\n",
                     (ushort)((imageHeaderBufferCopy[21] & 0xffff0000) >> 16));

// 05_0 start position x of current frame
System.Console.Write(" Chunk.OffsetX:         {0}\n",
                     (ushort)((imageHeaderBufferCopy[05] & 0x0000ffff)));

// 05_1 start position y of current frame
System.Console.Write(" Chunk.OffsetY:         {0}\n",
                     (ushort)((imageHeaderBufferCopy[05] & 0xffff0000) >> 16));

// 06_0 extension in x direction of current frame
System.Console.Write(" Chunk.Width:           {0}\n",
                     (ushort)((imageHeaderBufferCopy[06] & 0x0000ffff)));

// 06_1 extension in y direction of current frame
System.Console.Write(" Chunk.Height:          {0}\n",
                     (ushort)((imageHeaderBufferCopy[06] & 0xffff0000) >> 16));

// 29 framecounter
System.Console.Write(" Chunk.FrameCounter:    {0}\n",
                     imageHeaderBufferCopy[29]);
```

```
            // 07 GEV pixelformat
            System.Console.Write(" Chunk.PixelFormatHex:   0x{0:x}\n",
                                 imageHeaderBufferCopy[07]);
            string sPixelFormat = "";
            switch (imageHeaderBufferCopy[07])
            {
                case 0x01080001: sPixelFormat = "Mono8"; break;
                case 0x01100003: sPixelFormat = "Mono10"; break;
                case 0x010C0004: sPixelFormat = "Mono10Packed"; break;
                case 0x01100005: sPixelFormat = "Mono12"; break;
                case 0x010C0006: sPixelFormat = "Mono12Packed"; break;
                case 0x0108000A: sPixelFormat = "BayerGB8"; break;
                case 0x0110000E: sPixelFormat = "BayerGB10"; break;
                case 0x01100012: sPixelFormat = "BayerGB12"; break;
                case 0x01080009: sPixelFormat = "BayerRG8"; break;
                case 0x0110000D: sPixelFormat = "BayerRG10"; break;
                case 0x01100011: sPixelFormat = "BayerRG12"; break;
                //default: sPixelFormat = "unknown";
            }
            // 07 GEV pixelformat
            System.Console.Write(" Chunk.PixelFormat: {0}\n",
                                 sPixelFormat);
        }
        else
        {
            System.Console.Write(" FreeformatHeader3 is not available \n");
        }
    }
```

5.7.2.3 Special case struct FreeformatHeader2

TXG cameras do not support the chunk parser, so `struct FreeformatHeader2` has to be used to access the chunk data.

Chunk data can be found at the start of the image buffer. The image content begins after 264 bytes.

C++

```
//Start pattern of BGAPI Image Header V2 of TXG cameras
#define BGAPI2_IMAGEHEADER_STARTPATTERN_V2   0x1A2A3A4A

// BGAPI Image Header V2 of TXG cameras (some features are supported by customized cameras)
struct FreeformatHeader2
{
    unsigned int uiStartPattern;         // 00 Start pattern of BGAPI Image Header V2
    unsigned int uiIdentifier;           // 01 hardware type id of camera
    unsigned int uiMaster0;              // 02 first master register
    unsigned int uiMaster1;              // 03 second master register
    unsigned int uiFormat0;              // 04 first format register
    unsigned int uiFormat1;              // 05 second master register
    unsigned int uiFormatConfig;         // 06 format configuration, e.g. binning, partial,
    unsigned int uiBitMode;              // 07 pixel resolution
    unsigned int uiExposureValue;        // 08 exposure value
    unsigned int uiExposureControl;      // 09 exposure control
    unsigned int uiGain;                 // 10 analog gain
    unsigned int uiOffset;              // 11 analog offset
    unsigned int uiDigitalIO;            // 12 digital io state
    unsigned int uiFlashDelay;           // 13 flash delay
    unsigned int uiTrigger;              // 14 trigger mode settings
    unsigned int uiTriggerDelay;         // 15 trigger delay in us
    unsigned int uiTestpattern;          // 16 test pattern settings
    unsigned int uiPixelGainRed;         // 17 pixel gain red
    unsigned int uiPixelGainBlue;        // 18 pixel gain blue
    unsigned int uiPixelGainGreenRed;    // 19 pixel gain green red
    unsigned int uiPixelGainGreenBlue;   // 20 pixel gain green blue
    unsigned short usFrameStartPositionX; // 21_0 start position x of current frame
    unsigned short usFrameStartPositionY; // 21_1 start position y of current frame
    unsigned short usFrameExtensionX;    // 22_0 extension in x direction of current frame
    unsigned short usFrameExtensionY;    // 22_1 extension in y direction of current frame
    unsigned int uiLineCounter;          // 23 linecounter
    unsigned int uiSubFrameLength;       // 24 sub frame length
    unsigned int uiFrameCounter;         // 25 framecounter
    unsigned int uiUserDefined;          // 26 user defined special register
    unsigned int uiTemperature;          // 27 Temperature in C
    unsigned int uiSequencer;            // 28 sequencer configuration
    unsigned int uiDataFormat;           // 29 dataformat Rawbayer, RGB, YUV
    unsigned int uiTimeStamp;            // 30 timestamp for image capture
    unsigned int uiNotDefined[32];       // 31-62 Not defined
    unsigned int uiStopPattern;          // 63 Stop pattern of BGAPI Image Header V2
};
```

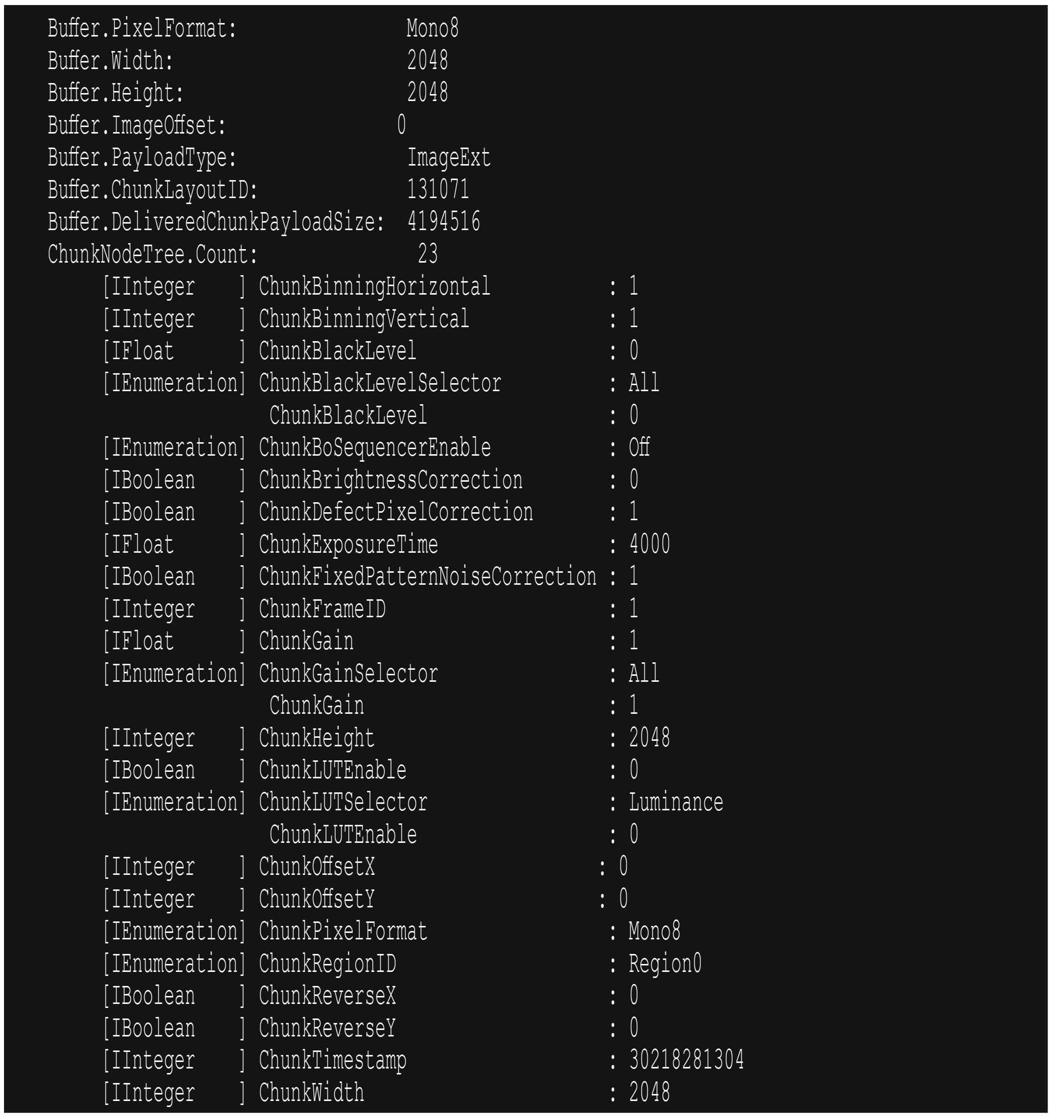
```
//IF CHUNK IS "FreeformatHeader2"
if( (pBufferFilled->GetPayloadType() == "ChunkData") &&
    (pBufferFilled->GetImageOffset() != 0) )
{
    std::cout << "Buffer.FrameID:     " << pBufferFilled->GetFrameID() << std::endl;
    std::cout << "Buffer.Timestamp:   " << pBufferFilled->GetTimestamp() << std::endl;
    std::cout << "Buffer.PixelFormat: " << pBufferFilled->GetPixelFormat() << std::endl;
    std::cout << "Buffer.Width:       " << pBufferFilled->GetWidth() << std::endl;
    std::cout << "Buffer.Height:      " << pBufferFilled->GetHeight() << std::endl;
    std::cout << "Buffer.XOffset:     " << pBufferFilled->GetXOffset() << std::endl;
    std::cout << "Buffer.YOffset:     " << pBufferFilled->GetYOffset() << std::endl;

    bo_uint64 img_offset = pBufferFilled->GetImageOffset();
    std::cout << "Buffer.ImageOffset: " << img_offset << std::endl;

    FreeformatHeader2 *pFreeformatHeader2 = (FreeformatHeader2*)pBufferFilled->GetMemPtr();
    if( pFreeformatHeader2->uiStartPattern == BGAPI2_IMAGEHEADER_STARTPATTERN_V2 )
    {
        std::cout << " Chunk.ExposureTime:     "
                  << pFreeformatHeader2->uiExposureValue
                  << std::endl;
        std::cout << " Chunk.Gain:             "
                  << pFreeformatHeader2->uiGain
                  << std::endl;
        std::cout << " Chunk.BlackLevelOffset: "
                  << pFreeformatHeader2->uiOffset
                  << std::endl;
        std::cout << " Chunk.OffsetX:          "
                  << pFreeformatHeader2->usFrameStartPositionX
                  << std::endl;
        std::cout << " Chunk.OffsetY:          "
                  << pFreeformatHeader2->usFrameStartPositionY
                  << std::endl;
        std::cout << " Chunk.Width:            "
                  << pFreeformatHeader2->usFrameExtensionX
                  << std::endl;
        std::cout << " Chunk.Height:           "
                  << pFreeformatHeader2->usFrameExtensionY
                  << std::endl;
        std::cout << " Chunk.FrameCounter:     "
                  << pFreeformatHeader2->uiFrameCounter
                  << std::endl;
        std::cout << " Chunk.PixelFormatHex:   0x"
                  << std::hex
                  << pFreeformatHeader2->uiDataFormat
                  << std::dec
                  << std::endl;
```

```
            BGAPI2::String sPixelFormat = "";
            switch(pFreeformatHeader2->uiDataFormat)
            {
                //TXG PixelFormats
                case 0x01080001: sPixelFormat = "Mono8"; break;
                case 0x01100003: sPixelFormat = "Mono10"; break;
                case 0x010C0004: sPixelFormat = "Mono10Packed"; break;
                case 0x01100005: sPixelFormat = "Mono12"; break;
                case 0x010C0006: sPixelFormat = "Mono12Packed"; break;
                case 0x0108000A: sPixelFormat = "BayerGB8"; break;
                case 0x0110000E: sPixelFormat = "BayerGB10"; break;
                case 0x01100012: sPixelFormat = "BayerGB12"; break;
                case 0x01080009: sPixelFormat = "BayerRG8"; break;
                case 0x0110000D: sPixelFormat = "BayerRG10"; break;
                case 0x01100011: sPixelFormat = "BayerRG12"; break;
                case 0x02180015: sPixelFormat = "BGR8Packed"; break;
                case 0x02180014: sPixelFormat = "RGB8Packed"; break;
                case 0x020C001E: sPixelFormat = "YUV411Packed"; break;
                case 0x0210001F: sPixelFormat = "YUV422Packed"; break;
                case 0x02180020: sPixelFormat = "YUV444Packed"; break;
                default:         sPixelFormat = "unknown";
            }
            std::cout << " Chunk.PixelFormat:      " << sPixelFormat << std::endl;
        }
        else
        {
            std::cout << "FreeformatHeader2 is not available." << std::endl;
        }
    }
```

C#

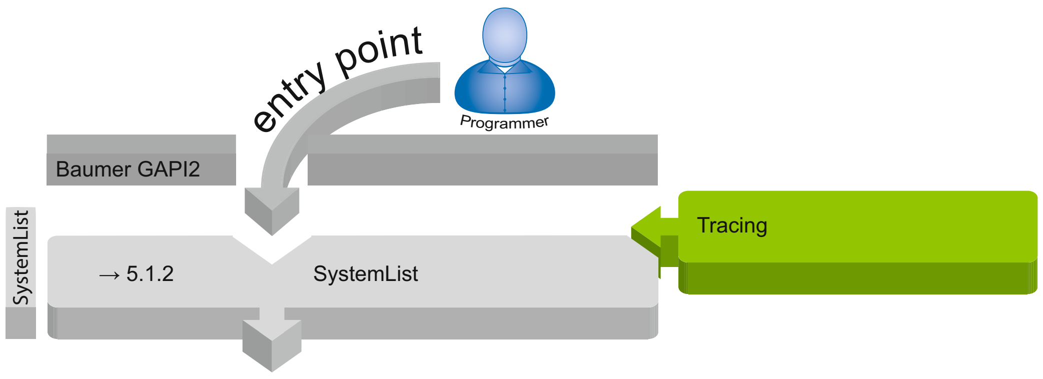
```
// TXG, EXG ImageOffset = 264 FreeformatHeader2
using System.Runtime.InteropServices; // Marshal.Copy()
if ((mBufferFilled.PayloadType == "ChunkData") &&
    (mBufferFilled.ImageOffset != 0))
{
    System.Console.Write("Buffer.FrameID:     {0}\n", mBufferFilled.FrameID);
    System.Console.Write("Buffer.Timestamp:   {0}\n", mBufferFilled.Timestamp);
    System.Console.Write("Buffer.PixelFormat: {0}\n", mBufferFilled.PixelFormat);
    System.Console.Write("Buffer.Width:       {0}\n", mBufferFilled.Width);
    System.Console.Write("Buffer.Height:      {0}\n", mBufferFilled.Height);
    System.Console.Write("Buffer.XOffset:     {0}\n", mBufferFilled.XOffset);
    System.Console.Write("Buffer.YOffset:     {0}\n", mBufferFilled.YOffset);

    ulong img_offset = (ulong)mBufferFilled.ImageOffset;
    System.Console.Write("Buffer.ImageOffset: {0}\n", img_offset);

    //COPY UNMANAGED IMAGEHEADERBUFFER TO A MANAGED BYTE ARRAY
    // 8-Bit byte img_offset converted to 32-Bit int offset
    int[] imageHeaderBufferCopy = new int[img_offset / 4];
    Marshal.Copy(mBufferFilled.MemPtr,
                 imageHeaderBufferCopy,
                 0,
                 (int)(img_offset / 4));

    //  00 Start pattern of BGAPI Image Header V2
    if (imageHeaderBufferCopy[0] == 0x1A2A3A4A)
    {
        // 08 exposure value
        System.Console.Write(" Chunk.ExposureTime:    {0}\n",
                             imageHeaderBufferCopy[8]);

        // 10 analog gain
        System.Console.Write(" Chunk.Gain:            {0}\n",
                             imageHeaderBufferCopy[10]);

        // 11 analog offset
        System.Console.Write(" Chunk.BlackLevelOffset: {0}\n",
                             imageHeaderBufferCopy[11]);

        // 21_0 start position x of current frame
        System.Console.Write(" Chunk.OffsetX:         {0}\n",
                             (ushort)((imageHeaderBufferCopy[21] & 0x0000ffff)));

        // 21_1 start position y of current frame
        System.Console.Write(" Chunk.OffsetY:         {0}\n",
                             (ushort)((imageHeaderBufferCopy[21] & 0xffff0000) >> 16));
        // 22_0 extension in x direction of current frame
        System.Console.Write(" Chunk.Width:           {0}\n",
                            (ushort)((imageHeaderBufferCopy[22] & 0x0000ffff)));

        // 22_1 extension in y direction of current frame
        System.Console.Write(" Chunk.Height:          {0}\n",
                             (ushort)((imageHeaderBufferCopy[22] & 0xffff0000) >> 16));

        // 25 framecounter
        System.Console.Write(" Chunk.FrameCounter:    {0}\n",
                            imageHeaderBufferCopy[25]);

        // 29 dataformat Rawbayer, RGB, YUV
        System.Console.Write(" Chunk.PixelFormatHex:   0x{0:x}\n",
                            imageHeaderBufferCopy[29]);
        string sPixelFormat = "";

        // 29 dataformat Rawbayer, RGB, YUV
        switch (imageHeaderBufferCopy[29])
        {
            case 0x01080001: sPixelFormat = "Mono8"; break;
            case 0x01100003: sPixelFormat = "Mono10"; break;
            case 0x010C0004: sPixelFormat = "Mono10Packed"; break;
            case 0x01100005: sPixelFormat = "Mono12"; break;
            case 0x010C0006: sPixelFormat = "Mono12Packed"; break;
            case 0x0108000A: sPixelFormat = "BayerGB8"; break;
            case 0x0110000E: sPixelFormat = "BayerGB10"; break;
            case 0x01100012: sPixelFormat = "BayerGB12"; break;
            case 0x01080009: sPixelFormat = "BayerRG8"; break;
            case 0x0110000D: sPixelFormat = "BayerRG10"; break;
            case 0x01100011: sPixelFormat = "BayerRG12"; break;
            case 0x02180015: sPixelFormat = "BGR8Packed"; break;
            case 0x02180014: sPixelFormat = "RGB8Packed"; break;
            case 0x020C001E: sPixelFormat = "YUV411Packed"; break;
            case 0x0210001F: sPixelFormat = "YUV422Packed"; break;
            case 0x02180020: sPixelFormat = "YUV444Packed"; break;
            //default: sPixelFormat = "unknown";
        }
        // 29 dataformat Rawbayer, RGB, YUV
        System.Console.Write(" Chunk.PixelFormat:  {0}\n", sPixelFormat);
    }
    else
    {
        System.Console.Write("   FreeformatHeader2 is not available \n");
    }
}
```

Output C++ / C#



```
Buffer.PixelFormat:                Mono8
Buffer.Width:                      2048
Buffer.Height:                     2048
Buffer.ImageOffset:               0
Buffer.PayloadType:                ImageExt
Buffer.ChunkLayoutID:              131071
Buffer.DeliveredChunkPayloadSize:  4194516
ChunkNodeTree.Count:                23
     [IInteger    ] ChunkBinningHorizontal           : 1
     [IInteger    ] ChunkBinningVertical             : 1
     [IFloat      ] ChunkBlackLevel                  : 0
     [IEnumeration] ChunkBlackLevelSelector          : All
                     ChunkBlackLevel                 : 0
     [IEnumeration] ChunkBoSequencerEnable           : Off
     [IBoolean    ] ChunkBrightnessCorrection        : 0
     [IBoolean    ] ChunkDefectPixelCorrection       : 1
     [IFloat      ] ChunkExposureTime                : 4000
     [IBoolean    ] ChunkFixedPatternNoiseCorrection : 1
     [IInteger    ] ChunkFrameID                     : 1
     [IFloat      ] ChunkGain                        : 1
     [IEnumeration] ChunkGainSelector                : All
                     ChunkGain                       : 1
     [IInteger    ] ChunkHeight                      : 2048
     [IBoolean    ] ChunkLUTEnable                   : 0
     [IEnumeration] ChunkLUTSelector                 : Luminance
                     ChunkLUTEnable                  : 0
     [IInteger    ] ChunkOffsetX                    : 0
     [IInteger    ] ChunkOffsetY                    : 0
     [IEnumeration] ChunkPixelFormat                 : Mono8
     [IEnumeration] ChunkRegionID                    : Region0
     [IBoolean    ] ChunkReverseX                    : 0
     [IBoolean    ] ChunkReverseY                    : 0
     [IInteger    ] ChunkTimestamp                   : 30218281304
     [IInteger    ] ChunkWidth                       : 2048
```

5.8 Tracing



The *Tracing* function gives you the option of specifying events in the program sequence that can be output in chronological order into a log file or debugger.

The tracing function can activated to any time. But it makes sense to call the function early, to detect errors.

Trace logfile size

Notice
For a comfortable analysis of the *.log file, you can rename the file in *.csv. Now you can open the renamed file with a table calculation programme, e.g. Microsoft® Excel®.

Trace output file is splitted and zipped if size is getting higher than 100 MBytes.

At the start of tracing, the available free space is checked on the harddisk.
Therefore, 3*100 MBytes of dummy data are written to harddisk in steps of 1 MByte, temporary.
In case of less space, like e.g. 240 MBytes of free space, the trace logfile size is reduced to a third of the available size, like 240 / 3 is 80 Mbytes.

If the limit of max 100 MBytes is reached, the file is renamed with an additional `.001` and compressed to a .zip file. Other existing .zip files are renamed to a higher number, like `.001.log.zip` is renamed to `.002.log.zip`, and `.002.log.zip` is renamed to `.003.log.zip`, and so on.

Example:

`bgapi2_trace_my.log`	uncompressed	up to 100 MBytes (newest data)
`bgapi2_trace_my.001.log.zip`	compressed	about 3 MBytes
`bgapi2_trace_my.002.log.zip`	compressed	about 3 MBytes
`bgapi2_trace_my.003.log.zip`	compressed	about 3 MBytes (oldest data)

The uncompressed trace logfile always holds the newest data. And, the higher the additional number of the zip file the older data are included. The maximum count of zip files per application is limited to 400.

Trace Standard Settings

The following tables shows the standard settings for the Trace when enabling the Trace function.

Output destination	Debugger
levels of severity	all ([Err] / [Wrn] / [Inf])
Configuration	Prefix, Timestamp

Standard settings for the automatic Trace of the Consumer + Baumer Producer:

Output destination	File
File name	`bgapi2_trace.log`
levels of severity	all ([Err] / [Wrn] / [Inf])
Configuration	Prefix, Timestamp

Standard settings for the automatic Trace of the Producer (currently only Baumer GigE Producer):

Output destination	File
File name	`baumer_gige.log`
levels of severity	all ([Err] / [Wrn] / [Inf])
Configuration	Prefix, Timestamp

Configuration

The log file shows preconfigured information on which software module generated the event.

Prefix in trace log	Module which created the log output
`[bgapi2_gige]`	Baumer GigE Producer
`[bgapi2_usb]`	Baumer USB Producer
`[bgapi2_genicam]`	Baumer Genicam Consumer

Automatic Trace Output

There is the possibility of an automatic Trace.

Windows® 7

Create a folder „Baumer" in your AppData\Local

```
C:\Users\Administrator\AppData\Local\Baumer
```

Create an empty file „bgapi2_trace_enable.ini"

```
C:\Users\Administrator\AppData\Local\Baumer\bgapi2_trace_
enable.ini
```

Linux

Create a folder „Baumer" in your home directory ~/.local/share/data

```
~/.local/share/Baumer
```

Create an empty file „bgapi2_trace_enable.ini"

```
~/.local/share/Baumer/bgapi2_trace_enable.ini
```

In Linux, the logfiles are compressed to .gz files.

Trace output file will be created automatically in that folder, if a application is started.

```
bgapi2_trace_bexplorer_x64.exe.log
bgapi2_trace_001_ImageCaptureMode_Polling.exe.log
bgapi2_trace_002_CameraParameterTree.exe.log
bgapi2_trace_003_CameraParameterSetup.exe.log
```

Notice

Do not forget to delete „bgapi2_trace_enable.ini", if no longer needed.

Detailed code

C++

```
// Output Destination
// You can choose between the output as a file or Debugger.
// Is it possible to give a file name. If a new file name is set, the messages are written
// immediately in the new file.
BGAPI2::Trace::ActivateOutputToFile(true, (BGAPI2::String)"bgapi2_trace_my.log");


// Output via the Debugger (e.g. the program DebugView)
BGAPI2::Trace::ActivateOutputToDebugger(false);


// Definition the level of severity
// The errors are divided into 3 levels of severity. Here are errors represented as
// [Err], Warnings represented as [Wrn] and information represented as [Inf].
BGAPI2::Trace::ActivateMaskError(true);
BGAPI2::Trace::ActivateMaskWarning(true);
BGAPI2::Trace::ActivateMaskInformation(true);


// The timestamp indicates the time when a trace message was generated. An additional option //
for the Timestamp is the indication of the difference to the last trace of a message. The //
difference is indicated in round brackets in ms. The following functions are available:
BGAPI2::Trace::ActivateOutputOptionPrefix(true);
BGAPI2::Trace::ActivateOutputOptionTimestamp(true);
BGAPI2::Trace::ActivateOutputOptionTimestampDiff(true);


// Finally, you must enable the function.
BGAPI2::Trace::Enable(true);
```

C#

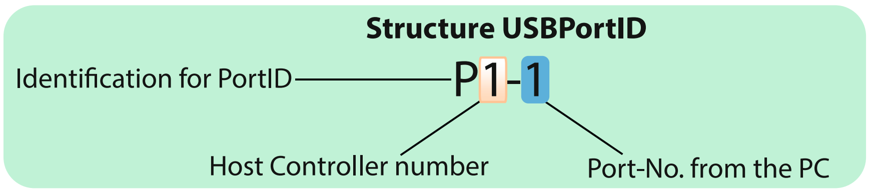
```
// Output Destination

// You can choose between the output as a file or Debugger.

// Is it possible to give a file name. If a new file name is set, the messages are written

// immediately in the new file.

BGAPI2.Trace.Instance.ActivateOutputToFile(true, "bgapi2_trace_my.log");


// Output via the Debugger (e.g. the program DebugView)

BGAPI2.Trace.Instance.ActivateOutputToDebugger(false);


// Definition the level of severity

// The errors are divided into 3 levels of severity. Here are errors represented as

// [Err], Warnings represented as [Wrn] and information represented as [Inf].

BGAPI2.Trace.Instance.ActivateMaskError(true);

BGAPI2.Trace.Instance.ActivateMaskWarning(true);

BGAPI2.Trace.Instance.ActivateMaskInformation(true);


// The timestamp indicates the time when a trace message was generated. An additional option //
for the Timestamp is the indication of the difference to the last trace of a message. The //
difference is indicated in round brackets in ms. The following functions are available:

BGAPI2.Trace.Instance.ActivateOutputOptionPrefix(true);

BGAPI2.Trace.Instance.ActivateOutputOptionTimestamp(true);

BGAPI2.Trace.Instance.ActivateOutputOptionTimestampDiff(true);


// Finally, you must enable the function.

BGAPI2.Trace.Instance.Enable(true);
```

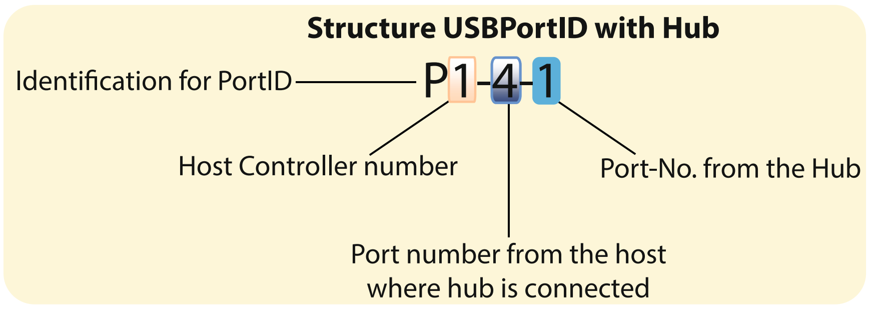
The follow shows an example part of an log file.

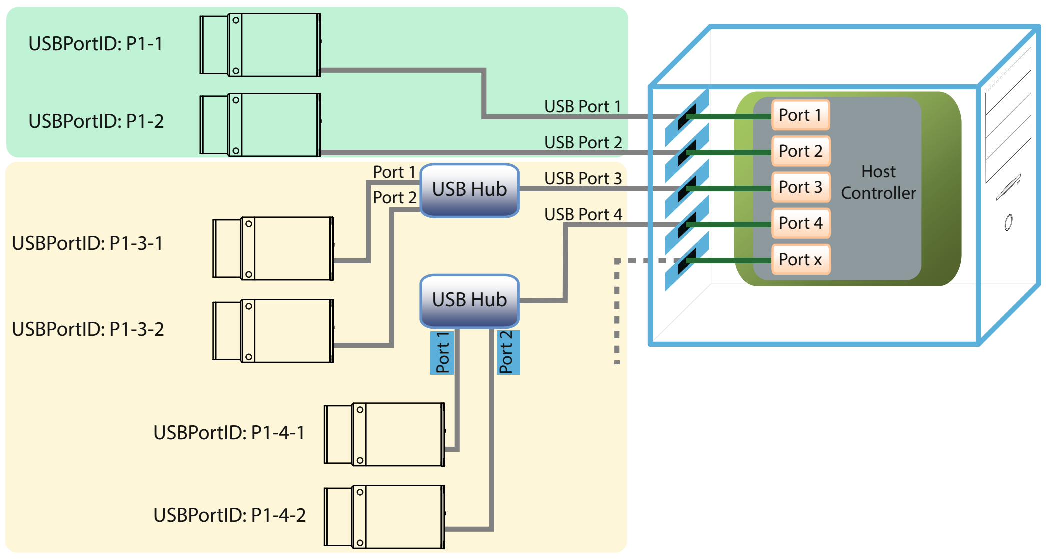
```
[...]

03.06.2015 13:24:36,113000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetFileName; TLProducer Name: bgapi2_gige.cti
03.06.2015 13:24:36,113000; 0,000000; bgapi2_gige; Inf; Baumer GigE TL; TLGetInfo; called for iInfoCmd 4
03.06.2015 13:24:36,113000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetTLType; TLProducer TLType: GEV
03.06.2015 13:24:36,114000; 0,000000; bgapi2_gige; Inf; Baumer GigE TL; TLGetInfo; called for iInfoCmd 3
03.06.2015 13:24:36,114000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetVersion; TLProducer Version: 2.3.6493.7026
03.06.2015 13:24:36,115000; 0,000000; bgapi2_gige; Inf; Baumer GigE TL; TLGetInfo; called for iInfoCmd 6
03.06.2015 13:24:36,115000; 0,000000; bgapi2_usb; Inf; Baumer USB TL; TLGetInfo; called
03.06.2015 13:24:36,115000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetFileName; TLProducer Name: bgapi2_usb.cti
03.06.2015 13:24:36,115000; 0,000000; bgapi2_usb; Inf; Baumer USB TL; TLGetInfo; called
03.06.2015 13:24:36,115000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetTLType; TLProducer TLType: U3V
03.06.2015 13:24:36,115000; 0,000000; bgapi2_usb; Inf; Baumer USB TL; TLGetInfo; called
03.06.2015 13:24:36,115000; 0,000000; bgapi2_genicam; Inf; ProducerObj; GetVersion; TLProducer Version: 2.3.7015.7026
03.06.2015 13:24:36,115000; 0,000000; bgapi2_usb; Inf; Baumer USB TL; TLGetInfo; called
[...]
```

5.9 PortID (only for USB cameras)

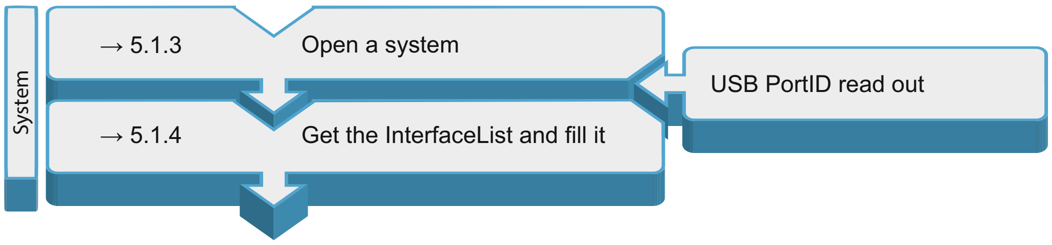
The *USBPortID* describes the invariable, hierarchical position of the USB ports on the computer.




Structure USBPortID with Hub
Identification for PortID
P1-4-1
Host Controller number
Port-No. from the Hub
Port number from the host where hub is connected



USB PortID read out



The PortID itself can be read out using the feature of the system.

C++

```
pSystem->Open();

NodeMap* pRootNodeMap = pSystem->GetNodeList();
if (pRootNodeMap &&
    pRootNodeMap->GetNodePresent("USBPortSelector") &&
    pRootNodeMap->GetNodePresent("USBPortID") &&
    pRootNodeMap->GetNodePresent("USBPortLocationPath"))
{
    // list all available PortID on USB TL
    Node* pSelector = pRootNodeMap->GetNode("USBPortSelector");
    if (pSelector->GetCurrentAccessMode() != "RW")
    {
        std::cout << "error : no Port Idents found!" << std::endl;
    }
    else
    {
        bo_int64 nMax = pSelector->GetIntMax();
        std::cout << nMax + 1 << " Port Idents with devices found!" << std::endl;
        for (bo_int64 i = 0; i <= nMax; i++)
        {
            pSelector->SetInt(i);
            BGAPI2::String strPortID = pRootNodeMap->GetNode("USBPortID")->GetValue();
            BGAPI2::String strLocPath = pRootNodeMap->GetNode("USBPortLocationPath")->GetValue();
            std::cout << i + 1 << ".  " << strPortID << "   " << strLocPath << std::endl;
        }
    }
}
```

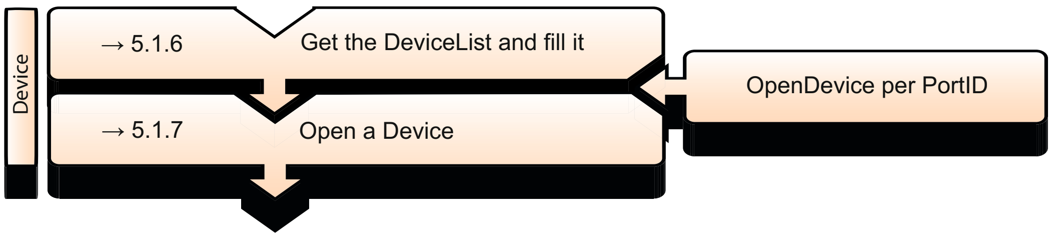
C#

```csharp
mSystem.Open();
if (mSystem.NodeList.GetNodePresent("USBPortSelector")
     && mSystem.NodeList.GetNodePresent("USBPortID")
     && mSystem.NodeList.GetNodePresent("USBPortLocationPath"))
{
    // list all available PortID on USB TL
    BGAPI2.Node portSelector = mSystem.NodeList["USBPortSelector"];
    if (portSelector.IsReadable && portSelector.IsWriteable)
    {
        long maxSelector = (long)portSelector.Max;
        System.Console.WriteLine((maxSelector+1).ToString() + " Port Idents with devices found!");
        System.Console.WriteLine();
        for (long iSelector = 0; iSelector <= maxSelector; iSelector++)
        {
            portSelector.Value = iSelector;
            System.Console.WriteLine("{0}.  {1}  {2}",
                                         (iSelector + 1).ToString(),
                                         (string)mSystem.NodeList["USBPortID"].Value,
                                         (string)mSystem.NodeList["USBPortLocationPath"].Value);
        }
    }
    else
    {
        System.Console.WriteLine("hint: no Port Idents found!");
    }
}
```

Output C++ / C#

```
3 Port Idents with devices found!

1 .  P2-1-5       PCIROOT(0)#PCI(1D00)#USBROOT(0)#USB(1)#USB(5)
2 .  P3-16-2      PCIROOT(0)#PCI(1400)#USBROOT(0)#USB(16)#USB(2)
3 .  P3-16-3      PCIROOT(0)#PCI(1400)#USBROOT(0)#USB(16)#USB(3)
```

Open Device per PortID



An OpenDevice per PortID might look like this.

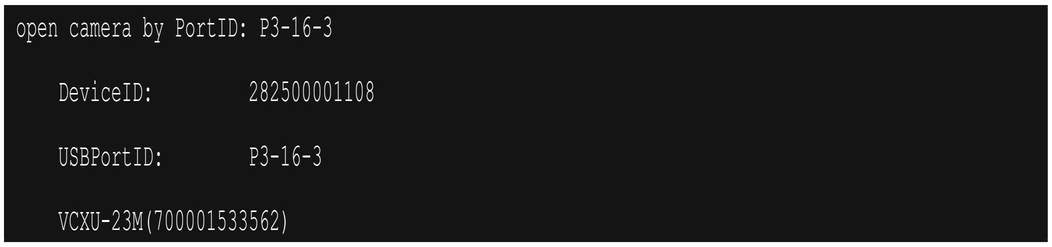
C++

```cpp
Device *getDeviceByPortID(BGAPI2::String strPortID, DeviceList *deviceList)
{
    for (DeviceList::iterator it = deviceList->begin(); it != deviceList->end(); it++)
    {
        Device *pDevice = (*it)->second;
        NodeMap* pDeviceNodeMap = pDevice->GetNodeList();
        if (pDeviceNodeMap->GetNodePresent("USBPortID"))
        {
            Node* pNode = pDeviceNodeMap->GetNode("USBPortID");
            if (pNode->GetValue() == strPortID.get())
            {
                return pDevice;
            }
        }
    }
    return NULL;
}


int main()
{

    //open system (USB3)
    //open interface (USB3)

    // open camera by (known) PortID
    BGAPI2::String strFirstPortID = "P3-16-3";
    std::cout << "open camera by PortID: " << strFirstPortID << std::endl;

    DeviceList *deviceList = pInterface->GetDevices();
    deviceList->Refresh(100);
    if (deviceList->size() > 0)
    {
        Device *pDevice = getDeviceByPortID(strFirstPortID, deviceList);
        if (pDevice)
        {
            BGAPI2::String strID = pDevice->GetID();
            std::cout << "DeviceID: " << strID << std::endl;

            NodeMap* pDeviceNodeMap = pDevice->GetNodeList();
            if (pDeviceNodeMap->GetNodePresent("USBPortID"))
            {
                Node* pNode = pDeviceNodeMap->GetNode("USBPortID");
                std::cout << "USBPortID: " << pNode->GetValue() << std::endl;
            }
            pDevice->Open();
std::cout << pDevice->GetModel() << "(" << pDevice->GetSerialNumber() << ")";
std::cout <<  std::endl;
        }
    }
    else
    {
        std::cout << "no camera found on u3v system and first interface." << std::endl;
    }
    ...
    ...
}
```

C#

```
static BGAPI2.Device getDeviceByPortId(string sPortId, BGAPI2.DeviceList devList)
{
    BGAPI2.Device dev = null;
    foreach (BGAPI2.Device tmpDev in devList.Values)
    {
        if (tmpDev.NodeList.GetNodePresent("USBPortID"))
        {
            if (sPortId == (string)tmpDev.NodeList["USBPortID"].Value)
            {
                dev = tmpDev;
                break;
            }
        }
    }
    return dev;
}

static int Main(string[] args)
{
    //open system (USB3)
    //open interface (USB3)

    // open camera by PortID
    string sFirstPortId = "P3-16-3";

    System.Console.WriteLine("open camera by PortID: " + sFirstPortId);
    mInterface.Devices.Refresh(100);
    BGAPI2.Device mDev = getDeviceByPortId(sFirstPortId, mInterface.Devices);
    if (mDev != null)
    {
        if (mDev.NodeList.GetNodePresent("USBPortID"))
        {
            System.Console.WriteLine("USBPortID: {0}",
                                     (string)mDev.NodeList["USBPortID"].Value);
        }

        mDev.Open();
        System.Console.WriteLine("mDev.Model + "(" + mDev.SerialNumber + ")");
        ...
        ...
    }
}
```

Output C++ / C#



```
open camera by PortID: P3-16-3

    DeviceID:         282500001108

    USBPortID:        P3-16-3

    VCXU-23M(700001533562)
```

6. Support / Software Examples

In case of any questions, or for troubleshooting, please contact our support team.

Worldwide

Baumer Optronic GmbH
Badstrasse 30
DE-01454 Radeberg, Germany

Tel: +49 (0)3528 4386 845

Email: support.cameras@baumer.com

Website: www.baumer.com

Baumer GAPI software examples

https://www.baumer.com/a/service-support/know-how/technical-application-notes-industrial-cameras/a/technical-application-notes-industrial-cameras#baumer-gapi-software-examples

Baumer

Baumer Optronic GmbH
Badstrasse 30
DE-01454 Radeberg, Germany
Phone +49 (0)3528 4386 0 · Fax +49 (0)3528 4386 86
sales@baumeroptronic.com · www.baumer.com