Baumer

Programmer's Guide
Baumer GAPI SDK 2.3

Document Version: v1.2
Release: 22.05.2014
Document Number: 11099008

Table of Contents

Linux

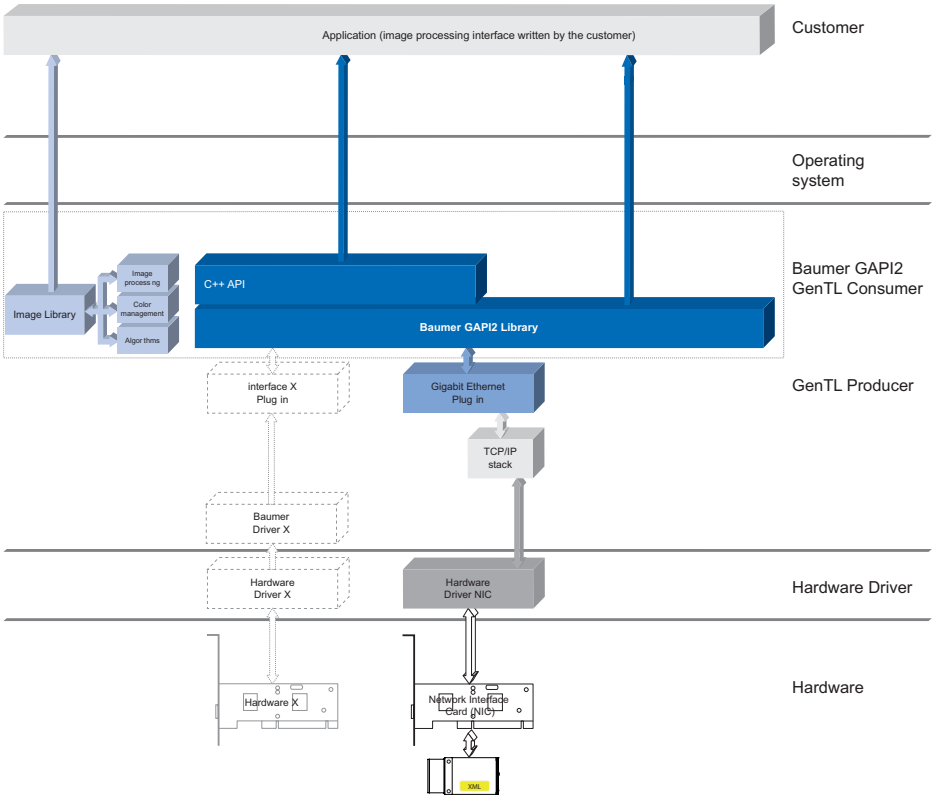


Figure 1 ►
Baumer GAPI stack (Linux)

Windows

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

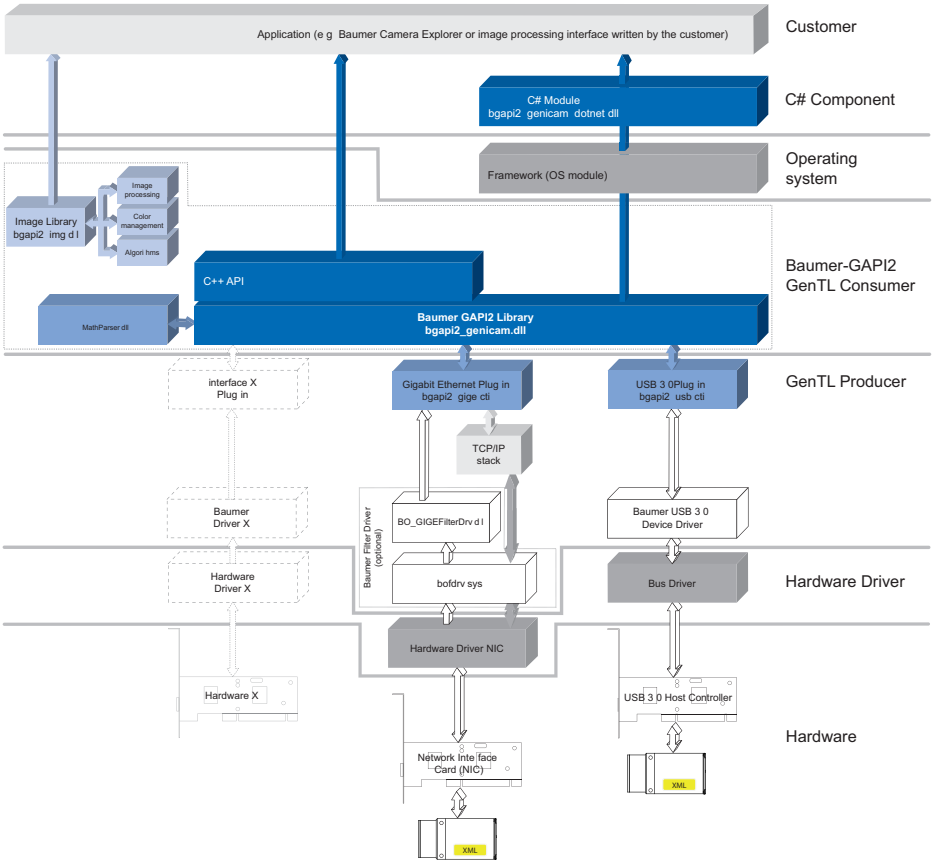


Figure 2 ►
Baumer GAPI stack (Microsoft® Windows®)

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.

2.1.1.2 USB 3.0 Vision

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.

Notice
USB is not supported on Linux® operating systems.

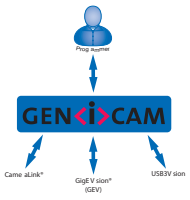
GenICam:

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).

2.1.2 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**
The GenICam Vision

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 XP 32/64 bit systems Microsoft Windows 7 32/64 bit systems Microsoft Windows 8 32/64 bit systems	
	Linux	Debian 7 Ubuntu 12.04 Ubuntu 13.04 Ubuntu 14.04 Fedora 20 OpenSUSE 13.1	
USB 3.0 Vision	**Windows**	Microsoft Windows 7 32/64 bit systems Microsoft Windows 8 32/64 bit systems	
	Linux	not supported	
Compiler	**Windows**	Microsoft® Compiler only (included in Microsoft® Visual Studio)	
	Linux	GCC Version >= 4.7	
C++ Version		C++11	
Framework (optional)		Windows OS: .Net Framework 4.0 or higher 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.

2.3.2 Recommended equipment

Help CHM-File / Header / Camera User Guides

The individual functions of the Baumer GAPI SDK are described in the CHM file provided (*GAPI_SDK_HELP.chm*).

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

You can also find information in the User's Guide for the respective camera.

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/cms/index.php?idcat=47

SFNC	1.5.1	2.0	2.1
Camera Series	MXG/VLG	MXU/VLU	LXG
	SXG/HXG		PXU

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

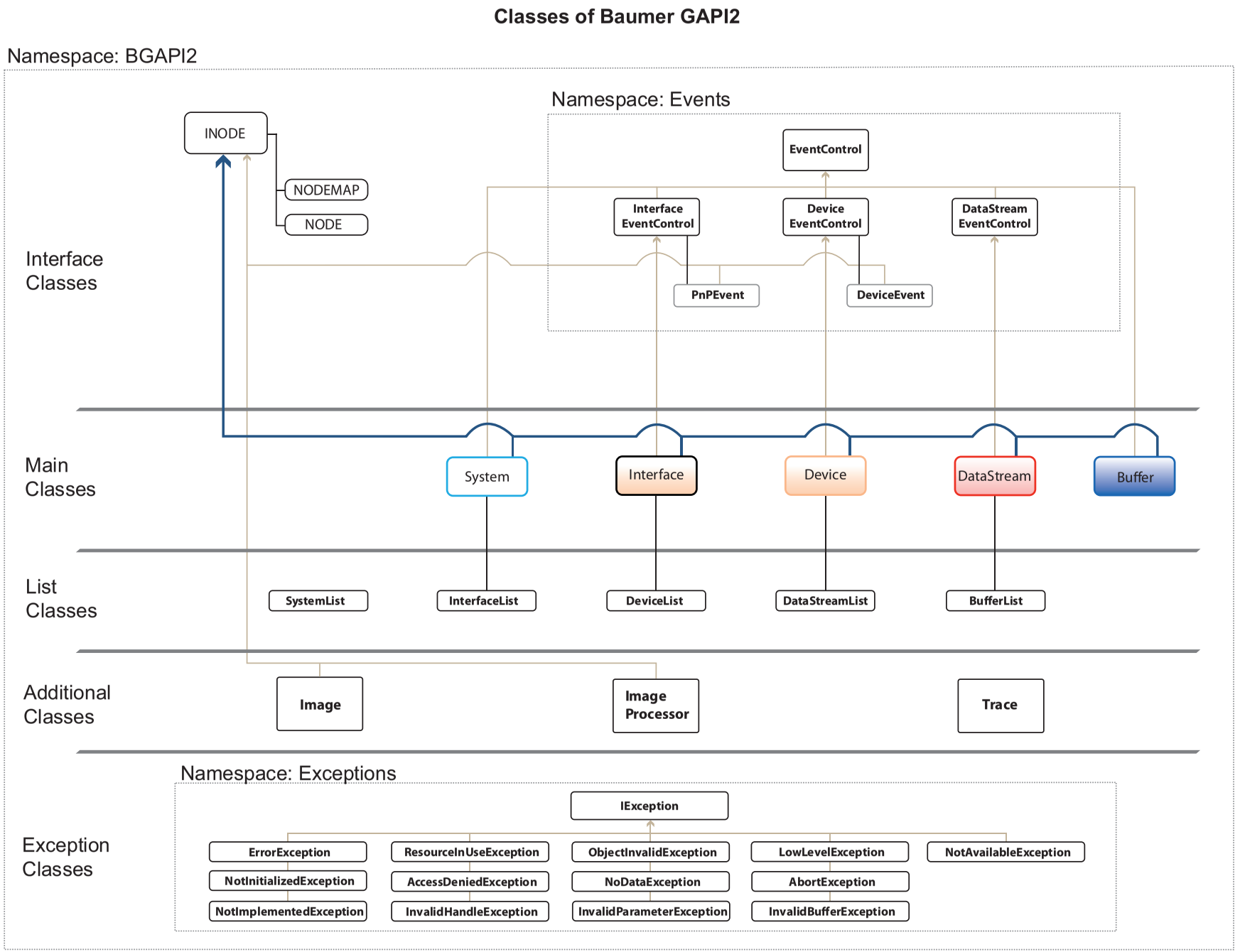
You can read out the range of features supported by your camera using *Baumer Camera Explorer*.

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)

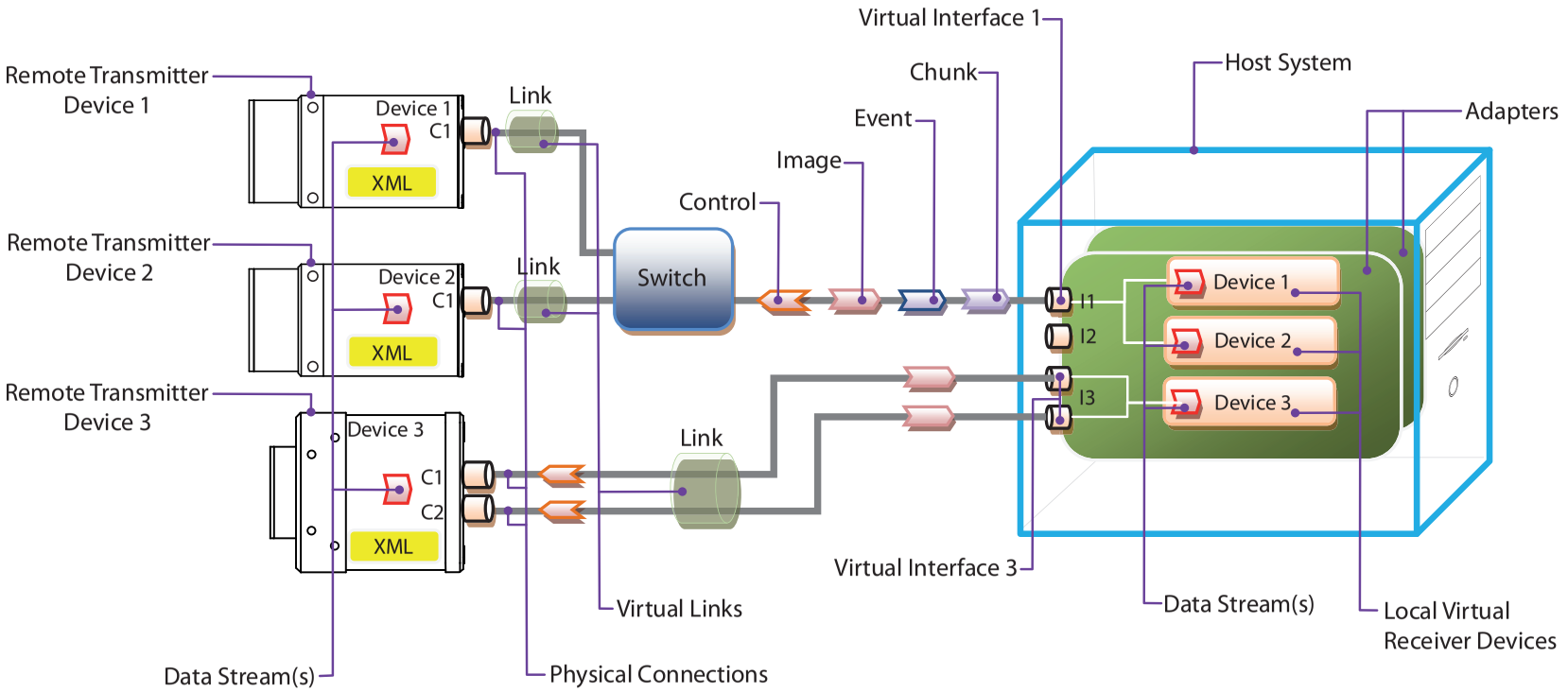
The programming examples included are only intended to be used as guidelines. The output may be different depending on the OS and interface types.



▲ **Figure 4**
Baumer GAPI 2 classes

Device Communication Model

Reference: GenICam Standard Features Naming Convention v2.1

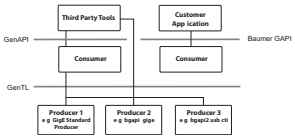


3. Central Idea Behind Baumer GAPI

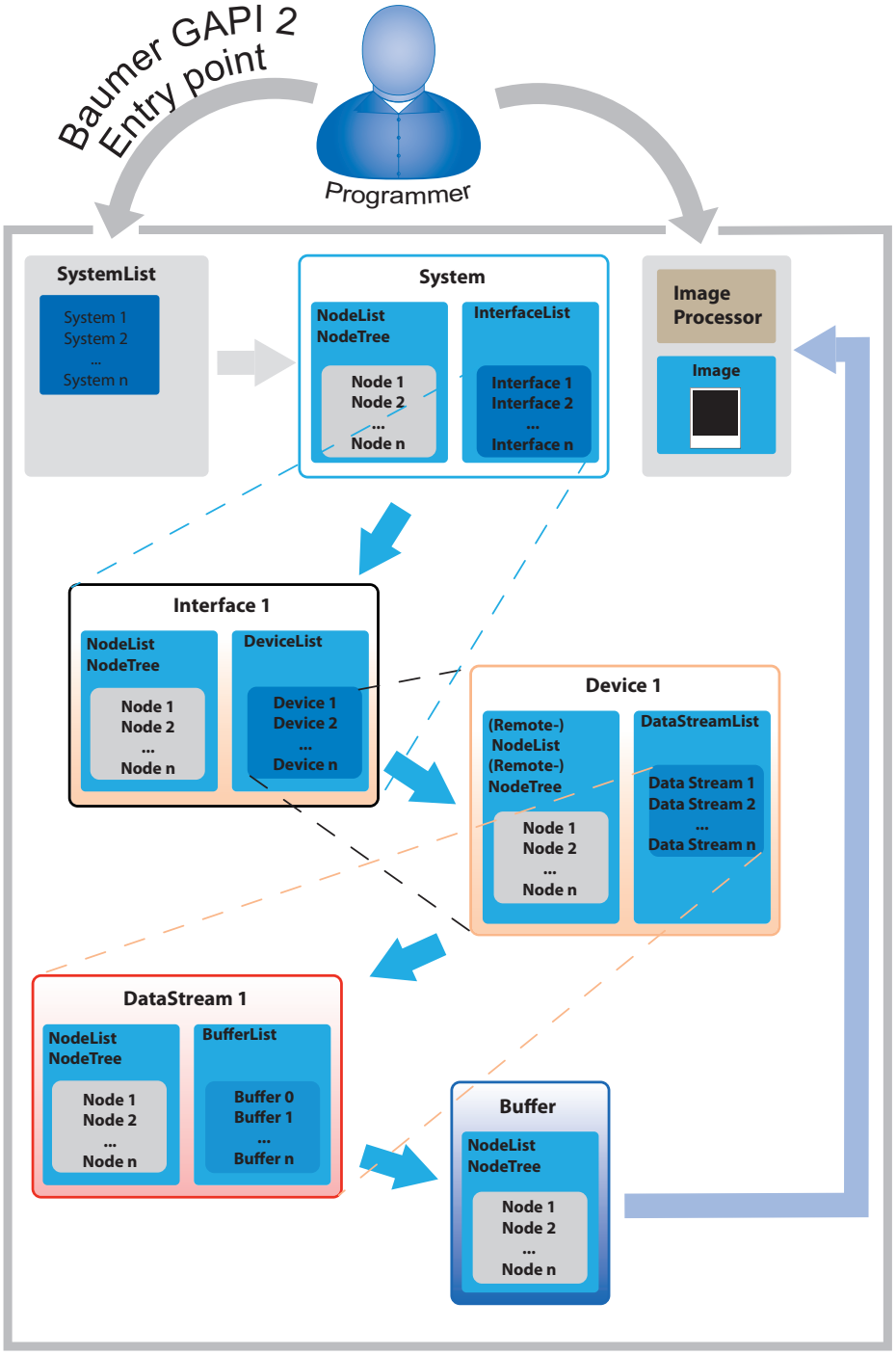
The fundamental idea behind the Baumer SDK was to release programmers from the need to define and instantiate all the required and usable objects, and to transfer these tasks to the Baumer GAPI.

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.



▲ 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

Parent class for:
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 *GAPI_SDK_HELP.chm* CHM 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 *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`.

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.
`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.
`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, and PnPEvents, which are generated by the interface object.

3.1.2.1 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.

3.1.2.2 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.

3.1.2.3 DataStreamEventControl

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

Functions of the DataStream EventControl class

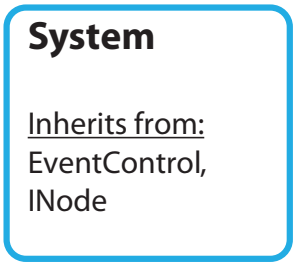
BGAPI2::DataStreamEventControl:: ...	
`RegisterNewBufferEvent();`	Enables standard compliant buffer events, specified within the xml description file.
`UnregisterNewBuffer-Event();`	Disables standard compliant buffer events, specified within the xml description file.
`GetFilledBuffer(bo_uint64 iTimeout);`	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



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 *(DeviceList).*
`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();`	Start command for parameter stacking (parameters will be queued).

`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

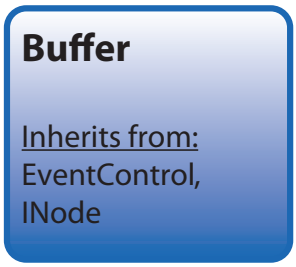
This class represents a data stream from the camera to the PC. It controls the data transfer and is responsible for buffer handling.

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-ChunkPayloadSize();`	Returns the delivered chunk payload size.
`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 *Buffer-List*.

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 ...
`erase(iterator& _where);`	Erases a list entry using the iterator.
`erase(const String& _keyval);`	Erases a list entry using the unique string identifier.
`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.

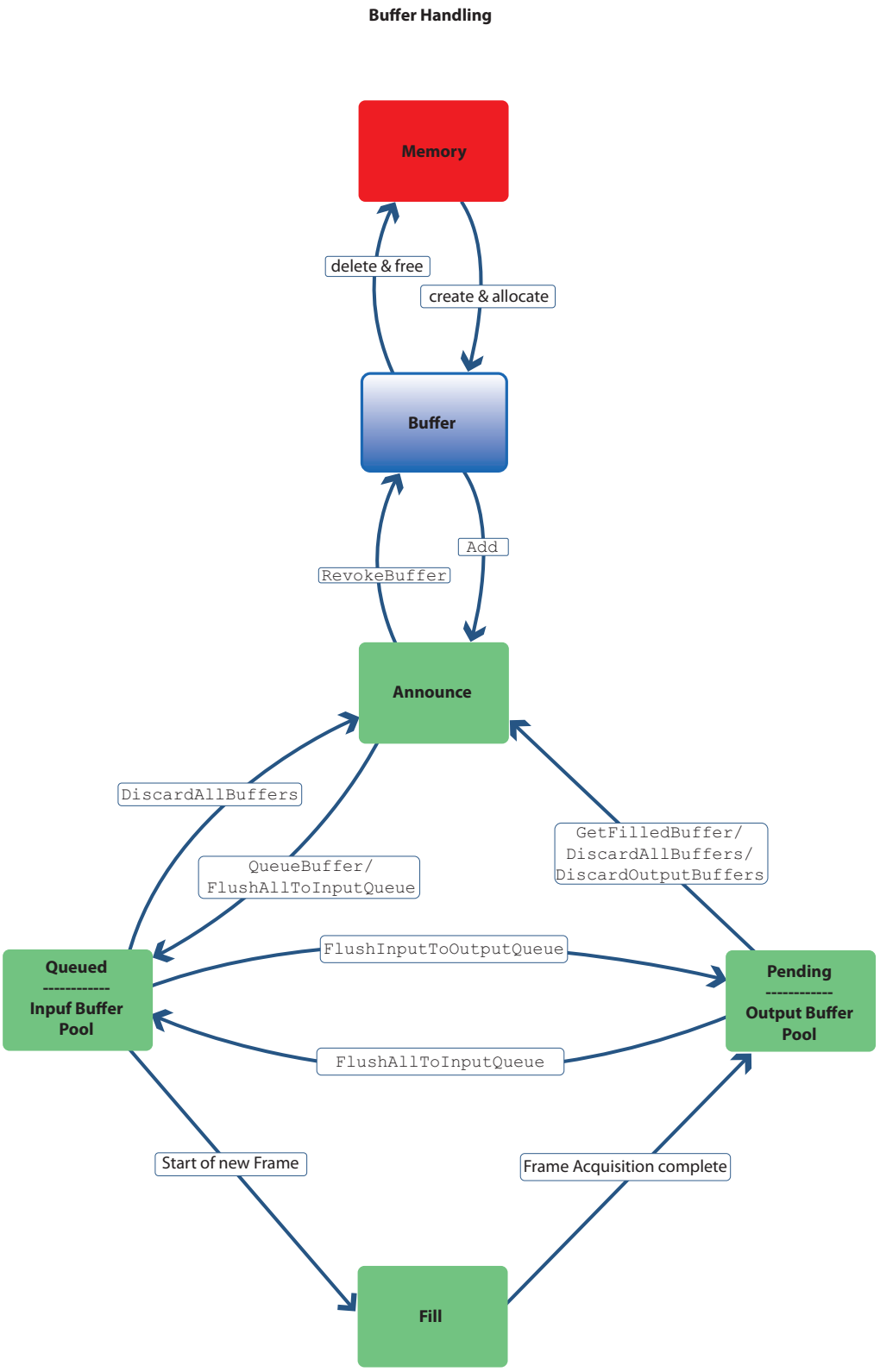
3.3.5 BufferList

BufferList Inherits from: 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.



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.
`TransformImage( String sPixelFormat, Image** ppOutputImage );`	Performs the image transformation.
`GetBuffer();`	Returns the image buffer.
`GetTransformBufferLength(String sPixelFormat);`	Returns the required buffer length of the image following transformation to a specified pixel format.
`TransformImageToBuffer( String sPixelFormat, void* pBuffer, bo_uint64 uBufferSize );`	Performs the image transformation using a user-allocated buffer of the previously determined length (GetTransformBufferLength).
`GetHistogram(bo_tHistRecords Histogram);`	Returns the histogram array of the image
`Release();`	Releases the occupied resources (excluding the user-allocated buffer).

3.4.2 Image Processor

Image Processor	This class creates image objects.
Inherits from: INode	**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 Image Processor class

BGAPI2::ImageProcessor:: ...	
`GetInstance();`	Creates a singleton instance of the class.
`ReleaseInstance();`	Releases the image processor instance.
`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.

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®

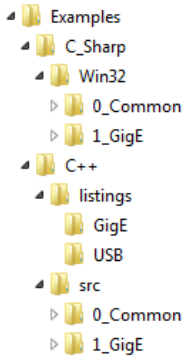
All required files were copied to your system during the installation process.

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)



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 compiler switch `D_GNULINUX` must be set for the gcc.

Details from the SDK examples:

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

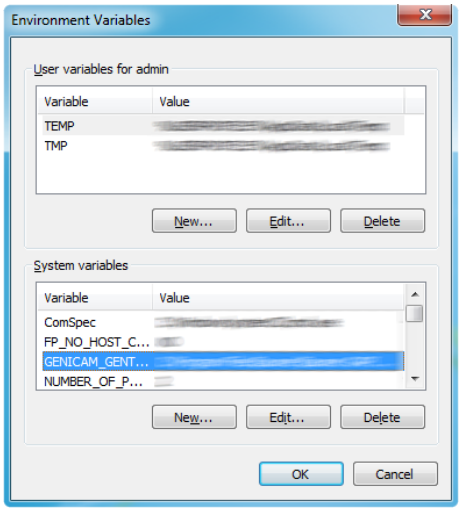
4.3 Setting System variables

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)*.

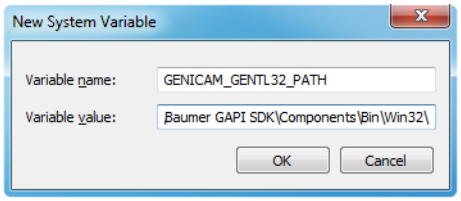
1. Click *Start,* then choose *Computer.*



2. Click *Properties.*
3. Select the *Advanced* tab.
4. Click *Environment Variables.*
5. Check that the entry GENICAM_ GENTL32_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***.
6. Click *New* within System variables.
7. Adjust the following.



   Variable name:

   GENICAM_GENTL32_PATH

   Variable value:

   32-bit systems

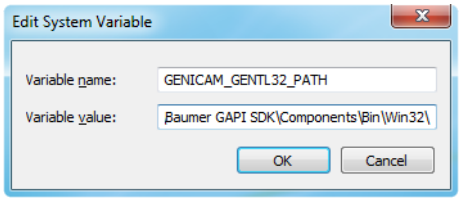
   `<BGAPI2 SDK>\Components\Bin\Win32`

   64-bit systems

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

   Continue with ***step 9***.

8. Select the GENICAM_GENTL32_ 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`.

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

4.4 Implementation

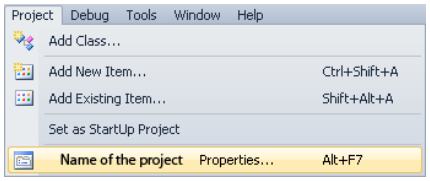
You will need to include different header files and libraries depending on the programming language and operating system. See the respective precautions for your operating system and programming language.

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”:



- “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\*.* .\
```

Notice

32 Bit:

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

bgapi2_genicam.dll
bgapi2_img.dll
BO_GigEFilterDrv.dll
MathParser.dll
```

64 Bit

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

bgapi2_genicam.dll
bgapi2_img.dll
BO_GigEFilterDrv.dll
MathParser.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.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.

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

```
#include "bgapi2_genicam.hpp"
```

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.

4.4.1.4 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
```

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
```

Requirements for Cmake (program to control the software compilation process) and GCC (Compiler)

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# ./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
```

4.4.2.4 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
```

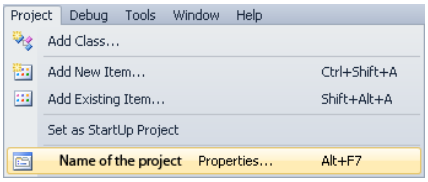
4.4.3 Implementation in C# (Windows)

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...”:



- “Build Events” → “Post-Build-Event”: “Command Line”:

absolute path:

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

or relative path:

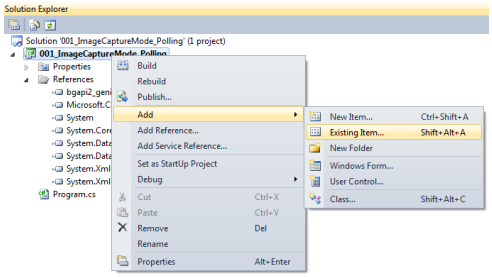
```
relative path: copy "..\..\..\..\..\..\..\Bin\Win32"\*.* .\
```

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:



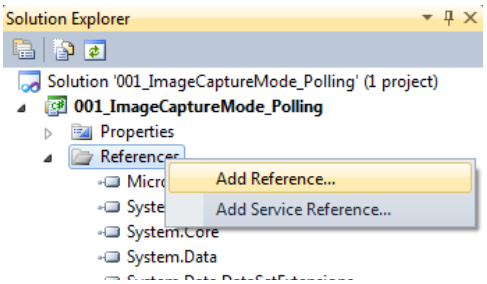
"Add → Existing Item...".

Add the respective DLL files.

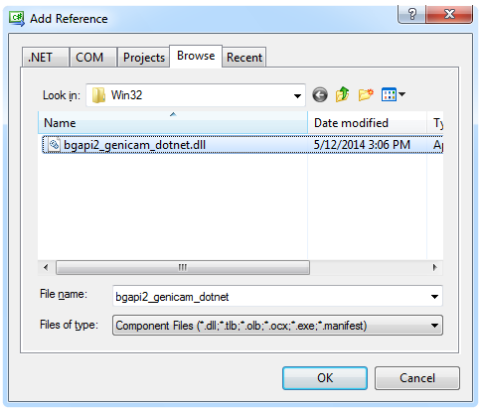
4.4.3.2 .NET Component

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

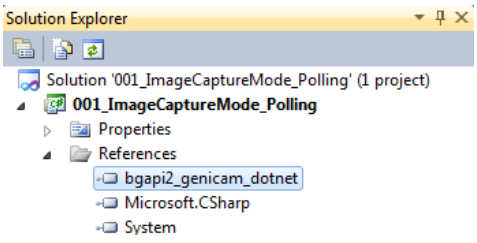
"Add Reference...".



Integrate the respective `bgapi2_genicam_dotnet.dll`.



Now you can see the integrated `bgapi2_genicam_dotnet.dll` under *References*.



Notice

32-Bit:

`<BGAPI SDK>\Components\Dev\C_Sharp\Win32`

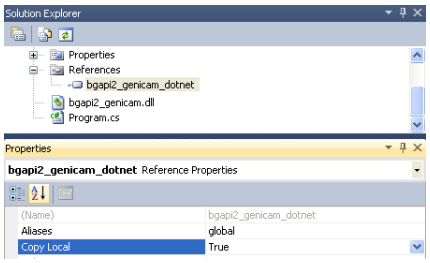
`bgapi2_genicam_dotnet.dll`

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".



Notice

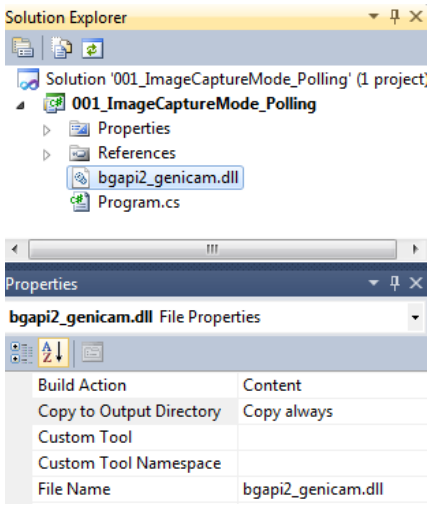
32-Bit:

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

bgapi2_genicam.dll
bgapi2_img.dll
BO_GigEFilterDrv.dll
MathParser.dll
```

64-Bit

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

bgapi2_genicam.dll
bgapi2_img.dll
BO_GigEFilterDrv.dll
MathParser.dll
```

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



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.

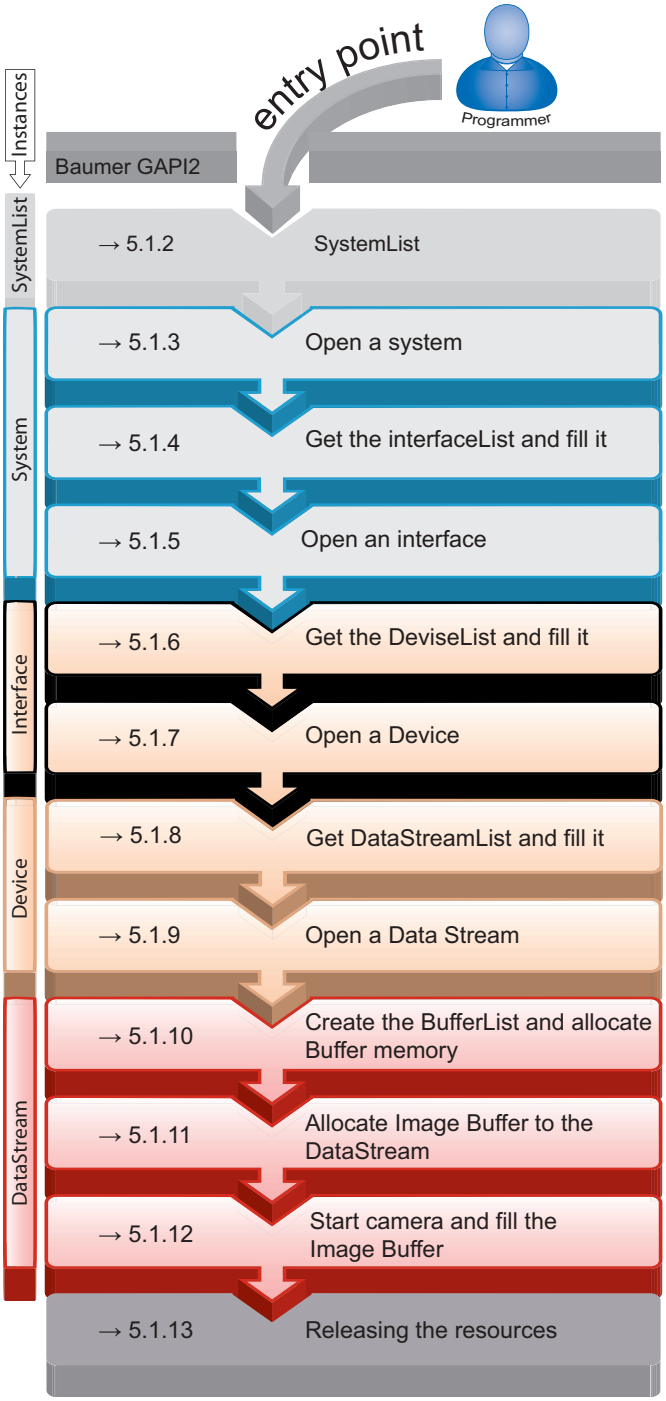
This tutorial assumes you are working with one interface and one camera in **C++** and **C#**.

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.



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.hpp"
using namespace BGAPI2;
```

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.

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

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

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

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.

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);
```

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();
```

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::GetDelivered Count()

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::GetChunkNode Tree()::GetNode("Chunk FrameID")::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*.

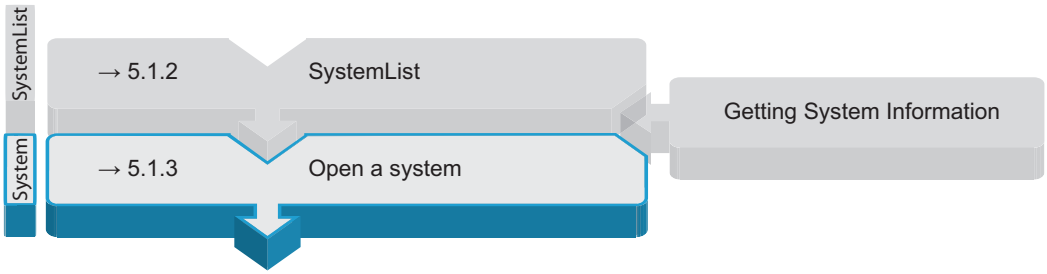
This information is necessary to distinguish between multiple located objects and make the correct choice. (e.g. correctly selecting the interface card to which the camera is connected).

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++

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

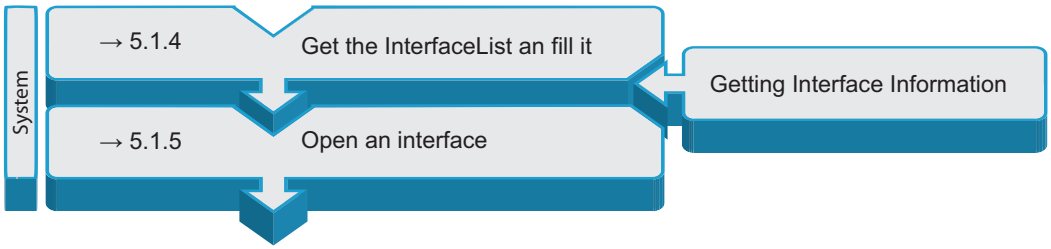
C#

```
foreach (KeyValuePair<string, BGAPI2.System> sys_pair inBGAPI2.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



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;
}
```

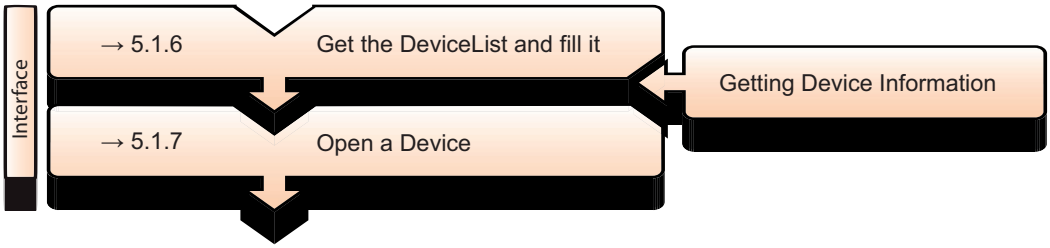
C#

```
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;
}
```

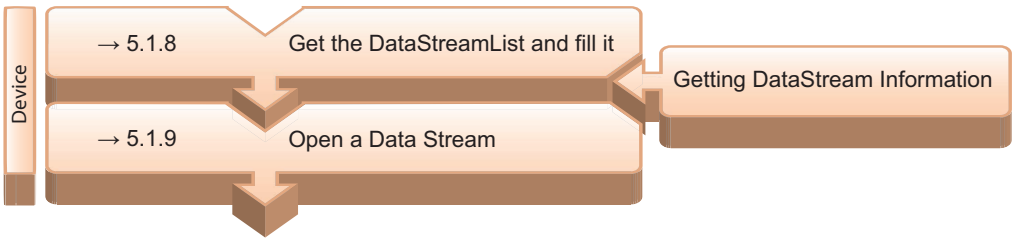
C#

```
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



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#

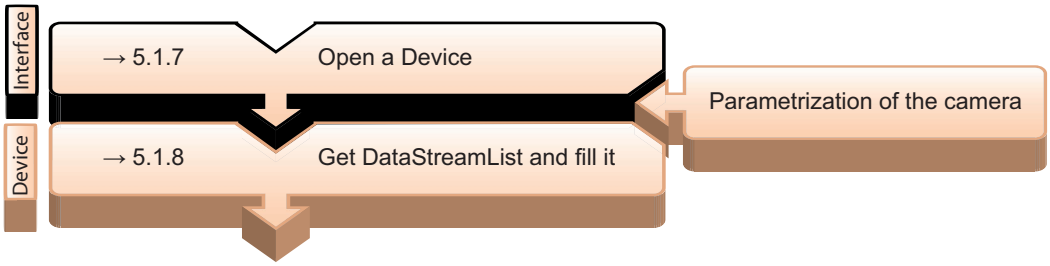
```
foreach (KeyValuePair<string, BGAPI2.DataStream> dst_pair in datastreamList)
{
     System.Console.Write("DataStream ID: {0}\n", dst_pair.Key);
}
```

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, *AcquisitionFrameRateEnable* 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}) is longer
        than MaxStringthLength ({1})\n", newDeviceUserID.Length, 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 = (double)mDevice.RemoteNodeList["Width"].Min;
long iImageWidthMax = (double)mDevice.RemoteNodeList["Width"].Max;

//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")->SetValue(1);`

C#

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

Write: `mDevice.RemoteNodeList["TriggerSource"].EnumerationValue = (long)1;`

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->GetImplemented() == true) &&
              (pNode->GetAvailable() == true) &&
              (pNode->GetLocked() == false) &&
              (pNode->GetVisibility() == "Invisible") )
     {
              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.IsAvailable == true) &&
              (mNode.IsImplemented == true) &&
              (mNode.IsLocked == false) &&
              (mNode.Visibility != "Invisible"))
     {


              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 and VLG-40C.

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;
```

C++

```
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)mDeviceRemoteNodeList["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
```

6. Support

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 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