

BASLER eXcite



API Reference

Document Number: DA00074903

Release Date: 13 April 2006

For customers in the U.S.A.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

You are cautioned that any changes or modifications not expressly approved in this manual could void your authority to operate this equipment.

The shielded interface cable recommended in this manual must be used with this equipment in order to comply with the limits for a computing device pursuant to Subpart J of Part 15 of FCC Rules.

For customers in Canada

This apparatus complies with the Class A limits for radio noise emissions set out in Radio Interference Regulations.

Pour utilisateurs au Canada

Cet appareil est conforme aux normes Classe A pour bruits radioélectriques, spécifiées dans le Règlement sur le brouillage radioélectrique.

Life Support Applications

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Basler customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Basler for any damages resulting from such improper use or sale.

Warranty Note

Do not open the housing of the camera. The warranty becomes void if the housing is opened.

All material in this publication is subject to change without notice and is copyright Basler Vision Technologies.

Contacting Basler Support Worldwide

Europe:

Basler AG
Ander Strusbek 60 - 62
22926 Ahrensburg
Germany

Tel.: +49-4102-463-500
Fax.: +49-4102-463-599

vc.support.europe@baslerweb.com

Americas:

Basler, Inc.
740 Springdale Drive, Suite 100
Exton, PA 19341
U.S.A.

Tel.: +1-877-934-8472
Fax.: +1-877-934-7608

vc.support.usa@baslerweb.com

Asia:

Basler Asia Pte Ltd
8 Boon Lay Way,
#03-03 Tradehub 21
Singapore 609964

Tel.: +65-6425-0472
Fax.: +65-6425-0473

vc.support.asia@baslerweb.com

www.basler-vc.com

Contents

1	API Overview	1
1.1	Introduction	1
1.2	Aspects Common to all Classes	3
1.3	Controlling the eXcite from a Local Program	4
1.4	Transferring Images From an eXcite to the Outside	13
1.5	Library Files	25
2	Module Documentation	27
2.1	Exceptions	27
2.2	Device Manager	28
2.3	Camera	29
2.4	CamT	30
2.5	CXCamInterface	31
2.6	Types for the data members of 'Camera'	32
2.7	Non-enum types for the data members of 'Camera'	33
2.8	Enum types for the data members of 'Camera'	34
2.9	Image transfer from an eXcite to a PC	36
3	Namespace Documentation	37
3.1	BaslerCamera Namespace Reference	37
3.2	BaslerCamera::StreamServer Namespace Reference	39
3.3	GxClientInterface Namespace Reference	40
3.4	XCamInterface Namespace Reference	41
4	Class Documentation	43
4.1	BaslerCamera::CamT< TliDelegate, Apilmpl > Class Template Reference	43
4.2	XCamInterface::CEnumeration_ColorCodingEnums Class Reference	47
4.3	XCamInterface::CEnumeration_PioOut0MonitorEnums Class Reference	49
4.4	XCamInterface::CEnumeration_PioOut0SettingEnums Class Reference	50

4.5	XCamInterface::CEnumeration_PioOut0SrcEnums Class Reference	51
4.6	XCamInterface::CEnumeration_PioOut1MonitorEnums Class Reference	52
4.7	XCamInterface::CEnumeration_PioOut1SettingEnums Class Reference	53
4.8	XCamInterface::CEnumeration_PioOut1SrcEnums Class Reference	54
4.9	XCamInterface::CEnumeration_PioOut2MonitorEnums Class Reference	55
4.10	XCamInterface::CEnumeration_PioOut2SettingEnums Class Reference	56
4.11	XCamInterface::CEnumeration_PioOut2SrcEnums Class Reference	57
4.12	XCamInterface::CEnumeration_PioOut3MonitorEnums Class Reference	58
4.13	XCamInterface::CEnumeration_PioOut3SettingEnums Class Reference	59
4.14	XCamInterface::CEnumeration_PioOut3SrcEnums Class Reference	60
4.15	XCamInterface::CEnumeration_Strobe0PolarityEnums Class Reference	61
4.16	XCamInterface::CEnumeration_Strobe1PolarityEnums Class Reference	62
4.17	XCamInterface::CEnumeration_Strobe2PolarityEnums Class Reference	63
4.18	XCamInterface::CEnumeration_Strobe3PolarityEnums Class Reference	64
4.19	XCamInterface::CEnumeration_TestImageEnums Class Reference	65
4.20	XCamInterface::CEnumeration_TriggerModeEnums Class Reference	66
4.21	XCamInterface::CEnumeration_TriggerPolarityEnums Class Reference	67
4.22	XCamInterface::CEnumeration_TriggerSourceEnums Class Reference	68
4.23	XCamInterface::CEnumeration_VideoModeEnums Class Reference	69
4.24	GxClientInterface::CGxClientInterface Class Reference	70
4.25	XCamInterface::CXCamInterface Class Reference	71
4.26	BaslerCamera::DeviceInfo Class Reference	77
4.27	BaslerCamera::DeviceIoException Class Reference	79
4.28	BaslerCamera::DeviceManager Class Reference	80
4.29	GenApi::GenericException Class Reference	82
4.30	BaslerCamera::GxStreamServer Class Reference	83
4.31	GenApi::IBoolean Struct Reference	87
4.32	BaslerCamera::IDevice Struct Reference	88
4.33	GenApi::IFloat Struct Reference	89
4.34	BaslerCamera::IInDataStream Struct Reference	90
4.35	GenApi::IInteger Struct Reference	92
4.36	GenApi::InvalidArgumentException Class Reference	93
4.37	BaslerCamera::IOutDataStream Struct Reference	94
4.38	BaslerCamera::StreamServer::IRegisterSet Struct Reference	96
4.39	GenApi::IString Struct Reference	97
4.40	GenApi::LogicalErrorException Class Reference	98

4.41	GenApi::OutOfRangeException Class Reference	99
4.42	GenApi::PropertyException Class Reference	100
4.43	BaslerCamera::PropertySet Class Reference	101
4.44	GenApi::RuntimeException Class Reference	102
5	File Documentation	103
5.1	BaslerCam.h File Reference	103
5.2	CamT.h File Reference	104
5.3	DataStream.h File Reference	105
5.4	Device.h File Reference	106
5.5	Exception.h File Reference	107
5.6	GxClient.h File Reference	109
5.7	GxClientInterface.h File Reference	110
5.8	GxStreamServer.h File Reference	111
5.9	IBoolean.h File Reference	112
5.10	IFloat.h File Reference	113
5.11	IInteger.h File Reference	114
5.12	IString.h File Reference	115
5.13	XCam.h File Reference	116
5.14	XCamInterface.h File Reference	117

Chapter 1

API Overview

Please Note

The eXcite API is designed to comply with the GenICam interface standard for industrial digital cameras. The GenICam standard is near completion, but the final form has not yet been released. Any changes made to the standard between now and when it is finally released should have only a minimal impact on the eXcite API. You should be aware, however, that once the GenICam standard is released, there may be some small changes to the names of the method calls in the eXcite API. This may require you to make some changes to any code that you have written to run on the eXcite.

1.1 Introduction

The Overview section of the API Reference gives a top level view of the classes provided to the application programmer by the eXcite library and outlines how to use these classes in application programs.

The exact details of each API class (and its public member variables and functions) are separately documented for each class in the following chapters of the API Reference. Clicking on a class, function, or variable name in the Overview section will take you to the page in the Reference that documents your selection in detail.

Use Cases Overview

The eXcite library provides classes for the following two use cases:

- [Controlling the camera section](#) of the eXcite from an application program that runs "locally" on the processor of the eXcite.
- [Transferring image data](#) from an eXcite to a PC (or to another eXcite) over an Ethernet network.

The second of these involves two distinct kinds of application programs: a server program running on the eXcite and a client program running on the PC. In total, the eXcite library thus provides classes for the following three kinds of application programs:

- For controlling the eXcite from a local program:
 - **1. A program running on the processor of the eXcite**
- For transferring images from eXcite to outside:
 - **2a. A server program running on the processor of the eXcite**
 - **2b. A client program running on the PC**

The set of features (classes, include files, library files) from the eXcite library used for each of these three kinds of application programs is different. The relevant features of the eXcite library to use in each case is discussed, in order, in the API Overview below. At the end of the overview, the [Library Files](#) section lists all of the library files included in the Basler eXcite Library and the dependencies among the libraries are illustrated.

Terminology

data type - The overview uses the term "data type" in its broad, general sense. That is, the term "data type" (or "type" for short) is used to refer to any kind of data type in C++, whether it is a built-in type in C (such as `int` or `double`); a C/C++ `struct`; a C/C++ `enum`; or a C++ class.

library, API, eXcite library, library file - The term "software library", or "library" for short, is used in the overview in an abstract sense to denote a collection of software functionality (classes, functions, macros) made available to an application programmer. The "API" of the library is the interface that the library makes public to the application programmers, through which they can access the functionality in the library.

The software library delivered by Basler with the eXcite device is called the "eXcite library".

The concrete physical representation of the eXcite library consists of (1) a set of ".h" include files, plus (2) a set of "library files", which contain the compiled source code of the library in an archived form. Note that the term "library file" has a more specific meaning than the term "library". Note also that the "eXcite library" includes more than one library file.

camera section - As shown in the diagram at the beginning of the "Operation and Features" section of the User's Manual, the eXcite device consists of a camera and a PC-like processor connected to each other. We use the terms "camera section" and "processor section" to refer to the camera and PC-like components separately. To be specific, the "camera section" of the eXcite includes the image sensor, its micro-controller and connected clock electronics, the image buffer, and the digital I/O lines. The "processor section" of the eXcite includes the MIPS processor, the Flash memory, the SDRAM working memory, and the GigE, USB, and RS-232 interfaces.

1.2 Aspects Common to all Classes

1.2.1 Namespaces

To avoid possible name conflicts with other classes in your code, everything in the eXcite library sits inside namespaces. The API of the eXcite library makes use of the following namespaces:

BaslerCamera - This namespace contains all of the important "top level" classes of the eXcite library for all types of application programs. For example, this namespace contains the [DeviceManager](#) , [Camera](#) , and [GxClient](#) classes, as well as the [GxStreamServer](#) class. Thus, this namespace is used by all of the different kinds of application programs that use the eXcite library.

GenApi - This namespace contains "helper" types that are used by all types of application programs.

The GenApi namespace contains the Exception classes used by the eXcite library (see [Exceptions](#)). In addition, this namespace contains the [IBoolean](#), [IInteger](#), [IFloat](#), and [IString](#) data types, which are used by many of the member objects of the [Camera](#) class (see [The Member Objects of Camera](#)) and which are also used by many of the member objects of the [GxClient](#) class (see [Transferring Images From an eXcite to the Outside](#)).

CAMERANAMESPACE - This namespace is only used for application programs that control the camera section of an eXcite device. That is, this namespace is not used for application programs (either server or client) used for transferring image data from the eXcite to a PC.

This namespace contains the `xxxEnums` enum types used for many of the member objects (data members) of the [Camera](#) class (see [The member objects of Camera](#)).

[**Note:** [CAMERANAMESPACE](#) is actually a `#define` for the namespace name [XCamInterface](#) used internally in the eXcite library. Unfortunately, the tools used to generate the API Reference don't see that `#define` and as a result, the API Reference refers to [CAMERANAMESPACE](#) as [XCamInterface](#). However, application programmers using the eXcite library should use [CAMERANAMESPACE](#) in their code, not [XCamInterface](#).]

We recommend putting the following `using` statements at the head of any of your source files that use the eXcite library:

```
using namespace BaslerCamera;
using namespace GenApi;
using namespace CAMERANAMESPACE;
```

1.2.2 Exceptions

Any of the member functions in any of the classes in the eXcite library, including constructors and destructors, can throw exceptions. Therefore, most application programs would want to put all places where they use or instantiate any of the eXcite library classes inside `try...catch` constructs, like this:

```
try
{
    ... use eXcite library class(es) ...
}
catch ( GenApi::GenericException & e )
{
    cerr << "Error occurred: " << e.GetDescription() << endl;
    cerr << "(" << e.GetSourceFileName()
        << ":" << e.GetSourceLine() << endl;
}
```

Each exception thrown by the eXcite library is a subclass of:

`GenApi::GenericException`

which, in turn, is a publicly inherited subclass of `std::exception`.

See the [Exceptions](#) section in the API Reference for a description of all exception classes and of their member functions.

1.3 Controlling the eXcite from a Local Program

This section outlines how to use the eXcite library to control the camera section of the eXcite from an application program that runs on the eXcite's processor.

1.3.1 Include File

To make all of the necessary features from the API of the eXcite library available to a program running on the eXcite for controlling the camera section of the eXcite, the only `.h` file you need to include is `<BaslerCam.h>`.

Before the `#include <BaslerCam.h>` preprocessor statement, you must set the `#define USE_XCAM`. Thus, at the beginning of all of your source files that use the eXcite library, you must include the following two lines:

```
#define USE_XCAM
#include <BaslerCam.h>
```

When compiling your application, pass the

```
-I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include
```

switch to the compiler.

Example:

```
mips-linux-gnu-g++ -c -I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include
-o Simplegrab.o simplegrab.cpp
```

1.3.2 Library Files

The objects described in this section are implemented in the `libxcam` and `libbaslercam` library files.

When linking your application dynamically, use the `-lxcam` compiler switch. Example:

```
mips-linux-gnu-g++ -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib
-o Simplegrab -lxcam simplegrab.o
```

When linking your application statically, several libraries must be linked to your application. Example:

```
mips-linux-gnu-g++ -static -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib
-o Simplegrab simplegrab.o -lxcam -lbasercam -lpthread
```

1.3.3 Using the DeviceManager Object

The purpose of the [DeviceManager](#) object is to create a [Camera](#) object.

This [Camera](#) object is of central importance to application programs, i.e., the application program operates the camera section of the eXcite device through the [Camera](#) object. In this subsection, we describe how to create (instantiate) a [Camera](#) object. The [Using the Camera Object](#) subsection describes how to actually use the [Camera](#) object.

Open-Camera and Close-Camera Blocks

Using the following block of code will cause the [DeviceManager](#) to create a [Camera](#) object through which you can operate the camera:

```
// (1) Get the device manager
DeviceManager & dm = DeviceManager::GetInstance();

// (2) Let the device manager enumerate all devices
DeviceManager::DeviceInfoList_t lstDevices(
    dm.EnumerateDevices( Camera::DeviceTypeId ));

// (3) Check if we found any devices
if ( lstDevices.size() == 0 )
    throw RUNTIME_EXCEPTION( "No devices found" );

// (4) Let the device manager create the first device
Camera* pCamera;
pCamera = dynamic_cast<Camera*>(
    dm.CreateDevice( *lstDevices.begin() ) );

// (5) Check if we got what we expected
if ( NULL == pCamera )
    throw RUNTIME_EXCEPTION( "Device isn't a camera device" );

// (6) Open device
pCamera->Open();
```

This is a standard block of code that most application programs would include as shown. We will refer to the above block of code as the "open Camera" block.

There is also a corresponding "close Camera" block. This is another standard block of code that most application programs would include as shown to close the [Camera](#) object once they are done with it. The "close Camera" block is the following two-line block of code:

```
// close the device
pCamera->Close();

// let the device manager destroy the device
dm.DestroyDevice( pCamera );
```

To see these standard code blocks used in context in an actual application program, please take a look at the Simplegrab code sample (see the Simplegrab/simplegrab.cpp source file in the set of code samples).

The Return List of EnumerateDevices()

The eXcite library features that you see used internally in the "open Camera" and "close Camera" code blocks do have other uses. However, these other uses of the features that we see at work inside the "open Camera" and "close Camera" blocks really only apply to the use case discussed in the [Transferring Images From an eXcite to the Outside](#) section. In that section, we will discuss these features as needed.

At this point, we only explain the basics about the internals of the "open Camera" block.

The `DeviceManager::EnumerateDevices()` function actually returns a list, namely a list of objects of the `DeviceInfo` type. (The `DeviceManager::DeviceInfoList_t` type is simply a typedef for `std::list<DeviceInfo>`.)

If `USE_XCAM` is defined (conforms to the [Include File](#) subsection), then the list returned by the `DeviceManager::EnumerateDevices()` function always contains exactly one item. Passing this list item to the `DeviceManager::CreateDevice()` function will create a `Camera` object through which to access and operate the camera section of the eXcite on which the application program is running.

[**Note:** The API is organized this way, with `EnumerateDevices()` returning a list instead of simply one single item, to allow the same API to be used for cases where the `EnumerateDevices()` function will return more than one item. These cases are described in the [Transferring Images From an eXcite to the Outside](#) section.]

Exceptions

Note that both the "open Camera" block and the "close Camera" block can throw exceptions, so that most application programs would want to put these blocks inside `try...catch` constructs (as explained in the [Exceptions](#) section).

In the "open Camera" block, we see two exceptions being thrown explicitly from the code block, namely by the two lines in which the `RUNTIME_EXCEPTION()` macro is used. This macro instantiates a `GenApi::RuntimeException` object with the macro parameter as its "Description."

[Application programmers can, if desired, also use any of the other similar `xyz_EXCEPTION()` macros defined by the eXcite library to instantiate and throw any of the exceptions from the eXcite library.]

The two exception-throws explicitly coded (by means of the `RUNTIME_EXCEPTION` macro) in the "open Camera" block are not the only exceptions that may be thrown by executing the code block. Any of the member functions (including constructors and destructors) of any of the classes from the eXcite library may throw exceptions.

1.3.4 Using the Camera Object

The camera section of a physical eXcite device is represented in application code by an object of the `Camera` type. A `Camera` object is used as a "handle" through which to operate one physical eXcite device. From an abstract point of view, this is similar to the way that a C program using the `<stdio.h>` standard library uses a `FILE *` pointer as a handle through which to operate on a file.

The camera is operated by application programs through:

- Member functions (= methods) of the `Camera` object

Example:

```
pCamera->PrepareGrab();
```

The [The Member Functions of Camera](#) subsection below gives an overview of the member functions and how to use them.

- Member objects of the `Camera` object. These member objects also each have their own methods (= member functions).

Example:

```
pCamera->Shutter.SetValue( ... );
```

The [The Member Objects of Camera](#) subsection below gives an overview of the member objects and their methods and how to use them.

The collection of code samples supplied with the eXcite includes complete example application programs that illustrate how the various member functions and member objects of [Camera](#) are typically used, including all initialization steps that may be necessary (such as opening the [Camera](#) object and setting camera parameters).

1.3.4.1 Where to Find the Camera Members in the API Reference

Your application code should use the [Camera](#) object to operate the eXcite device as described in this overview and as illustrated by the code samples. Internally, via one or more typedefs, the [Camera](#) object is implemented as an instantiation of a template class named [CamT](#). The [CXCamInterface](#) class is one of the most important base classes from which this [CamT](#) template class inherits.

In the API Reference, the member functions and member objects of the [Camera](#) class are presented as follows:

- The member functions of [Camera](#), as briefly described in the [The Member Functions of Camera](#) section of this overview, are presented as the members of the [BaslerCamera::CamT](#) template class.
- The member objects of [Camera](#), as briefly described in the [The Member Objects of Camera](#) section of this overview, are presented as the members of the [CXCamInterface](#) class.

The reader should bear in mind that the [BaslerCamera::CamT](#) template class and the [CXCamInterface](#) class are internals to the eXcite library. The members of these internal (template) classes appear as direct members in the user [Camera](#) object and should be accessed by application code through the [Camera](#) object only.

1.3.4.2 The Member Functions of Camera

The member functions of [Camera](#) conceptually fall into two kinds of functions:

1) Functions to open/close the [Camera](#) object:

Function call	Purpose
<code>pCamera-> Open() ;</code>	Open Camera object
<code>pCamera-> Close() ;</code>	Close Camera object
<code>bool myBool = pCamera-> IsOpen() ;</code>	Query whether Camera object is open

The purpose and usage of these functions is described in the [Using the DeviceManager Object](#) section (see the "open/close Camera" code blocks).

2) Functions to grab images:

Function call	Purpose
<code>pCamera-> PrepareGrab();</code>	Initialize for grabbing images
<code>pCamera-> FinishGrab();</code>	De-initialize image grabbing
<code>bool myBool = pCamera-> IsReadyForGrab();</code>	Query whether device is ready for grab
<code>pCamera-> QueueBuffer(nBytes, pBuffer, pUser);</code>	Pass to the API the address of a user buffer(s) for the result image(s) to be written
<code>IInDataStream::BufferStatus bs = pCamera-> WaitForBuffer(ppBuffer, ppUser, timeout_ms);</code>	Wait for an image grab to complete and retrieve the grabbed image

For a description of how to use this set of functions, see the `SimpleGrab/simplegrab.cpp` and `MultiGrab/multigrab.cpp` sample code source files and the description of the SimpleGrab and MultiGrab samples in the eXcite User's Manual. The `BaslerCamera::CamT` page in the API Reference contains a full description of the arguments and return values for these functions.

The application programmer should pay special attention to how the user buffer(s) are passed into and retrieved from the functions "QueueBuffer()" and "WaitForBuffer()".

In the case where you want to quickly grab a sequence of images, the application program should use multiple user buffers. Before the image grabbing begins, all of the buffers should be "queued in" (by calls to the "QueueBuffer()" function), that is: registered to the eXcite library. Once image grabbing is started, the user buffers are then filled with image data as soon as possible and as long as the eXcite library has unused user buffers available.

The SimpleGrab and MultiGrab samples, along with their description texts in the User's Manual, show and explain how this buffer queueing and retrieving should be done. The use of multiple user buffers is shown (and explained in) the MultiGrab sample.

Note that apart from the above member functions of `Camera`, the `OneShot`, `ContinuousShot`, and `SoftwareTrigger` member *objects* of `Camera` are also directly concerned with the control of image grabbing. These are explained in the next section.

1.3.4.3 The Member Objects of Camera

The purpose of the member objects of `Camera` is twofold:

- To set and read camera parameters
- To command image capture (grabbing) to start or stop

For a full discussion of the purpose and function of the camera's parameters, see the "Operation and Features" section of the eXcite User's Manual.

Each camera parameter is represented in the `Camera` object as a member object (data member) of the `Camera` class.

[**Note:** The members of `Camera` described in this section are actually references to objects elsewhere that are managed internally in the eXcite library. These references are initialized for the API user by the "open Camera" block described in [Using the Device Manager object](#). The application programmer can use these members as if they were simply objects. Accordingly, we will refer to them in this overview as the "member objects" (or, synonymously, the "data members") of the `Camera` class.]

The member objects of `Camera`, in turn, have member functions (= methods) through which the value of the parameters represented by the member objects can be accessed. The precise set of methods for a member object depends on the type (data type) of the member object. However, the most basic methods are present in all member objects.

In the text below, we first present an overview of the [data types](#) that are used for the member objects, followed by an overview of the most important [methods of the member objects](#). Next, we explain [where to find more information about each member object](#). The section is concluded by a few remarks about some of the more important subsets of member objects.

Data Types of the Member Objects

The data types used for the member objects of [Camera](#) are:

- `IBoolean`, `IInteger`, `IFloat`, `IString`

[**Note:** These data types sit in the `GenApi` namespace, that is, refer to them in your code by `GenApi::IBoolean`, `GenApi::IInteger`, `GenApi::IFloat`, `GenApi::IString` unless you use a `using` statement as described in the [Namespaces](#) section.]

These data types are similar to the C++ `bool`, `int`, `double`, and `std::string` data types, respectively. But because The "IInteger" parameter has some additional features compared to a normal "int", it needs some additional remarks.

In the context of Basler cameras, "IInteger" parameters are also called **scalar** parameters. The features or characteristics that the "IInteger" data type has in addition to the behavior normally expected of an "int" type, are as follows: in general, the value i of an "IInteger" parameter is limited to a specific interval, and within this interval, its value is limited to only every N th integer number. More exactly described, its possible values i are:

$$i = i_{\min} + N k$$

$$i_{\min} \leq i \leq i_{\max}$$

where k is any integer, N is called the "increment", and $i_{\min}$ and $i_{\max}$ are called the interval bounds. The increment and interval bounds are fixed, read-only properties of the "IInteger" member object. The values of the increment and interval bounds for each "IInteger" member object can be queried by the application program, as described below under the [Methods of the Member Objects](#) heading.

- `xxxEnums` types

where the `xxx` in the name is a short designation of the camera parameter.

[**Note:** These data types sit in the `CAMERANAMESPACE = XCamInterface` namespace. That is, refer to them in your code by `CAMERANAMESPACE::xxxEnum` unless you use a `using` statement as described in the [Namespaces](#) section.]

These data types are similar to `enum` types in C/C++ in that a member object of one of the `xxxEnums` data types can only have one of a specific set of values.

All the `xxxEnums` types are listed on the [Enum types for the data members of 'Camera'](#) page in the API Reference. Clicking on any of the types takes you to a page that documents the set of values a member object of that type can have.

Usage example for these `xxxEnums` values:

```
pCamera->TriggerMode =
    CEnumeration_TriggerModeEnums::TriggerMode_TriggerMode0;
```

Methods of the Member Objects

This subsection provides an overview of the most important methods. Not all of the available methods are mentioned below. Those that are not mentioned here are either of lesser importance to the application programmer or are intended only for internal use within Basler. (The full set of methods for any of the member objects can be found on the page for that member object in the API Reference.)

The following four methods are present in all of the member objects of [Camera](#):

```
TYPE GetValue( void );
```

```
TYPE operator()( void );
```

(These two methods operate identically and return the value of the camera parameter.)

```
void SetValue( TYPE arg1 );
```

```
TYPE & operator=( TYPE arg1 );
```

(Either of these two methods can be used, depending on your personal preference, to set the camera parameter to the value `arg1`.)

[**Note:** For read-only parameters, these two methods are present, but are implemented so that calling them causes an exception to be thrown.]

The following two methods are present only in the member objects of [Camera](#) that are of one of the `xxxEnum` data types:

```
std::string ToString( void );
```

```
void FromString( const std::string & ValueStr );
```

These functions are alternatives to the `GetValue()` and `SetValue()` members, respectively. The difference is that they read and set the value of the "enum" member object as a string value.

The `FromString()` method converts a string to one of the possible enum values of the member object and sets the member object to that value. The `ToString()` method returns the enum value of the member object represented as a human-readable string value.

The following three methods are only implemented for "Integer" parameters ("Integer" parameters are also called [scalar](#) parameters):

```
TYPE GetMin( void );
```

```
TYPE GetMax( void );
```

(These methods return the interval bounds, $i_{\min}$ and $i_{\max}$, respectively.)

```
TYPE GetInc( void );
```

(This method returns the increment N .)

Where to Find More Information About Each Member Object

More information about each of the individual member objects of "Camera" can be found at the following two places:

- For a full list of all the member objects, please see the [CXCamInterface](#) page in the API Reference. That page in the API Reference is the central "hub" for detailed information about each of the member objects. The page briefly describes the function of each member object, and states the data type for each member object.

For each member object, clicking on the data type of the member object takes you to a page that lists and describes the specific set of methods available for that member object. For member objects of one of the `xxxEnums` types, the page also documents all the valid values for the member object.

- For a full description of the function and purpose of each member object, please refer to the Operation and Features section of the User's Manual. That section in the User's Manual describes and explains the usage of the member objects from a conceptual point of view.

Member Objects Used to Control the Start of Exposure and Image Transfer

The member objects of Camera that are used to control the start of exposure and image transfer are as follows:

- `pCamera->OneShot`
- `pCamera->ContinuousShot`
- `pCamera->SoftwareTrigger`

`pCamera->OneShot` operates the camera in "one-shot" mode, i.e.,

```
pCamera->OneShot = true;
```

is used to start exposure and transfer of a single image. If the camera is configured in "shot only" mode (i.e., triggering is disabled), setting `OneShot` to true will immediately start an exposure. When the camera is configured to use a software trigger or an external trigger, setting `OneShot` to true will prepare the camera to start exposure and image transfer. Exposure and transfer of image start when the appropriate trigger signal arrives. The `OneShot` parameter resets itself to false when the exposure has finished.

`pCamera->ContinuousShot` operates the camera in "continuous-shot" mode, i.e.,

```
pCamera->ContinuousShot = true;
```

lets the camera continuously expose and transfer images. If the camera is configured in "shot only" mode (i.e., triggering is disabled), the exposure of the first image begins when `ContinuousShot` is set to true and subsequent exposures begin automatically. If software triggering or external triggering is enabled, a new image is exposed and transferred each time the appropriate trigger signal arrives. Image exposure and transfer are stopped by setting `ContinuousShot` to false. For example:

```
pCamera->ContinuousShot = false;
```

If the software trigger feature is enabled as described in the User's Manual and if `ContinuousShot` is set to true, The `SoftwareTrigger` member object is used to issue a software trigger. An eXcite configured this way will expose and transfer an image each time `SoftwareTrigger` is set to true.

Member Objects for I/O Control

The member objects of "Camera" for controlling the eXcite's four digital input ports (= physical input ports) and the four digital output ports (= physical output ports) are:

`PioInput` (read only) – Reads the state of all four of the digital inputs simultaneously

`PioOutput` – Reads or sets the state of all four of the digital outputs simultaneously

`PioOut$Monitor` – Reads the current state of one digital output port

`PioOut$Setting` – Sets the state of one digital output port

`PioOut$Src` – Selects the source signal for one digital output port

`PioOut$Invert` – Sets the invert function for one digital output port

where \$ = 0, 1, 2, 3.

The "PioInput" and "PioOutput" member objects access (read and/or write) all four input or all four output ports simultaneously. These two member objects are of the "Integer" data type. The value (state) of the four ports is represented in this integer as the four least-significant bits. The "Operation and Features" section of the User's Manual explains the correspondence (mapping) between the integer value and the individual ports in detail.

Each of the other digital I/O control member objects accesses or controls only one specific individual digital output port. [**Note:** Only the digital *output* ports can be accessed individually. There are no member objects for reading the state of only one individual digital *input* port.]

Each member object for individually accessing/controlling a single output port has a numeral in its name. This number corresponds to the number of the output port that is operated by that member object. For example, the "PioOut0Setting" member object sets the state of output port 0, the "PioOut1Setting" member object sets the state of output port 1, and so on. Since there are four output ports, the member objects for operating one individual output port thus come in sets of four. Each such set of four related output control member objects is represented in the brief listing above with the numeral in the member object name replaced by a \$ sign.

Each of the *output* ports has two kinds of **configuration** information attached to it:

- The port can be assigned to take its value from one of the following **sources**:
 - The internally generated "Trigger Ready" signal.
 - The internally generated "Integrate Enable" signal.
 - Internally generated strobe control signals.
Digital output line "N" can be tied to internal strobe control signal "N" (N = 0, 1, 2, 3) only.
 - User settable. This means that the value of the output port can be set by an application program, by means of the "PioOutput" or "PioOut\$Setting" member objects.
Note carefully that the "SetValue()" method of these member objects only has effect on those output lines that are in "user settable" mode.

In the above, the phrasing "assigned" means that the output port is connected to an internally generated signal, and takes the value of this internal signal. That is, any state change in the internal signal is transferred immediately to the output port.

Changing the association of output port to one of these sources is done through the "PioOut\$Src" member objects.

- The port can be configured as **inverting or non-inverting**.

By default, each port is in non-inverting mode. The inversion mode can be changed through the "PioOut\$Invert" member objects.

Inversion is done as the *last* step during the transfer of the signal to the output port. That is, when a port is in inverting mode and is assigned as "user settable", writing a "1" to the port will make the port low. When the port is in inverting mode and tied to one of the other internal signals, when the internal signal goes high, the output port value will go low.

More detailed information about I/O port control can be found in the "Operation and Features" section of the User's Manual.

The "SimpleDio" code sample included with the eXcite illustrates the use of most of the digital I/O features in an actual application program.

1.4 Transferring Images From an eXcite to the Outside

1.4.1 Overview

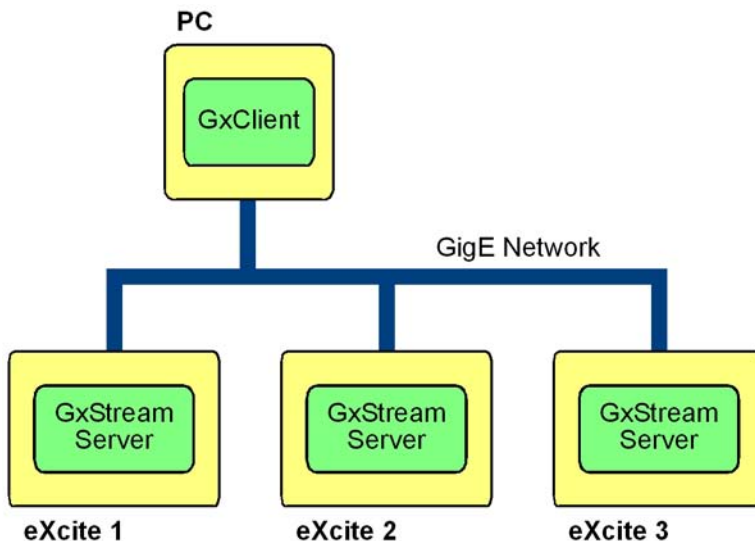
The eXcite library contains classes to build application programs for transferring images from an eXcite device to a PC via an Ethernet network connection.

[**Note:** The "PC" that the image data is transferred to can be any computer, including another eXcite. The eXcite library classes described in this section are usable for transferring image data not only to a normal PC but also to another eXcite.]

The image transfer classes in the eXcite library are intended to be used where the eXcite and the receiving PC are connected by Ethernet network. The simplest use case is with a single eXcite in the network:



It's also possible to connect two or more eXcite devices to a PC via an Ethernet network and to transfer image data from any of the eXcite devices, or from two or more of them simultaneously, to the PC:



The image transfer mechanism built using the eXcite library classes is always client-server based, and is organized as follows:

On the PC, an application program runs that **gets** (receives) the image data. On the eXcite, an application program runs that acts as a data source for the image data and that **sends** the image data.

The data-sourcing program on the eXcite is called the **server** and the data-getting program on the PC is

called the **client**.

The server program on the eXcite is the passive component. It is started first, then waits for incoming requests from a client and starts sending image data when such a client request is received.

The client program is the active component. It looks for an image-sourcing server in the network, then opens a connection to it and requests the server to start sending it image data.

The image-transfer classes in the eXcite library provide full functionality for all these tasks. When using these classes, the application programmer doesn't have to deal with (or know of) the underlying network layers used for communication over the Ethernet network.

The image-transfer classes provided by the eXcite library are:

Class	For use in
GxClient	the client program on the PC
GxStreamServer	the server program on the eXcite

1.4.2 Include Files

For applications implementing the server part, add the following include statement to your source file(s):

```
#include <gxdevice/GxStreamServer.h>
```

For applications implementing the client part, add the following include statements to your source file(s):

```
#include <device/DeviceManager.h>
#include <gxdevice/GxClient.h>
```

When compiling your application for the MIPS processor, pass the

```
-I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include
```

switch to the compiler.

Example:

```
mips-linux-gnu-g++ -c -I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include
-o StreamingServer.o StreamingServer.cpp
```

To compile your client application for a Linux PC, pass the

```
-I/opt/excite-tools/host/include
```

switch to the compiler.

Example:

```
g++ -c -I/opt/excite-tools/host/include -o StreamingClient.o StreamingClient.cpp
```

1.4.3 Library Files

Applications using the GxClient and GxServer classes must be linked against the libgxdevice library file.

The library files used for applications running on the MIPS processor can be found at: `/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib`. The library files used for applications running on a Linux PC are located at: `/opt/excite-tools/host/lib`.

When linking your application dynamically, use the `-lgxdevice` compiler switch.

Examples:

```
mips-linux-gnu-g++ -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib
-o StreamingServer StreamingServer.o -lgxdevice

g++ -L/opt/excite-tools/host/lib -o StreamingClient StreamingClient.o -lgxdevice
```

When linking your application statically, several other libraries must also be linked.

Examples:

```
mips-linux-gnu-g++ -static -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib
-o StreamingServer StreamingServer.o -lgxdevice -lbaslercam -lgxpp -lgx -lpthread

g++ -static -L/opt/excite-tools/host/lib -o StreamingClient StreamingClient.o
-lgxdevice -lbaslercam -lgxpp -lgx -lpthread
```

[**Note:** When running an application linked dynamically against the `libgxdevice` library on a Linux PC, the runtime linker must know where to find the Basler eXcite library files. For example, use the `LD_LIBRARY_PATH` environment variable.]

Example:

```
export LD_LIBRARY_PATH=/opt/excite-tools/host/lib:$LD_LIBRARY_PATH
./StreamingClient
```

1.4.4 Samples

We recommend that you take a look at the two sample programs included with the eXcite that illustrate image transfer from eXcite to PC. These two sample programs, "StreamingServer" and "StreamingClient" are located in the samples directory on a CD included with the eXcite. Each of the programs consists of only one source file. These are:

- StreamingServer.cpp = the sample server program (to be compiled for running on the eXcite)
- StreamingClient.cpp = the sample client program (to be compiled for running on the PC)

1.4.5 Server Port Number is Configurable

As with many other examples of network communication, the GxClient/GxStreamServer communication makes use of a "port number" for the server. The purpose of the server port number is to allow multiple GxStreamServer-based servers to run simultaneously on the same eXcite.

For example, each of the eXcite devices in a network might run one image-sourcing server for images of type "A" and another image-sourcing server for images of type "B". Each type of server must have a different, distinct port number associated with it. In our example, for instance, the type "A" servers might have a port number of 3219 and the type "B" servers might have a 7212 port number.

Each request sent from client to server incorporates, as a part of the message, the port number of the type of server with which the client wishes to communicate. At the server side, each server "listens" and responds only to incoming client requests targeted to "its" port number.

The server port number that a GxStreamServer-based server listens to, and the server port number that a GxClient-based client program connects to, both have the same default value (which is fixed in the eXcite library).

In the case where there is only one GxStreamServer-based server program running on the eXcite, the application programmer can simply use this default server port number. The server and client will then find each other and communicate with each other.

The port number used can be set to something other than the default value as follows:

- To specify a non-default server port number to the GxStreamServer class, we pass this server port number as the last argument to the "GxStreamServer::Open()" method. This argument has as its default value the default server port number. The "StreamingServer/StreamingServer.cpp" sample illustrates this.
- To specify a non-default server port number to the GxClient class, the application programmer must use a "PropertySet" object passed to the "DeviceManager::EnumerateDevices()" call. Before passing the "PropertySet" to the "EnumerateDevices()" function, the "PropertySet" object is initialized to contain the port number information. The "StreamingClient/StreamingClient.cpp" sample illustrates this.

1.4.6 Application-level Protocol

Every type of communication needs a "communications protocol", i.e., a set of conventions about how the communication is structured that is adhered to by both of the communicating parties.

Everything in the lower-level layers of the communication between the GxStreamServer-based server and the GxClient-based client is hidden from the application programmer by the eXcite library. Only the highest level of the control and (time)sequencing of the communication remains visible to the application programmer. This highest level consists of the calls to the member functions of GxStreamServer and GxClient in the application code.

Inherent in the design of the GxStreamServer/GxClient classes is that, for image transmission to work properly, the server and client application programs must call the member functions of GxStreamServer and GxClient in a certain way, especially with regard to the order (in time) of the function calls. The way in which the client and server application programs, through calls to the member functions of GxStreamServer and GxClient, control the communication so that client and server program interact usefully is called the "application-level protocol".

First, we present a diagram that summarizes this application-level protocol. The diagram maps the flow, over time, of the communication process between server and client. (In the diagram, time increases as you move from top to bottom.)

After the diagram, we describe in more detail the application-level protocol that client and server program must follow. The description, like the diagram, simply tracks the flow of the communication process between server and client over time, as seen by the client and server application programs.

In the diagram starting on the next page, the arrows indicate the period of time during which a function call is executing.

Server

Initialize (= GxStreamServer::Open())

Wait for incoming client connection
and wait till client opens a
"streaming channel" in that connection
[The two waits are one single function call
= GxStreamServer::WaitForStreamOpen()]



Sees that client has opened streaming channel
(= the GxStreamServer::WaitForStreamOpen())

Sends image size information

Sends image width, height, depth

Waits till client sets streamingEnabled to "true"
(= GxStreamServer::WaitForStreamingEnabled())



Sees that client has set streamingEnabled to "true"
(= the GxStreamServer::WaitForStreamOpen()
call returns successfully)

Starts sending first image
= { GxStreamServer::QueueBuffer();
GxStreamServer::WaitForBuffer() }



Waits till send of first image complete
(= the GxStreamServer::WaitForBuffer())

Client

Open connection to server = GxClient::Open()

Open streaming channel = GxClient::PrepareGrab()



Get image size = GxClient::TotalBytes()



The GxClient::TotalBytes() call returns

Get other image dimensions (width, height, depth)



The GxClient::Width(), Height(), DataDepth()
calls return

Initialize user buffers

Enqueue all user buffers:
call GxClient::QueueBuffer() for each buffer

GxClient::EnableStreaming = true

Calls GxClient::WaitForBuffer() for retrieving
the buffer containing the 1st image



Waits till receive of 1st image done & retrieves
the buffer containing the received image
(= the GxClient::WaitForBuffer()
call returns successfully)

Re-queue the buffer for receiving further
image data
(= GxStreamServer::QueueBuffer();)

Checks if still streamingEnabled == true
(= GxStreamServer::IsStreamingEnabled())

YES, so ...

Starts sending 2nd image
= { GxStreamServer::QueueBuffer();
GxStreamServer::WaitForBuffer() }



Waits till send of 2nd image complete
(= the GxStreamServer::WaitForBuffer()
call returns successfully)

Calls GxClient::WaitForBuffer()
for retrieving the buffer containing
the 2nd image



Waits till receive of 2nd image done
and retrieves the buffer containing the
received image = GxClient::WaitForBuffer()

Re-queue the buffer for receiving further
image data = GxClient::QueueBuffer();

... REPEAT ...

... UNTIL :

Sets GxClient::EnableStreaming = false

Completes sending of Nth image
= GxStreamServer::WaitForBuffer()

Checks if still streamingEnabled == true
(= GxStreamServer::IsStreamingEnabled())

NO, so ...

Waits for client to close streaming channel
= GxStreamServer::WaitForStreamClose()



Sees that client has closed streaming channel
(= the WaitForStreamClose() call
returns successfully)

Closes streaming channel
(= GxClient::FinishGrab())

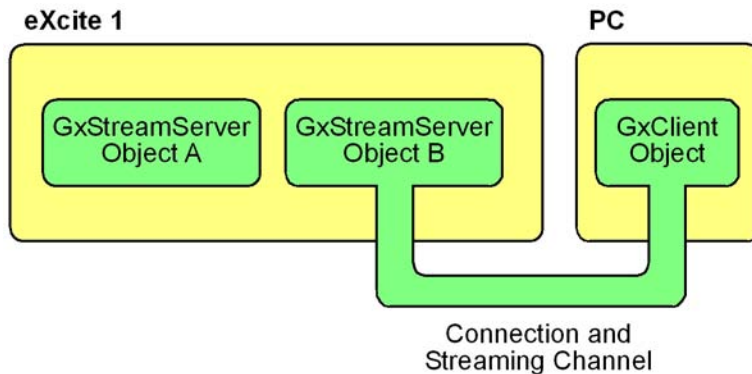
Since the server is the passive party and the client the active party, the server program must be "on line" first. Thus, the first act of the process is to start the server program. As soon as it is started, the server program instantiates a "GxStreamServer" object, calls its "Open()" function (which initializes the object), and then calls its "WaitForStreamOpen()" function. This last function call puts the GxStreamServer object in the state of listening for, and waiting for, a client that wants to establish a connection with the server.

The client establishes the connection by calling its "Open()" method.

After the "connection" between the GxStreamServer object on the eXcite and the GxClient object on the PC is established, the GxStreamServer and GxClient objects create an additional communication channel between them. This is called the "**streaming channel**". The actual image data is transferred from server to client via this streaming channel. The client opens the streaming channel by calling its "PrepareGrab()"

method.

Both the "connection" and the "streaming channel" are communication channels between two parties only: namely between (on one side) one specific GxStreamServer object on one specific eXcite, and (on the other side) one specific GxClient object on the PC.



The `GxStreamServer::WaitForStreamOpen()` function blocks communication until either a `GxClient` object has connected to the `GxStreamServer` and has opened the streaming channel, or until the specified time-out expires. When the `GxStreamServer::WaitForStreamOpen()` function returns successfully, the server program knows that a client has connected and has successfully established a streaming channel.

[**Note:** Every `GxStreamServer::Wait...()` call must be checked in the server program on its return value. If a return value indicating failure is returned on any of these calls, this means that the client has not properly followed the communication protocol, e.g., the client may have simply hung up the connection. Whenever a `GxStreamServer::Wait...()` call returns unsuccessfully, the server can break off the protocol and can terminate the connection. The server program can terminate the connection at any point, regardless of the state of the connection, by simply calling the "`GxStreamServer::Close()`" function.]

Apart from the image data itself, there is another kind of information that must be transferred from the server to the client, i.e., **information about the size of the images**.

With respect to image size, it is important to note that the `GxStreamServer`/`GxClient`-based communication is organized in such a way that:

- It is always the server, and not the client, that determines the dimensions (width, height, data depth per pixel) of the transferred images.
- All the images transferred from one "`GxStreamServer`" instance, during the whole of its lifetime, have the same dimensions (width, height, data depth per pixel). Note that as a consequence, all the images transferred via a streaming channel must have the same dimensions as long as that streaming channel is up and running,.

The image dimensions (width, height, data depth per pixel) are specified by the server program to the "`GxStreamServer`" object by means of arguments to the "`GxStreamServer`" constructor.

The image size information must be transferred from server to client first, before the actual images are transferred. This image size information is needed by the client program to instantiate buffers into which the received image data can be written.

To get the image size information from the `GxStreamServer` object into the client program, the client program uses the following read-only member objects of the "`GxClient`" object:

- Width = image width (in pixels)
- Height = image height (in pixels)
- DataDepth = data depth (in bytes) of each pixel
- TotalBytes = total size (in bytes) of one image

The API for these GxClient member objects is similar to that of the member objects of "Camera" as discussed in the [The Member Objects of Camera](#) section. All "GxClient" member objects are of the "Integer" data type. The "GetValue()" method (like its equivalent "operator()") of the GxClient member objects is implemented as the **action** of getting the relevant value from the server.

[**Note:** We recommend that the client application program uses the "TotalBytes" value to determine the size of the buffers instead of computing the buffer size from "Width*Height*DataDepth".]

As soon as the client program has allocated the buffers to be used to receive the data sent by the server, it "queues in" these buffers into the GxClient object by means of calls to the GxClient::QueueBuffer() function. This makes the addresses of the user buffers known to the GxClient object. The GxClient object will start receiving image data and writing the received data into the available user buffers as soon as possible.

At this point, we have the client in a state where it is trying to receive image data from the server and at this point, the server could in principle start sending image data. In addition to the GxClient member objects already described above, the GxClient class has one more member object named **EnableStreaming**. The purpose of this member object is to allow the client to tell the server to start and stop sending data.

On the server side, there are two functions that inspect the value of "EnableStreaming", namely:

bool GxStreamServer::isStreamingEnabled() - Returns the current value of StreamingEnabled.

GxStreamServer::WaitForStreamingEnabled() - Blocks (waits) until StreamingEnabled is "true".

Using these functions, the image-sending loop in the server can now be coded as:

```
if ( pServer->WaitForStreamingEnabled(TIMEOUT) )
    while ( pServer->isStreamingEnabled() )
    {
        send the next image frame
    }
```

Now, the server can be stopped by the client program, simply by the client program setting GxClient::EnableStream to "false".

The member functions of GxStreamServer and GxClient that carry out the transfer of images are "QueueBuffer()" and "WaitForBuffer()". Each of these functions is present in both the GxStreamServer and GxClient classes and each has a similar function in both classes.

The "QueueBuffer()" function "enqueues" a user buffer into the GxStreamServer or GxClient object. GxStreamServer::QueueBuffer() tells the server the address of the buffer to be sent. GxClient::QueueBuffer() passes the buffer used to receive the data to the GxClient object.

The "WaitForBuffer()" function completes the sending or receiving of one image. The function is blocked (wait) until the next image has been fully sent (for GxStreamServer) or received (for "GxClient").

For both the GxStreamServer and GxClient objects, whenever a "WaitForBuffer()" call returns successfully, it means that the GxStreamServer or GxClient object is done with the buffer. That is, the successful return from a WaitForBuffer() call "de-queues" one user buffer (removes one user buffer from the set of buffers known to and being operated on by the GxStreamServer or GxClient object). After a successful return from WaitForBuffer(), the application program can re-queue the buffer (by means of a new "QueueBuffer()" call).

1.4.7 The Server Program

To make all the necessary features from the API of the eXcite library visible to the server program, the application programmer need only include the following line at the head of the source file(s):

```
#include <gxdevice/GxStreamServer.h>
```

The server functionality provided by the eXcite library all sits in one class: GxStreamServer.

The "GxStreamServer" class does have the two "QueueBuffer()" and "WaitForBuffer()" member functions that are roughly similar to the member functions of the same name in the "Camera" and "GxClient" classes. The main difference is, of course, that in the "GxStreamServer" class, these functions *send* data, whereas the "Camera" and "GxClient" member functions *receive*. A more minor difference is that the API (the function prototype) of these two functions in "GxStreamServer" is similar, but not identical to, that of the corresponding functions of "Camera" and "GxClient".

A server program will normally instantiate just one instance of "GxStreamServer". This GxStreamServer instance incarnates the server process.

Overview of Member functions of GxStreamServer:

1) Methods to initialize and de-initialize GxStreamServer object:

- Open()
- Close()

Also closes any client connection that is still open

2) Methods for the detect-that-stream-opened part of the communication:

- WaitForStreamOpen()
- bool WaitForStreamClose()
- bool IsStreamOpen()

3) Methods for the StreamingEnable part of the communication:

- WaitForStreamingEnabled()
- bool IsStreamingEnabled()

4) methods for the send-individual-images part of the communication:

- QueueBuffer()
Passes a buffer containing one image to the GxStreamServer object, and starts sending the image data
- WaitForBuffer()
Completes sending the buffer and blocks until the send is complete)

For more info on each of these functions, see the [GxStreamServer](#) page in the API Reference.

For an example of all these functions in the context of a realistic full-server program, Please take a look at the "StreamingServer/StreamingServer.cpp" sample.

Note that all of the classes in the eXcite library may throw exceptions (see [Exceptions](#)), so that most application programs would want to put every use (or instantiation) of any of the classes from the eXcite library inside "try ... catch" constructs.

1.4.8 The Client Program

To make all of the necessary features from the API of the eXcite library visible to the client program, include the following lines at the head of the source file(s):

```
#include <device/DeviceManager.h>
#include <gxdevice/GxClient.h>
```

It is not necessary to set any `define`.

The purpose of the client program is to download (to "get") images. The eXcite library has made use of the parallel between this function and the grabbing of images done by the part of the eXcite library described in the [Controlling the eXcite from a Local Program](#) section. While in the [Controlling the eXcite from a Local Program](#) section the application program running on the eXcite is grabbing images from the camera section of the eXcite, the client program running on the PC is "grabbing" images from a remote eXcite device over a network.

The most important object in the client program is the "GxClient" object. The API for the GxClient object is similar to the "Camera" object described in the [Controlling the eXcite from a Local Program](#) section.

The Member Functions of GxClient

The GxClient object has the same set of member functions as the "Camera" object (the member functions of "Camera" were described in the [The Member Functions of Camera](#) section). In the case of "GxClient", the member functions execute the following actions:

1) Functions to open/close the connection to a GxStreamServer object:

- `pClient->Open();`
Open a connection to the GxStreamServer object
- `pClient->Close();`
Close a connection to the GxStreamServer object
- `bool myBool = pClient->IsOpen();` ;
Queries whether a connection to the server is open

2) Functions to open/close a "streaming channel" to GxStreamServer object:

- `pClient->PrepareGrab();`
Open a "streaming channel"
- `pClient->FinishGrab();`
Close a "streaming channel"
- `bool myBool = pClient->IsReadyForGrab();`
Query whether a "stream" is in an open state

3) Functions for the part of the application-level communication protocol in which individual images are transferred:

- `pClient->QueueBuffer( nBytes, pBuffer, pUser );`
Pass the API the address of a user buffer where the result image(s) (received from server) should be written

```
• IInDataStream::BufferStatus bs = pClient->WaitForBuffer( ppBuffer, ppUser,
  timeout_ms );
```

Download (receive) one image from the server and wait for this image download to complete

The EnableStreaming part of the application level protocol is done on the client side by means of the "Enable-Streaming" *member object* of "GxClient".

The Member Objects of GxClient

The GxClient and Camera objects are also similar in that, apart from member function, they each have a set of member objects. In both, these member objects constitute an important part of the API. The application program accesses the member objects of "GxClient" in the same way as the member objects of "Camera" are accessed. Therefore, the discussion in the [The Member Objects of Camera](#) section for the IInteger/IBoolean/IFloat/IString types of the member objects, and on how to use the set of methods for each of these types to access the member object, is also valid for the member objects of "GxClient".

The complete set of member objects of "GxClient" is:

- EnableStreaming
- Width = Width of frame in pixels
- Height = Height of frame in pixels
- DataDepth = Number of bytes per pixel
- TotalBytes = Number of bytes in one frame

EnableStreaming is a read-write parameter, and the other parameters are read-only.

"EnableStreaming" is provided as a facility for the client program to pass a command to the server to start or stop the sending of image data. Setting

```
pClient->EnableStreaming = true;
```

tells the server (that the GxClient object is connected to) that it should start sending image data.

Setting

```
pClient->EnableStreaming = false;
```

tells the server (that the GxClient object is connected to) that the server should stop sending image data. How "EnableStreaming" is intended to be used is discussed in some detail in the [Application-level Protocol](#) section above.

The purpose of the other member objects, as discussed in the [Application-level Protocol](#) section, is to transfer image size and image dimension data from server to client. Recall that the server is the party that determines the values of these quantities.

After the connection between GxClient and GxStreamServer is established, the client application program can query these member objects as to the image dimensions. This image dimension information is needed on the client side for the client to be able to interpret the received image data. In addition, the image size information is needed by the client program so that the client program can initialize buffer(s) into which to receive the image data.

The "TotalBytes" member object is provided specifically for the purpose of informing the client about the size of the buffer (in bytes) needed for one image. The client program should use

```
bufferSize = pClient->TotalBytes()
```

rather than calculating the buffer size from the values of Width, Height, and DataDepth.

For more information about how to use the member functions and member objects of the `GxClient` object to receive ("download") image data from an eXcite running a `GxStreamServer`-based server program, see the [GxClient](#) page in the API Reference.

The DeviceManager and Its Return List

The "GxClient" object is created by the "DeviceManager" in a similar fashion to the way the "Camera" object is created in the [Controlling the eXcite from a Local Program](#) section. The only difference is that in our present "GxClient" case, the parameter passed to `DeviceManager::EnumerateDevices()` is now "`GxClient::DeviceTypeId`", instead of "`Camera::DeviceTypeId`".

When passing "`GxClient::DeviceTypeId`" as the argument value, "`EnumerateDevices()`" will look for all `GxStreamServer` objects (= servers) that are up and running in the network.

As explained in the [Server Port Number is Configurable](#) subsection, the port number that the "`EnumerateDevices()`" functions uses to discover `GxStreamServer` instances can be specified by passing the port number in the form of a "PropertySet" object as an extra parameter into the "`EnumerateDevices()`" function. If no port number is specified, the default port number is used.

The `DeviceManager::EnumerateDevices()` returns a list containing all of the "devices" of the desired type (i.e., listening on the specified port) that are reachable. In the [Controlling the eXcite from a Local Program](#) use case, the list always contains exactly one element (namely, the local eXcite device). In the case of the enumeration of `GxStreamServer` instances in the network, the list returned by the `DeviceManager::EnumerateDevices()` call can contain zero, one, or more entries, depending on how many `GxStreamServer` objects are running on the eXcite devices connected to the network. Each entry in the list refers to exactly one of these "remote" `GxStreamServer` objects.

The `DeviceManager::EnumerateDevices()` method returns an iterator type

```
DeviceManager::DeviceInfoList_t::iterator
```

which is defined by the include files of the eXcite library as

```
std::list<DeviceInfo>::iterator
```

and can therefore be used in the familiar C++ STL `<list>` way. The application program (= client program) must now iterate through the list returned by `EnumerateDevices()`, in order to select the remote `GxStreamServer` object that it wants to contact.

The list element for the selected remote `GxStreamServer` object is then passed to the "`DeviceManager::CreateDevice()`" function. This function creates a "GxClient" object "bound" to that remote `GxStreamServer` object and returns a pointer to this newly created "GxClient" object.

Note carefully that the "GxClient" object is "bound" to one specific remote `GxStreamServer` object from the moment it was created. In other words, the "GxClient" is created in a form that is configured for connecting to one specific remote `GxStreamServer` object. Note, however, that the actual connection to that server object is not yet in existence. The remote `GxStreamServer` is connected when the `GxClient::Open()` function is called. The "GxClient" object created in this way is then further used in the client program as described in the earlier subsections.

For an example of all the `DeviceManager` functions and `GxClient` API features used in the context of a complete client program, we recommend that you look at the "StreamingClient/StreamingClient.cpp" sample code.

Note that all the classes in the eXcite library may throw exceptions (see [Exceptions](#)), so that most application programs would want to put every use (or instantiation) of any of the classes from the eXcite library inside "try ... catch" constructs.

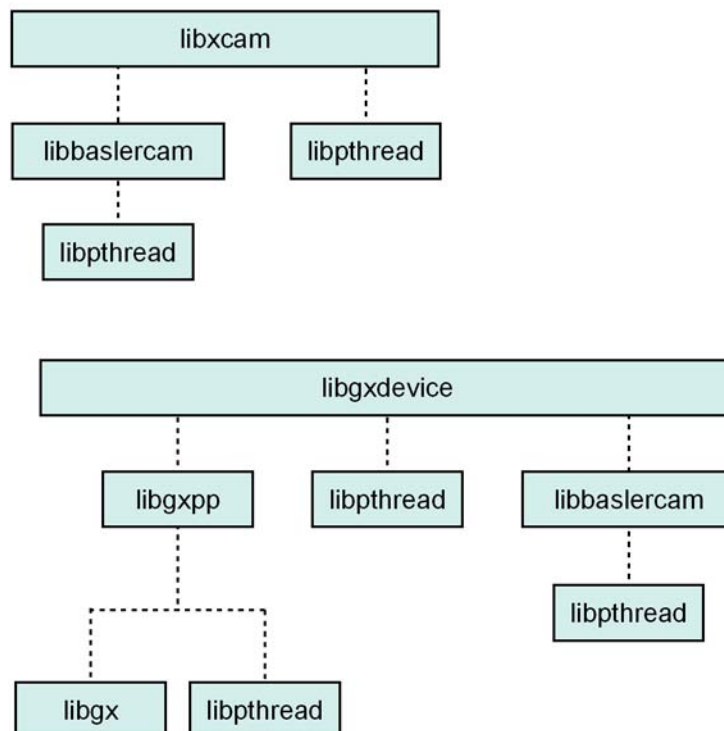
1.5 Library Files

The library files included with the eXcite library are:

Name of Library File	Contains
libbaslercam	General infrastructure: DeviceManager, GenApi
libxcam	Camera class, for the use case of controlling eXcite from a local program
libgxdevice	GxClient and GxStreamServer classes, for both the server and client programs for image transfer from an eXcite to a PC
libgx	Underlying library used by libgxdevice
libgxpp	Underlying library used by libgxdevice

Apart from these library files created by Basler, the application programmer also needs the general Linux "pthread" library "libpthread".

The dependencies between these library files are diagrammed below. An "A" item above a "B", item connected to it by a vertical line, means that "A" uses (depends on) "B".



Chapter 2

Module Documentation

2.1 Exceptions

Exceptions.

Classes

- class [BaslerCamera::DeviceIoException](#)
Exception class thrown to indicate device I/O related errors.
- class [GenApi::GenericException](#)
Base class for all exceptions thrown by the library.
- class [GenApi::InvalidArgumentException](#)
Exception fired when an argument is invalid.
- class [GenApi::OutOfRangeException](#)
Exception fired if an argument is out of range.
- class [GenApi::PropertyException](#)
Exception fired if a property access fails.
- class [GenApi::RuntimeException](#)
Runtime exception.
- class [GenApi::LogicalErrorException](#)
Exception thrown to indicate logical errors in the program flow.

2.1.1 Detailed Description

This group contains the Exception classes that can be thrown by the functions (including constructor functions) in the eXcite library.

The [Exceptions](#) section in the API Overview briefly explains how the eXcite library is organized with respect to Exceptions.

2.2 Device Manager

The DeviceManager class and related classes.

Classes

- class [BaslerCamera::DeviceInfo](#)
Stores information used to open a device.
- class [BaslerCamera::DeviceManager](#)
Enumerates, creates, and destroys devices.
- class [BaslerCamera::PropertySet](#)

2.2.1 Detailed Description

The DeviceManager class and its related classes are used to create the 'Camera' object through which an eXcite camera is operated from your application code.

The [Using the DeviceManager Object](#) section in the API Overview explains how to use these classes.

2.3 Camera

Components related to the 'Camera' class.

Modules

- [CamT](#)
CamT class template.
- [CXCamInterface](#)
The camera control interface.
- [Types for the data members of 'Camera'](#)
Types for the data members of 'Camera'.

Typedefs

- `typedef CamT< XCamDelegateT< XCamInterface::CXCamInterface >, XCamInterface::CXCamInterface > BaslerCamera::XCam`
The eXcite camera device.

2.3.1 Detailed Description

This module contains all components related to the 'Camera' class used by the application code.

2.4 CamT

CamT class template.

Classes

- class [BaslerCamera::CamT< TliDelegate, ApiImpl >](#)
Class template used to implement a camera device.

2.4.1 Detailed Description

The CamT class template implements the member functions of the 'Camera' class used by the application code.

2.5 CXCamInterface

The camera control interface.

Classes

- class [XCamInterface::CXCamInterface](#)
The device's control interface.

2.5.1 Detailed Description

Contains the member objects of the 'Camera' class used by the application code. These member objects form the "camera control interface" used to set and get camera parameters.

2.6 Types for the data members of 'Camera'

Types for the data members of 'Camera'.

Modules

- [Non-enum types for the data members of 'Camera'](#)

Non-enum types for the data members of 'Camera'.

- [Enum types for the data members of 'Camera'](#)

Enum types for the data members of 'Camera'.

2.6.1 Detailed Description

This group contains the data types for the member objects (or synonymously "data members") of the 'Camera' class used by the application code.

2.7 Non-enum types for the data members of 'Camera'

Non-enum types for the data members of 'Camera'.

Classes

- struct [GenApi::IBoolean](#)
Interface for Boolean properties.
- struct [GenApi::IFloat](#)
Interface for float properties.
- struct [GenApi::IInteger](#)
Interface for integer properties.
- struct [GenApi::IString](#)
Interface for string properties.

2.7.1 Detailed Description

This group contains the non-enum data types for the member objects of the 'Camera' class used by the application code.

2.8 Enum types for the data members of 'Camera'

Enum types for the data members of 'Camera'.

Classes

- class [XCamInterface::CEnumeration_ColorCodingEnums](#)
Enumeration class used for the ColorCoding parameter.
- class [XCamInterface::CEnumeration_TriggerModeEnums](#)
Enumeration class used for the TriggerMode parameter.
- class [XCamInterface::CEnumeration_TriggerPolarityEnums](#)
Enumeration class used for the TriggerPolarity parameter.
- class [XCamInterface::CEnumeration_TriggerSourceEnums](#)
Enumeration class used for the TriggerSource parameter.
- class [XCamInterface::CEnumeration_TestImageEnums](#)
Enumeration class used for the TestImage parameter.
- class [XCamInterface::CEnumeration_PioOut0SrcEnums](#)
Enumeration class used for the PioOut0Src parameter.
- class [XCamInterface::CEnumeration_PioOut0MonitorEnums](#)
Enumeration class used for the PioOut0Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut0SettingEnums](#)
Enumeration class used for the PioOut0Setting parameter.
- class [XCamInterface::CEnumeration_PioOut1SrcEnums](#)
Enumeration class used for the PioOut1Src parameter.
- class [XCamInterface::CEnumeration_PioOut1MonitorEnums](#)
Enumeration class used for the PioOut1Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut1SettingEnums](#)
Enumeration class used for the PioOut1Setting parameter.
- class [XCamInterface::CEnumeration_PioOut2SrcEnums](#)
Enumeration class used for the PioOut2Src parameter.
- class [XCamInterface::CEnumeration_PioOut2MonitorEnums](#)
Enumeration class used for the PioOut2Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut2SettingEnums](#)
Enumeration class used for the PioOut2Setting parameter.
- class [XCamInterface::CEnumeration_PioOut3SrcEnums](#)

Enumeration class used for the PioOut3Src parameter.

- class [XCamInterface::CEnumeration_PioOut3MonitorEnums](#)
Enumeration class used for the PioOut3Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut3SettingEnums](#)
Enumeration class used for the PioOut3Setting parameter.
- class [XCamInterface::CEnumeration_Strobe0PolarityEnums](#)
Enumeration class used for the Strobe0Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe1PolarityEnums](#)
Enumeration class used for the Strobe1Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe2PolarityEnums](#)
Enumeration class used for the Strobe2Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe3PolarityEnums](#)
Enumeration class used for the Strobe3Polarity parameter.
- class [XCamInterface::CEnumeration_VideoModeEnums](#)
Enumeration class used for the VideoMode parameter.

2.8.1 Detailed Description

This group contains the enum data types for the member objects of the 'Camera' class used by the application code.

2.9 Image transfer from an eXcite to a PC

Classes for image transfer from an eXcite to a PC.

Namespaces

- namespace [BaslerCamera::StreamServer](#)
Contains interfaces and classes used to implement a stream server.

Classes

- struct [BaslerCamera::StreamServer::IRegisterSet](#)
Interface to be implemented by classes implementing a register set.
- class [BaslerCamera::GxStreamServer](#)
A customizable stream server class that allows sending a stream consisting of fixed size image data frames.

Typedefs

- typedef `CamT< GxDelegateT< GxClientInterface::CGxClientInterface >, GxClientInterface::CGxClientInterface > BaslerCamera::GxClient`
A device class used to connect to a [BaslerCamera::GxStreamServer](#).

2.9.1 Detailed Description

This group contains the classes for image transfer from an eXcite to a PC for both client and server application programs.

2.9.2 Typedef Documentation

2.9.2.1 `typedef CamT<GxDelegateT<GxClientInterface::CGxClientInterface>, GxClientInterface::CGxClientInterface> BaslerCamera::GxClient`

Use a [GxClient](#) object to grab data from an application implementing a [BaslerCamera::GxStreamServer](#).

Chapter 3

Namespace Documentation

3.1 BaslerCamera Namespace Reference

The namespace containing most of the components of the BaslerCam library.

Classes

- class [CamT](#)
Class template used to implement a camera device.
- struct [InDataStream](#)
Interface to be implemented by devices grabbing frames from an (image) data stream.
- struct [OutDataStream](#)
Interface to be implemented by classes providing an (image) data stream.
- class [DeviceIoException](#)
Exception class thrown to indicate device I/O related errors.
- struct [IDevice](#)
Interface to be implemented by devices.
- class [DeviceInfo](#)
Stores information used to open a device.
- class [DeviceManager](#)
Enumerates, creates, and destroys devices.
- class [PropertySet](#)
- class [GxStreamServer](#)
A customizable stream server class that allows sending a stream consisting of fixed size image data frames.

Namespaces

- namespace [StreamServer](#)

Contains interfaces and classes used to implement a stream server.

Typedefs

- typedef [XCam Camera](#)

The Camera object used to parameterize the camera and to capture images.

- typedef [CamT](#) < XCamDelegateT < [XCamInterface::CXCamInterface](#) >, [XCamInterface::CXCamInterface](#) > [XCam](#)

The eXcite camera device.

- typedef [CamT](#) < GxDelegateT < [GxClientInterface::CGxClientInterface](#) >, [GxClientInterface::CGxClientInterface](#) > [GxClient](#)

A device class used to connect to a [BaslerCamera::GxStreamServer](#).

3.2 BaslerCamera::StreamServer Namespace Reference

Contains interfaces and classes used to implement a stream server.

Classes

- struct [IRegisterSet](#)

Interface to be implemented by classes implementing a register set.

3.3 GxClientInterface Namespace Reference

The namespace containing the device's control interface and related enumeration types.

Classes

- class [CGxClientInterface](#)
The device's control interface.

3.4 XCamInterface Namespace Reference

The namespace containing the device's control interface and related enumeration types.

Classes

- class [CEnumeration_ColorCodingEnums](#)
Enumeration class used for the ColorCoding parameter.
- class [CEnumeration_TriggerModeEnums](#)
Enumeration class used for the TriggerMode parameter.
- class [CEnumeration_TriggerPolarityEnums](#)
Enumeration class used for the TriggerPolarity parameter.
- class [CEnumeration_TriggerSourceEnums](#)
Enumeration class used for the TriggerSource parameter.
- class [CEnumeration_TestImageEnums](#)
Enumeration class used for the TestImage parameter.
- class [CEnumeration_PioOut0SrcEnums](#)
Enumeration class used for the PioOut0Src parameter.
- class [CEnumeration_PioOut0MonitorEnums](#)
Enumeration class used for the PioOut0Monitor parameter.
- class [CEnumeration_PioOut0SettingEnums](#)
Enumeration class used for the PioOut0Setting parameter.
- class [CEnumeration_PioOut1SrcEnums](#)
Enumeration class used for the PioOut1Src parameter.
- class [CEnumeration_PioOut1MonitorEnums](#)
Enumeration class used for the PioOut1Monitor parameter.
- class [CEnumeration_PioOut1SettingEnums](#)
Enumeration class used for the PioOut1Setting parameter.
- class [CEnumeration_PioOut2SrcEnums](#)
Enumeration class used for the PioOut2Src parameter.
- class [CEnumeration_PioOut2MonitorEnums](#)
Enumeration class used for the PioOut2Monitor parameter.
- class [CEnumeration_PioOut2SettingEnums](#)
Enumeration class used for the PioOut2Setting parameter.
- class [CEnumeration_PioOut3SrcEnums](#)

Enumeration class used for the PioOut3Src parameter.

- class [CEnumeration_PioOut3MonitorEnums](#)
Enumeration class used for the PioOut3Monitor parameter.
- class [CEnumeration_PioOut3SettingEnums](#)
Enumeration class used for the PioOut3Setting parameter.
- class [CEnumeration_Strobe0PolarityEnums](#)
Enumeration class used for the Strobe0Polarity parameter.
- class [CEnumeration_Strobe1PolarityEnums](#)
Enumeration class used for the Strobe1Polarity parameter.
- class [CEnumeration_Strobe2PolarityEnums](#)
Enumeration class used for the Strobe2Polarity parameter.
- class [CEnumeration_Strobe3PolarityEnums](#)
Enumeration class used for the Strobe3Polarity parameter.
- class [CEnumeration_VideoModeEnums](#)
Enumeration class used for the VideoMode parameter.
- class [CXCamInterface](#)
The device's control interface.

Chapter 4

Class Documentation

4.1 BaslerCamera::CamT< TliDelegate, ApilImpl > Class Template Reference

Class template used to implement a camera device.

Inherits BaslerCamera::DeviceImpl, ApilImpl, and [BaslerCamera::IInDataStream](#).

Public Types

- enum [BufferStatus](#) { [bsOk](#) = 0, [bsTimeOut](#) = 1, [bsCancelled](#) = 2, [bsError](#) = 3 }
result type of [WaitForBuffer\(\)](#)

Public Member Functions

Construction/Destruction

*Although the constructor is declared as public, you **must not** use the constructor to create a device.*

Always use the [BaslerCamera::DeviceManager](#) to create devices!

- **CamT** (TliDelegate *pDelegate, const [DeviceInfo](#) &deviceInfo)
- virtual ~**CamT** ()

implementation of BaslerCamera::IDevice

- virtual void [Open](#) ()
Open the device.
- virtual bool [IsOpen](#) () const
Check if a device has been opened.
- virtual void [Close](#) ()
Close the device.

implementation of BaslerCamera::IInDataStream

- virtual void [PrepareGrab](#) ()
Prepare data acquisition.
- virtual void [QueueBuffer](#) (size_t nBytes, void *pBuffer, void *pUser)
Pass a buffer to be filled with data.
- virtual void [FlushBuffers](#) ()
Cancel all pending buffers.
- virtual [IInDataStream::BufferStatus WaitForBuffer](#) (void **ppBuffer, void **ppUser, unsigned long timeout_ms)
Wait for the next buffer.
- virtual void [FinishGrab](#) ()
Finish image acquisition.
- virtual bool [IsReadyForGrab](#) () const
Are we prepared for grabbing?

Static Public Attributes

- static const DeviceInfo::DeviceTypeId_t [DeviceTypeId](#)
Identifier used by the [DeviceManager](#) to enumerate and create the device.

4.1.1 Detailed Description

`template<class TliDelegate, class ApiImpl> class BaslerCamera::CamT< TliDelegate, ApiImpl >`

The [CamT](#) class combines the camera control interface (i.e., setting of camera parameters) and the interface used for grabbing images ([IInDataStream](#)).

Template parameters:

- [TliDelegate](#): Class wrapping the transport layer interface
- [ApiImpl](#) Class providing the API for the camera control interface (typically generated from XML)

4.1.2 Member Enumeration Documentation

4.1.2.1 enum [BaslerCamera::IInDataStream::BufferStatus](#) [inherited]

Enumerator:

bsOk success

bsTimeOut timeout occurred

bsCancelled buffer has been cancelled by calling [IInDataStream::FlushBuffers](#)

bsError error occurred

4.1.3 Member Function Documentation

4.1.3.1 `template<class TliDelegate, class Apilmpl> virtual void BaslerCamera::CamT<TliDelegate, Apilmpl>::PrepareGrab () [virtual]`

Necessary resource allocation will be performed within this function. `PrepareGrab()` must be called before calling `QueueBuffer()`, `FlushBuffers()`, and `WaitForBuffer()`.

Implements `BaslerCamera::IInDataStream`.

4.1.3.2 `template<class TliDelegate, class Apilmpl> virtual void BaslerCamera::CamT<TliDelegate, Apilmpl>::QueueBuffer (size_t nBytes, void * pBuffer, void * pUser) [virtual]`

`QueueBuffer()` enqueues the buffer into the input queue. For each buffer enqueued into the input queue by calling `QueueBuffer()`, one call to `WaitForBuffer()` should be issued to get the processed buffer back from the output queue.

When grabbing data from a camera, `QueueBuffer()` doesn't trigger the camera device to send images. Use device dependent functions like `OneShot()` or `ContinuousShot()` to cause the camera to send data. Normally one or more buffers should be enqueued in the input queue before triggering the camera.

Parameters:

- ← **nBytes** size of the buffer in bytes
- ← **pBuffer** pointer to the data buffer
- ← **pUser** additional context information that will be passed back to `WaitForBuffer()` when the buffer is filled

Implements `BaslerCamera::IInDataStream`.

4.1.3.3 `template<class TliDelegate, class Apilmpl> virtual void BaslerCamera::CamT<TliDelegate, Apilmpl>::FlushBuffers () [virtual]`

All buffers enqueued with `QueueBuffer()` will be cancelled and enqueued to the output queue. The cancelled buffers can be retrieved from the output queue by calling `WaitForBuffer()`.

Implements `BaslerCamera::IInDataStream`.

4.1.3.4 `template<class TliDelegate, class Apilmpl> virtual IInDataStream::BufferStatus BaslerCamera::CamT<TliDelegate, Apilmpl>::WaitForBuffer (void ** ppBuffer, void ** ppUser, unsigned long timeout_ms) [virtual]`

`WaitForBuffer()` retrieves the next buffer from the output queue. If there is no buffer available, `WaitForBuffer()` blocks until the specified timeout expires or a buffer becomes available.

Parameters:

- ↔ **ppBuffer** stores the buffer containing the (image-)data
- ↔ **ppUser** stores the user provided context pointer
- ← **timeout_ms** Maximum period to wait for the buffer [in ms]

Implements `BaslerCamera::IInDataStream`.

4.1.3.5 `template<class TliDelegate, class Apimpl> virtual void BaslerCamera::CamT<TliDelegate, Apimpl>::FinishGrab () [virtual]`

All allocated resources will be freed. There shouldn't be any buffers enqueued when calling `FinishGrab()`. The implementation of `FinishGrab()` will cancel all pending buffers and removes them from the output queue.

Ensure, that all buffers enqueued per `QueueBuffer()` have been retrieved by calling `WaitForBuffer()` before `FinishGrab()` is called. Otherwise, `FinishGrab()` will remove buffers from the output queue. These buffers are not passed to the client.

Don't call the `WaitForBuffer()`, `QueueBuffer()`, or `FlushBuffers()` methods after calling `FinishGrab()`!

Implements `BaslerCamera::IInDataStream`.

4.2 XCamInterface::CEnumeration_ColorCodingEnums Class Reference

Enumeration class used for the ColorCoding parameter.

Inherits CEnumerationBase.

Public Types

- enum [ColorCodingEnums](#) {
[ColorCoding_Mono8](#) = 0, [ColorCoding_YUV8_4_1_1](#) = 1, [ColorCoding_YUV8_4_2_2](#) = 2, [ColorCoding_YUV8_4_4_4](#) = 3,
[ColorCoding_RGB8](#) = 4, [ColorCoding_Mono16](#) = 5, [ColorCoding_RGB16](#) = 6, [ColorCoding_SMono16](#) = 7,
[ColorCoding_SRGB16](#) = 8, [ColorCoding_Raw8](#) = 9, [ColorCoding_Raw16](#) = 10, [ColorCoding_VendorSpecific0](#) = 128,
[ColorCoding_VendorSpecific1](#) = 129, [ColorCoding_VendorSpecific2](#) = 130, [ColorCoding_VendorSpecific3](#) = 131, [ColorCoding_VendorSpecific4](#) = 132,
[ColorCoding_VendorSpecific5](#) = 133, [ColorCoding_VendorSpecific6](#) = 134 }
Valid values for ColorCoding.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.2.1 Member Enumeration Documentation

4.2.1.1 enum [XCamInterface::CEnumeration_ColorCodingEnums::ColorCodingEnums](#)

Enumerator:

- [ColorCoding_Mono8](#)** Y component has 8bit data .
- [ColorCoding_YUV8_4_1_1](#)** 4:1:1 YUV 8 format, each component has 8bit data .
- [ColorCoding_YUV8_4_2_2](#)** 4:2:2 YUV 8 format, each component has 8bit data .
- [ColorCoding_YUV8_4_4_4](#)** 4:4:4 YUV 8 format, each component has 8bit data .
- [ColorCoding_RGB8](#)** RGB 8 format, each component has 8bit data .
- [ColorCoding_Mono16](#)** Y component has 16bit unsigned data .
- [ColorCoding_RGB16](#)** RGB 16 format, each component has 16bit unsigned data .
- [ColorCoding_SMono16](#)** Y component has 16bit signed data .
- [ColorCoding_SRGB16](#)** RGB 16 format, each component has 16bit signed data .
- [ColorCoding_Raw8](#)** Raw data output of color filter sensor, 8bit data .
- [ColorCoding_Raw16](#)** Raw data output of color filter sensor, 16bit data .
- [ColorCoding_VendorSpecific0](#)** First of 128 vendor specific color codes .

ColorCoding_VendorSpecific1 VendorSpecific 1.
ColorCoding_VendorSpecific2 VendorSpecific 2.
ColorCoding_VendorSpecific3 VendorSpecific 3.
ColorCoding_VendorSpecific4 VendorSpecific 4.
ColorCoding_VendorSpecific5 VendorSpecific 5.
ColorCoding_VendorSpecific6 VendorSpecific 6.

4.3 XCamInterface::CEnumeration_PioOut0MonitorEnums Class Reference

Enumeration class used for the PioOut0Monitor parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut0MonitorEnums](#) { [PioOut0Monitor_Low](#) = 0, [PioOut0Monitor_High](#) = 1 }
- Valid values for PioOut0Monitor.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.3.1 Member Enumeration Documentation

4.3.1.1 enum [XCamInterface::CEnumeration_PioOut0MonitorEnums::PioOut0MonitorEnums](#)

Enumerator:

PioOut0Monitor_Low Low.

PioOut0Monitor_High High .

4.4 XCamInterface::CEnumeration_PioOut0SettingEnums Class Reference

Enumeration class used for the PioOut0Setting parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut0SettingEnums](#) { [PioOut0Setting_Low](#) = 0, [PioOut0Setting_High](#) = 1 }
Valid values for PioOut0Setting.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.4.1 Member Enumeration Documentation

4.4.1.1 enum [XCamInterface::CEnumeration_PioOut0SettingEnums::PioOut0SettingEnums](#)

Enumerator:

[PioOut0Setting_Low](#) Low.

[PioOut0Setting_High](#) High .

4.5 XCamInterface::CEnumeration_PioOut0SrcEnums Class Reference

Enumeration class used for the PioOut0Src parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut0SrcEnums](#) {
[PioOut0Src_IntegrationEnable](#) = 0, [PioOut0Src_ReadyforTrigger](#) = 1, [PioOut0Src_SerialTx](#) = 2, [PioOut0Src_UserSet](#) = 3,
[PioOut0Src_Strobe0](#) = 4 }
Valid values for PioOut0Src.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.5.1 Member Enumeration Documentation

4.5.1.1 enum [XCamInterface::CEnumeration_PioOut0SrcEnums::PioOut0SrcEnums](#)

Enumerator:

PioOut0Src_IntegrationEnable Integration Enable.

PioOut0Src_ReadyforTrigger Ready for Trigger .

PioOut0Src_SerialTx Serial Tx.

PioOut0Src_UserSet User Set .

PioOut0Src_Strobe0 Strobe0.

4.6 XCamInterface::CEnumeration_PioOut1MonitorEnums Class Reference

Enumeration class used for the PioOut1Monitor parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut1MonitorEnums](#) { [PioOut1Monitor_Low](#) = 0, [PioOut1Monitor_High](#) = 1 }
- Valid values for PioOut1Monitor.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.6.1 Member Enumeration Documentation

4.6.1.1 enum [XCamInterface::CEnumeration_PioOut1MonitorEnums::PioOut1MonitorEnums](#)

Enumerator:

PioOut1Monitor_Low Low.

PioOut1Monitor_High High .

4.7 XCamInterface::CEnumeration_PioOut1SettingEnums Class Reference

Enumeration class used for the PioOut1Setting parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut1SettingEnums](#) { [PioOut1Setting_Low](#) = 0, [PioOut1Setting_High](#) = 1 }
- Valid values for PioOut1Setting.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.7.1 Member Enumeration Documentation

4.7.1.1 enum [XCamInterface::CEnumeration_PioOut1SettingEnums::PioOut1SettingEnums](#)

Enumerator:

PioOut1Setting_Low Low.

PioOut1Setting_High High .

4.8 XCamInterface::CEnumeration_PioOut1SrcEnums Class Reference

Enumeration class used for the PioOut1Src parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut1SrcEnums](#) {
 [PioOut1Src_IntegrationEnable](#) = 0, [PioOut1Src_ReadyforTrigger](#) = 1, [PioOut1Src_SerialTx](#) = 2, [PioOut1Src_UserSet](#) = 3,
 [PioOut1Src_Strobe1](#) = 4 }
Valid values for PioOut1Src.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.8.1 Member Enumeration Documentation

4.8.1.1 enum [XCamInterface::CEnumeration_PioOut1SrcEnums::PioOut1SrcEnums](#)

Enumerator:

PioOut1Src_IntegrationEnable Integration Enable.

PioOut1Src_ReadyforTrigger Ready for Trigger .

PioOut1Src_SerialTx Serial Tx.

PioOut1Src_UserSet User Set .

PioOut1Src_Strobe1 Strobe1.

4.9 XCamInterface::CEnumeration_PioOut2MonitorEnums Class Reference

Enumeration class used for the PioOut2Monitor parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut2MonitorEnums](#) { [PioOut2Monitor_Low](#) = 0, [PioOut2Monitor_High](#) = 1 }
- Valid values for PioOut2Monitor.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.9.1 Member Enumeration Documentation

4.9.1.1 enum [XCamInterface::CEnumeration_PioOut2MonitorEnums::PioOut2MonitorEnums](#)

Enumerator:

PioOut2Monitor_Low Low.

PioOut2Monitor_High High .

4.10 XCamInterface::CEnumeration_PioOut2SettingEnums Class Reference

Enumeration class used for the PioOut2Setting parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut2SettingEnums](#) { [PioOut2Setting_Low](#) = 0, [PioOut2Setting_High](#) = 1 }
Valid values for PioOut2Setting.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.10.1 Member Enumeration Documentation

4.10.1.1 enum [XCamInterface::CEnumeration_PioOut2SettingEnums::PioOut2SettingEnums](#)

Enumerator:

PioOut2Setting_Low Low.

PioOut2Setting_High High .

4.11 XCamInterface::CEnumeration_PioOut2SrcEnums Class Reference

Enumeration class used for the PioOut2Src parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut2SrcEnums](#) {
[PioOut2Src_IntegrationEnable](#) = 0, [PioOut2Src_ReadyforTrigger](#) = 1, [PioOut2Src_SerialTx](#) = 2, [PioOut2Src_UserSet](#) = 3,
[PioOut2Src_Strobe2](#) = 4 }
Valid values for PioOut2Src.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.11.1 Member Enumeration Documentation

4.11.1.1 enum [XCamInterface::CEnumeration_PioOut2SrcEnums::PioOut2SrcEnums](#)

Enumerator:

PioOut2Src_IntegrationEnable Integration Enable.

PioOut2Src_ReadyforTrigger Ready for Trigger .

PioOut2Src_SerialTx Serial Tx.

PioOut2Src_UserSet User Set .

PioOut2Src_Strobe2 Strobe2.

4.12 XCamInterface::CEnumeration_PioOut3MonitorEnums Class Reference

Enumeration class used for the PioOut3Monitor parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut3MonitorEnums](#) { [PioOut3Monitor_Low](#) = 0, [PioOut3Monitor_High](#) = 1 }
- Valid values for PioOut3Monitor.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.12.1 Member Enumeration Documentation

4.12.1.1 enum [XCamInterface::CEnumeration_PioOut3MonitorEnums::PioOut3MonitorEnums](#)

Enumerator:

PioOut3Monitor_Low Low.

PioOut3Monitor_High High .

4.13 XCamInterface::CEnumeration_PioOut3SettingEnums Class Reference

Enumeration class used for the PioOut3Setting parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut3SettingEnums](#) { [PioOut3Setting_Low](#) = 0, [PioOut3Setting_High](#) = 1 }
- Valid values for PioOut3Setting.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.13.1 Member Enumeration Documentation

4.13.1.1 enum [XCamInterface::CEnumeration_PioOut3SettingEnums::PioOut3SettingEnums](#)

Enumerator:

PioOut3Setting_Low Low.

PioOut3Setting_High High .

4.14 XCamInterface::CEnumeration_PioOut3SrcEnums Class Reference

Enumeration class used for the PioOut3Src parameter.

Inherits CEnumerationBase.

Public Types

- enum [PioOut3SrcEnums](#) {
 [PioOut3Src_IntegrationEnable](#) = 0, [PioOut3Src_ReadyforTrigger](#) = 1, [PioOut3Src_SerialTx](#) = 2, [PioOut3Src_UserSet](#) = 3,
 [PioOut3Src_Strobe3](#) = 4 }
Valid values for PioOut3Src.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.14.1 Member Enumeration Documentation

4.14.1.1 enum [XCamInterface::CEnumeration_PioOut3SrcEnums::PioOut3SrcEnums](#)

Enumerator:

PioOut3Src_IntegrationEnable Integration Enable.

PioOut3Src_ReadyforTrigger Ready for Trigger .

PioOut3Src_SerialTx Serial Tx.

PioOut3Src_UserSet User Set .

PioOut3Src_Strobe3 Strobe3.

4.15 XCamInterface::CEnumeration_Strobe0PolarityEnums Class Reference

Enumeration class used for the Strobe0Polarity parameter.

Inherits CEnumerationBase.

Public Types

- enum [Strobe0PolarityEnums](#) { [Strobe0Polarity_LowActiveOutput](#) = 0, [Strobe0Polarity_HighActiveOutput](#) = 1 }

Valid values for Strobe0Polarity.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)

Get list of entry nodes.

4.15.1 Member Enumeration Documentation

4.15.1.1 enum [XCamInterface::CEnumeration_Strobe0PolarityEnums::Strobe0PolarityEnums](#)

Enumerator:

Strobe0Polarity_LowActiveOutput Low Active Output.

Strobe0Polarity_HighActiveOutput High Active Output.

4.16 XCamInterface::CEnumeration_Strobe1PolarityEnums Class Reference

Enumeration class used for the Strobe1Polarity parameter.

Inherits CEnumerationBase.

Public Types

- enum [Strobe1PolarityEnums](#) { [Strobe1Polarity_LowActiveOutput](#) = 0, [Strobe1Polarity_HighActiveOutput](#) = 1 }

Valid values for Strobe1Polarity.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)

Get list of entry nodes.

4.16.1 Member Enumeration Documentation

4.16.1.1 enum [XCamInterface::CEnumeration_Strobe1PolarityEnums::Strobe1PolarityEnums](#)

Enumerator:

Strobe1Polarity_LowActiveOutput Low Active Output.

Strobe1Polarity_HighActiveOutput High Active Output.

4.17 XCamInterface::CEnumeration_Strobe2PolarityEnums Class Reference

Enumeration class used for the Strobe2Polarity parameter.

Inherits CEnumerationBase.

Public Types

- enum [Strobe2PolarityEnums](#) { [Strobe2Polarity_LowActiveOutput](#) = 0, [Strobe2Polarity_HighActiveOutput](#) = 1 }

Valid values for Strobe2Polarity.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)

Get list of entry nodes.

4.17.1 Member Enumeration Documentation

4.17.1.1 enum [XCamInterface::CEnumeration_Strobe2PolarityEnums::Strobe2PolarityEnums](#)

Enumerator:

Strobe2Polarity_LowActiveOutput Low Active Output.

Strobe2Polarity_HighActiveOutput High Active Output.

4.18 XCamInterface::CEnumeration_Strobe3PolarityEnums Class Reference

Enumeration class used for the Strobe3Polarity parameter.

Inherits CEnumerationBase.

Public Types

- enum [Strobe3PolarityEnums](#) { [Strobe3Polarity_LowActiveOutput](#) = 0, [Strobe3Polarity_HighActiveOutput](#) = 1 }

Valid values for Strobe3Polarity.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)

Get list of entry nodes.

4.18.1 Member Enumeration Documentation

4.18.1.1 enum [XCamInterface::CEnumeration_Strobe3PolarityEnums::Strobe3PolarityEnums](#)

Enumerator:

Strobe3Polarity_LowActiveOutput Low Active Output.

Strobe3Polarity_HighActiveOutput High Active Output.

4.19 XCamInterface::CEnumeration_TestImageEnums Class Reference

Enumeration class used for the TestImage parameter.

Inherits CEnumerationBase.

Public Types

- enum [TestImageEnums](#) {
[TestImage_Disabled](#) = 0, [TestImage_TestImage1](#) = 1, [TestImage_TestImage2](#) = 2, [TestImage_TestImage3](#) = 3,
[TestImage_TestImage4](#) = 4, [TestImage_TestImage5](#) = 5, [TestImage_TestImage6](#) = 6, [TestImage_TestImage7](#) = 7 }
Valid values for TestImage.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.19.1 Member Enumeration Documentation

4.19.1.1 enum [XCamInterface::CEnumeration_TestImageEnums::TestImageEnums](#)

Enumerator:

- TestImage_Disabled*** Disabled.
- TestImage_TestImage1*** Test Image 1.
- TestImage_TestImage2*** Test Image 2.
- TestImage_TestImage3*** Test Image 3.
- TestImage_TestImage4*** Test Image 4.
- TestImage_TestImage5*** Test Image 5.
- TestImage_TestImage6*** Test Image 6.
- TestImage_TestImage7*** Test Image 7.

4.20 XCamInterface::CEnumeration_TriggerModeEnums Class Reference

Enumeration class used for the TriggerMode parameter.

Inherits CEnumerationBase.

Public Types

- enum [TriggerModeEnums](#) { [TriggerMode_TriggerMode0](#) = 0, [TriggerMode_TriggerMode1](#) = 1, [TriggerMode_TriggerMode2](#) = 2, [TriggerMode_TriggerMode3](#) = 3 }

Valid values for TriggerMode.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)

Get list of entry nodes.

4.20.1 Member Enumeration Documentation

4.20.1.1 enum [XCamInterface::CEnumeration_TriggerModeEnums::TriggerModeEnums](#)

Enumerator:

TriggerMode_TriggerMode0 Trigger Mode 0.

TriggerMode_TriggerMode1 Trigger Mode 1.

TriggerMode_TriggerMode2 Trigger Mode 2.

TriggerMode_TriggerMode3 Trigger Mode 3.

4.21 XCamInterface::CEnumeration_TriggerPolarityEnums Class Reference

Enumeration class used for the TriggerPolarity parameter.

Inherits CEnumerationBase.

Public Types

- enum [TriggerPolarityEnums](#) { [TriggerPolarity_LowActive](#) = 0, [TriggerPolarity_HighActive](#) = 1 }
- Valid values for TriggerPolarity.*

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
- Get list of entry nodes.*

4.21.1 Member Enumeration Documentation

4.21.1.1 enum [XCamInterface::CEnumeration_TriggerPolarityEnums::TriggerPolarityEnums](#)

Enumerator:

TriggerPolarity_LowActive Low Active.

TriggerPolarity_HighActive HighActive.

4.22 XCamInterface::CEnumeration_TriggerSourceEnums Class Reference

Enumeration class used for the TriggerSource parameter.

Inherits CEnumerationBase.

Public Types

- enum [TriggerSourceEnums](#) {
 [TriggerSource_ExTrigPort0](#) = 0, [TriggerSource_ExTrigPort1](#) = 1, [TriggerSource_ExTrigPort2](#) = 2,
 [TriggerSource_ExTrigPort3](#) = 3,
 [TriggerSource_SoftTrig](#) = 7 }
 Valid values for TriggerSource.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
 Get list of entry nodes.

4.22.1 Member Enumeration Documentation

4.22.1.1 enum [XCamInterface::CEnumeration_TriggerSourceEnums::TriggerSourceEnums](#)

Enumerator:

- TriggerSource_ExTrigPort0*** Physical input port 0.
- TriggerSource_ExTrigPort1*** Physical input port 1.
- TriggerSource_ExTrigPort2*** Physical input port 2.
- TriggerSource_ExTrigPort3*** Physical input port 3.
- TriggerSource_SoftTrig*** Software trigger.

4.23 XCamInterface::CEnumeration_VideoModeEnums Class Reference

Enumeration class used for the VideoMode parameter.

Inherits CEnumerationBase.

Public Types

- enum [VideoModeEnums](#) {
[VideoMode_VideoMode0](#) = 0, [VideoMode_VideoMode1](#) = 1, [VideoMode_VideoMode2](#) = 2, [VideoMode_VideoMode3](#) = 3,
[VideoMode_VideoMode4](#) = 4, [VideoMode_VideoMode5](#) = 5, [VideoMode_VideoMode6](#) = 6, [VideoMode_VideoMode7](#) = 7 }
Valid values for VideoMode.

Public Member Functions

- virtual void [GetEntries](#) (GenApi::NodeList_t &Entries)
Get list of entry nodes.

4.23.1 Member Enumeration Documentation

4.23.1.1 enum [XCamInterface::CEnumeration_VideoModeEnums::VideoModeEnums](#)

Enumerator:

- VideoMode_VideoMode0*** Video Mode 0.
- VideoMode_VideoMode1*** Video Mode 1.
- VideoMode_VideoMode2*** Video Mode 2.
- VideoMode_VideoMode3*** Video Mode 3.
- VideoMode_VideoMode4*** Video Mode 4.
- VideoMode_VideoMode5*** Video Mode 5.
- VideoMode_VideoMode6*** Video Mode 6.
- VideoMode_VideoMode7*** Video Mode 7.

4.24 GxClientInterface::CGxClientInterface Class Reference

The device's control interface.

Public Member Functions

- [CGxClientInterface](#) (GenApi::IPort *pCameraPortImpl=NULL, GenApi::IPort *pGrabberPortImpl=NULL)
Constructor.
- GenApi::INode * [GetFeature](#) (const char *name)
Gets a feature by name.

Public Attributes

- [GenApi::IBoolean](#) & [EnableStreaming](#)
Tells the server to start/stop sending of streaming data.
- [GenApi::Integer](#) & [TotalBytes](#)
Number of bytes per frame (RO).
- [GenApi::Integer](#) & [Width](#)
Width of the frame in pixels (RO).
- [GenApi::Integer](#) & [Height](#)
Height of the frame in pixels (RO).
- [GenApi::Integer](#) & [DataDepth](#)
Number of bytes per pixel (RO).

4.25 XCamInterface::CXCamInterface Class Reference

The device's control interface.

Public Member Functions

- [CXCamInterface](#) (GenApi::IPort *pCameraPortImpl=NULL, GenApi::IPort *pGrabberPortImpl=NULL)

Constructor.

- GenApi::INode * [GetFeature](#) (const char *name)

Gets a feature by name.

Public Attributes

- [GenApi::Integer](#) & [AdvFeature](#)

Locks the advanced features.

- [GenApi::Boolean](#) & [OneShot](#)

Used to grab a single image.

- [GenApi::Boolean](#) & [ContinuousShot](#)

Used to start continuous image grabbing.

- [GenApi::Integer](#) & [ErrorFlag1](#)

Error in AOI Position, AOI Size, ColorID.

- [GenApi::Integer](#) & [ErrorFlag2](#)

Error in BytesPerPacket.

- [GenApi::Boolean](#) & [ErrorShutter](#)

Shutter control error.

- [GenApi::Boolean](#) & [ErrorGain](#)

Gain control error.

- [GenApi::Boolean](#) & [ErrorBrightness](#)

Brightness control error.

- [GenApi::Boolean](#) & [ErrorTrigger](#)

Trigger control error.

- [GenApi::Boolean](#) & [ErrorWhiteBalance](#)

White balance control error.

- [GenApi::Float](#) & [FrameInterval](#)

Indicates the current frame period in seconds.

- [GenApi::Integer & XPosition](#)
Sets the x position for the area of interest.
- [GenApi::Integer & YPosition](#)
Sets the y position for the area of interest.
- [GenApi::Integer & Width](#)
Sets the width for the area of interest.
- [GenApi::Integer & Height](#)
Sets the height for the area of interest.
- [GenApi::Integer & Shutter](#)
Sets the exposure time.
- [GenApi::Integer & Brightness](#)
Sets the image brightness.
- [GenApi::Integer & Gain](#)
Sets the camera's gain.
- [GenApi::Integer & WhiteBalanceUB](#)
Sets the blue level on color cameras.
- [GenApi::Integer & WhiteBalanceVR](#)
Sets the red level on color cameras.
- [GenApi::Boolean & TriggerEnable](#)
Enables the trigger feature.
- [XCamInterface::CEnumeration_TriggerModeEnums & TriggerMode](#)
Sets the trigger mode.
- [XCamInterface::CEnumeration_TriggerPolarityEnums & TriggerPolarity](#)
Sets the trigger polarity.
- [XCamInterface::CEnumeration_TriggerSourceEnums & TriggerSource](#)
Selects an input port or a software signal as the source for triggering.
- [GenApi::Boolean & SoftwareTrigger](#)
Triggers an image grab with minimal latency.
- [GenApi::Boolean & TriggerFlag](#)
Indicates whether a trigger has occurred.
- [GenApi::Integer & TriggerCounter](#)
Current value of the trigger counter.
- [XCamInterface::CEnumeration_TestImageEnums & TestImage](#)
Enables the test image feature and selects a test image.

- [GenApi::IBoolean](#) & [ShutterTimeBaseEnable](#)
Enables the Shutter Base feature.
- [GenApi::IFloat](#) & [ShutterTimeBase](#)
Sets the time base for the Shutter feature in seconds.
- [GenApi::IString](#) & [ExtVerInfo](#)
Gets the version numbers for the camera's internal software.
- [GenApi::Integer](#) & [PioOutput](#)
Parallel Output Control.
- [GenApi::Integer](#) & [PioInput](#)
Reads the state of the four digital inputs.
- [XCamlInterface::CEnumeration_PioOut0SrcEnums](#) & [PioOut0Src](#)
Selects the source signal for physical output port 0.
- [XCamlInterface::CEnumeration_PioOut0MonitorEnums](#) & [PioOut0Monitor](#)
Reads the current state of physical output port 0.
- [GenApi::IBoolean](#) & [PioOut0Invert](#)
Sets the invert function on physical output port 0.
- [XCamlInterface::CEnumeration_PioOut0SettingEnums](#) & [PioOut0Setting](#)
Sets the state of physical output port 0 (if the source selection is set to user settable).
- [XCamlInterface::CEnumeration_PioOut1SrcEnums](#) & [PioOut1Src](#)
Selects the source signal for physical output port 1.
- [XCamlInterface::CEnumeration_PioOut1MonitorEnums](#) & [PioOut1Monitor](#)
Reads the current state of physical output port 1.
- [GenApi::IBoolean](#) & [PioOut1Invert](#)
Sets the invert function on physical output port 1.
- [XCamlInterface::CEnumeration_PioOut1SettingEnums](#) & [PioOut1Setting](#)
Sets the state of physical output port 1 (if the source selection is set to user settable).
- [XCamlInterface::CEnumeration_PioOut2SrcEnums](#) & [PioOut2Src](#)
Selects the source signal for physical output port 2.
- [XCamlInterface::CEnumeration_PioOut2MonitorEnums](#) & [PioOut2Monitor](#)
Reads the current state of physical output port 2.
- [GenApi::IBoolean](#) & [PioOut2Invert](#)
Sets the invert function on physical output port 2.
- [XCamlInterface::CEnumeration_PioOut2SettingEnums](#) & [PioOut2Setting](#)

Sets the state of physical output port 2 (if the source selection is set to user settable).

- [XCamlInterface::CEnumeration_PioOut3SrcEnums](#) & [PioOut3Src](#)
Selects the source signal for physical output port 3.
- [XCamlInterface::CEnumeration_PioOut3MonitorEnums](#) & [PioOut3Monitor](#)
Reads the current state of physical output port 3.
- [GenApi::IBoolean](#) & [PioOut3Invert](#)
Sets the invert function on physical output port 3.
- [XCamlInterface::CEnumeration_PioOut3SettingEnums](#) & [PioOut3Setting](#)
Sets the state of physical output port 3 (if the source selection is set to user settable).
- [GenApi::IBoolean](#) & [Strobe0Enable](#)
Enables the strobe 0 output signal.
- [XCamlInterface::CEnumeration_Strobe0PolarityEnums](#) & [Strobe0Polarity](#)
Sets the strobe 0 output signal polarity.
- [GenApi::Integer](#) & [Strobe0Delay](#)
Sets the strobe 0 output signal delay.
- [GenApi::Integer](#) & [Strobe0Duration](#)
Sets the strobe 0 output signal duration.
- [GenApi::IBoolean](#) & [Strobe1Enable](#)
Enables the strobe 1 output signal.
- [XCamlInterface::CEnumeration_Strobe1PolarityEnums](#) & [Strobe1Polarity](#)
Sets the strobe 1 output signal polarity.
- [GenApi::Integer](#) & [Strobe1Delay](#)
Sets the strobe 1 output signal delay.
- [GenApi::Integer](#) & [Strobe1Duration](#)
Sets the strobe 1 output signal duration.
- [GenApi::IBoolean](#) & [Strobe2Enable](#)
Enables the strobe 2 output signal.
- [XCamlInterface::CEnumeration_Strobe2PolarityEnums](#) & [Strobe2Polarity](#)
Sets the strobe 2 output signal polarity.
- [GenApi::Integer](#) & [Strobe2Delay](#)
Sets the strobe 2 output signal delay.
- [GenApi::Integer](#) & [Strobe2Duration](#)
Sets the strobe 2 output signal duration.

- [GenApi::IBoolean](#) & [Strobe3Enable](#)
Enables the strobe 3 output signal.
- [XCamlInterface::CEnumeration_Strobe3PolarityEnums](#) & [Strobe3Polarity](#)
Sets the strobe 3 output signal polarity.
- [GenApi::Integer](#) & [Strobe3Delay](#)
Sets the strobe 3 output signal delay.
- [GenApi::Integer](#) & [Strobe3Duration](#)
Sets the strobe 3 output signal duration.
- [GenApi::Integer](#) & [StrobeDurationTimeBase](#)
Sets the duration time base for the strobe output signal feature.
- [GenApi::IFloat](#) & [Strobe0Duration_s](#)
Indicates the duration for strobe 0 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe1Duration_s](#)
Indicates the duration for strobe 1 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe2Duration_s](#)
Indicates the duration for strobe 2 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe3Duration_s](#)
Indicates the duration for strobe 3 output signal in seconds.
- [GenApi::Integer](#) & [StrobeDelayTimeBase](#)
Sets the delay time base for the strobe output signal feature.
- [GenApi::IFloat](#) & [Strobe0Delay_s](#)
Indicates the delay for strobe 0 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe1Delay_s](#)
Indicates the delay for strobe 1 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe2Delay_s](#)
Indicates the delay for strobe 2 output signal in seconds.
- [GenApi::IFloat](#) & [Strobe3Delay_s](#)
Indicates the delay for strobe 3 output signal in seconds.
- [XCamlInterface::CEnumeration_VideoModeEnums](#) & [VideoMode](#)
Video Modes.
- [XCamlInterface::CEnumeration_ColorCodingEnums](#) & [ColorCoding](#)
Controls the color mode.
- [GenApi::Integer](#) & [Bandwidth](#)

Sets the data rate (in bytes per packet) for transferring captured images from the camera section to the processor section.

- [GenApi::Integer](#) & [DataDepth](#)

Indicates the effective data depth.

- [GenApi::Integer](#) & [PixelNumber](#)

Indicates the number of pixels per frame in the captured images.

- [GenApi::Integer](#) & [PacketNumber](#)

Indicates the number of packets per frame for the captured images.

- [GenApi::Integer](#) & [TotalBytes](#)

Indicates the number of bytes per frame in the captured images.

- [GenApi::Boolean](#) & [SaveMemory](#)

Used to save current status.

- [GenApi::Integer](#) & [SaveMemoryCh](#)

Save settings to this channel.

- [GenApi::Integer](#) & [StartupMemoryCh](#)

Used to set the startup memory channel.

- [GenApi::Integer](#) & [CpuTemperature](#)

Indicates the current CPU temperature in degrees C.

- [GenApi::Integer](#) & [BoardTemperature](#)

Indicates the current board temperature in degrees C.

4.26 BaslerCamera::DeviceInfo Class Reference

Stores information used to open a device.

Inherits [BaslerCamera::PropertySet](#).

Public Member Functions

- [DeviceInfo](#) (const std::string &fullDeviceName, const std::string &friendlyDeviceName, const DeviceTypeld_t &deviceTypeld)
Constructor.
- std::string [GetFullDeviceName](#) () const throw ()
Returns the full device name.
- std::string [GetFriendlyDeviceName](#) () const throw ()
Returns the friendly device name.
- DeviceTypeld_t [GetDeviceTypeld](#) () const throw ()
Returns the device type ID.
- std::string [GetProperty](#) (const std::string &key) const
Returns a property.
- void [AddProperty](#) (const std::string &key, const std::string &value)
Adds a property.
- bool [HasProperty](#) (const std::string &key) const
Checks if the container contains a property.

4.26.1 Detailed Description

A device info object must contain all necessary information used to open a device. We are assuming that a device info object at least contains a friendly device name (e.g., to be displayed in device selection combo boxes) and a full device name (e.g., used by the OS to identify the device). Friendly and full device names don't need to be different.

4.26.2 Member Function Documentation

4.26.2.1 std::string BaslerCamera::PropertySet::GetProperty (const std::string & key) const [inherited]

Parameters:

key The name of the property

Returns:

the value of the property If the property set doesn't contain a value for the specified key, a Basler-Camera::RuntimeException will be thrown.

4.26.2.2 void BaslerCamera::PropertySet::AddProperty (const std::string & *key*, const std::string & *value*) [inherited]

Parameters:

key Name of the property

value The value of the property

4.26.2.3 bool BaslerCamera::PropertySet::HasProperty (const std::string & *key*) const
[inherited]

Parameters:

key The name of the property to check

Returns:

true when the property exists, false otherwise.

4.27 BaslerCamera::DeviceIoException Class Reference

Exception class thrown to indicate device I/O related errors.

Inherits [GenApi::GenericException](#).

Public Member Functions

- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std:exception).

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.28 BaslerCamera::DeviceManager Class Reference

Enumerates, creates, and destroys devices.

Public Types

- typedef std::list< [DeviceInfo](#) > [DeviceInfoList_t](#)
list of device info objects

Public Member Functions

- [IDevice](#) * [CreateDevice](#) (const [DeviceInfo](#) &deviceInfo)
Create a device.
- void [DestroyDevice](#) ([IDevice](#) *pDevice)
Destroy a device.
- [DeviceInfoList_t](#) [EnumerateDevices](#) (const [DeviceInfo::DeviceTypeId_t](#) &id, const [PropertySet](#) *p-EnumerationProps=NULL)
Enumerate devices.

Static Public Member Functions

- static [DeviceManager](#) & [GetInstance](#) ()
Returns the one and only Device Manager (singleton pattern).

4.28.1 Member Function Documentation

4.28.1.1 [IDevice](#)* [BaslerCamera::DeviceManager::CreateDevice](#) (const [DeviceInfo](#) & *deviceInfo*)

Parameters:

← ***deviceInfo*** device info object carrying all information needed for device creation

4.28.1.2 void [BaslerCamera::DeviceManager::DestroyDevice](#) ([IDevice](#) * *pDevice*)

Parameters:

← ***pDevice*** pointer to the device

4.28.1.3 [DeviceInfoList_t](#) [BaslerCamera::DeviceManager::EnumerateDevices](#) (const [DeviceInfo::DeviceTypeId_t](#) & *id*, const [PropertySet](#) * *pEnumerationProps* = NULL)

Devices of a given type will be enumerated. Each device type is associated with a unique device type identifier.

Parameters:

- ← ***id*** device type identifier
- ← ***pEnumerationProps*** (optional) For some device types, additional parameters can be specified for the enumeration process. These parameters are given in the form of a pointer to [PropertySet](#).

4.29 GenApi::GenericException Class Reference

Base class for all exceptions thrown by the library.

Inherits exception.

Inherited by [BaslerCamera::DeviceIoException](#), [GenApi::InvalidArgumentException](#), [GenApi::LogicalErrorException](#), [GenApi::OutOfRangeException](#), [GenApi::PropertyException](#), and [GenApi::RuntimeException](#).

Public Member Functions

- [GenericException](#) (const char *description, const char *sourceFileName, unsigned int sourceLine)
Constructor.
- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std:exception).
- virtual ~[GenericException](#) () throw ()
Destructor.

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.30 BaslerCamera::GxStreamServer Class Reference

A customizable stream server class that allows sending a stream consisting of fixed size image data frames. Inherits BaslerCamera::StreamServer::RegisterSet< Impl >, and BaslerCamera::GxStreamServerBase.

Public Types

- typedef IRegisterSet::Result(Impl::* [ReadHandler_t](#))(uint32_t *pData)
ReadHandler_t: The type of handler functions called when registers are to be read.
- typedef IRegisterSet::Result(Impl::* [WriteHandler_t](#))(uint32_t Data)
WriteHandler_t: The type of handler functions called when registers are to be written.
- typedef uint32_t [Address_t](#)
Type used for addresses.
- enum [Result](#)
Result type of IRegisterSet's methods.
- enum [BufferStatus](#) { [bsOk](#) = 0, [bsStreamClosed](#) = 1 , [bsError](#) = 3 }
result type of WaitForBuffer()

Public Member Functions

- virtual unsigned long [GetTotalBytes](#) () const
Returns the number of bytes a frame must contain.
- [GxStreamServer](#) (unsigned long width, unsigned long height, unsigned long datadepth, [StreamServer::IRegisterSet](#) *pCustomRegisterSet=NULL)
Create a stream server.
- void [Register](#) ([Address_t](#) addr, [ReadHandler_t](#) rh, [WriteHandler_t](#) wh)
Register handler functions for a specific register.
- virtual [Result](#) [DoReadRegister](#) ([Address_t](#) address, uint32_t *pData)=0
Called when a register is to be read.
- virtual [Result](#) [DoWriteRegister](#) ([Address_t](#) address, uint32_t Data)=0
- void [Open](#) (const std::string &serverName, unsigned long portNr=s_defaultPort)
Creates the server.
- void [Close](#) ()
Terminates the server.
- virtual bool [IsStreamOpen](#) ()
Has the client opened the stream?

- virtual bool [WaitForStreamOpen](#) (unsigned long timeout_ms)
Wait until the client has opened the stream.
- virtual bool [IsStreamingEnabled](#) ()
Has the client enabled streaming?
- virtual bool [WaitForStreamingEnabled](#) (unsigned long timeout_ms)
Wait until the client has enabled streaming.
- virtual bool [WaitForStreamClose](#) (unsigned long timeout_ms)
Wait until the client has closed streaming.

Implementation of BaslerCamera::IOutDataStream

Implementation of BaslerCamera::IOutDataStream

4.30.1 Detailed Description

A stream server accepts connections from GxClient devices via (Gigabit-)Ethernet and sends (image-)data to a connected client. A stream server can also implement a register set.

The client establishes the connection to the stream server by calling the [GxClient::Open\(\)](#) method. When the connection has been established, the Client can access the registers implemented by the stream server.

The client must open the data stream by calling the [GxClient::PrepareGrab\(\)](#) function. When the data stream has been opened, the client can provide data buffers by calling the [GxClient::QueueBuffer\(\)](#) method. When the buffers are queued in, streaming must be enabled. The server implements a StreamingEnable register, which can be accessed by the GxClient::EnableStreaming member object.

The server should not send any data before streaming has been enabled.

The client closes the stream by calling the [GxClient::FinishGrab\(\)](#) method.

4.30.2 Member Typedef Documentation

4.30.2.1 `template<class Impl> typedef IRegisterSet::Result( Impl::* Basler-Camera::StreamServer::RegisterSet< Impl >::ReadHandler_t)(uint32_t *pData)`
[inherited]

It's a pointer to a member function of the Impl class

4.30.2.2 `template<class Impl> typedef IRegisterSet::Result( Impl::* Basler-Camera::StreamServer::RegisterSet< Impl >::WriteHandler_t)(uint32_t Data)`
[inherited]

It's a pointer to a member function of the Impl class

4.30.3 Member Enumeration Documentation

4.30.3.1 enum [BaslerCamera::IOutDataStream::BufferStatus](#) [inherited]

Enumerator:

- bsOk*** success
- bsStreamClosed*** stream has been closed by the client
- bsError*** buffer has been cancelled

4.30.4 Constructor & Destructor Documentation

4.30.4.1 [BaslerCamera::GxStreamServer::GxStreamServer](#) (unsigned long *width*, unsigned long *height*, unsigned long *datadepth*, [StreamServer::IRegisterSet](#) * *pCustomRegisterSet* = NULL)

Parameters:

- ← ***width*** The width of one image frame [pixels]
- ← ***height*** The height of one image frame [pixels]
- ← ***datadepth*** Number of bytes per pixel
- ← ***pCustomRegisterSet*** An "additional" register set can be passed to the stream server. The stream server dispatches register read and write requests to the pCustomRegisterSet when the server itself isn't implementing the requested registers.

4.30.5 Member Function Documentation

4.30.5.1 virtual [Result](#) [BaslerCamera::StreamServer::IRegisterSet::DoReadRegister](#) ([Address_t](#) *address*, uint32_t * *pData*) [pure virtual, inherited]

Parameters:

- ← ***address*** Address to read from
- ***pData*** The register content will be written to this pointer

Returns:

IRegisterSet::rrOk on success

4.30.5.2 virtual [Result](#) [BaslerCamera::StreamServer::IRegisterSet::DoWriteRegister](#) ([Address_t](#) *address*, uint32_t *Data*) [pure virtual, inherited]

Parameters:

- ← ***address*** Address to write to
- ← ***Data*** Data to be written

Returns:

IRegisterSet::rrOk on success

4.30.5.3 `void BaslerCamera::GxStreamServerBase::Open (const std::string & serverName, unsigned long portNr = s_defaultPort) [inherited]`

Parameters:

- ← ***serverName*** The server's name
- ← ***portNr*** The number of the port to which the server is bound

4.30.5.4 `virtual bool BaslerCamera::GxStreamServerBase::WaitForStreamOpen (unsigned long timeout_ms) [virtual, inherited]`

The client opens the stream by calling the [GxClient::PrepareGrab\(\)](#) function.

Parameters:

- ← ***timeout_ms*** Specify a timeout in ms. When the timeout expires, the function returns false.

Returns:

true when a client has opened the stream within the specified period, false otherwise

4.30.5.5 `virtual bool BaslerCamera::GxStreamServerBase::WaitForStreamingEnabled (unsigned long timeout_ms) [virtual, inherited]`

The server should not send any data before the client has enabled streaming. The server should implement a register where the client can write to enable streaming.

Parameters:

- ← ***timeout_ms*** Specify a timeout in ms. When the timeout expires, the function returns false

Returns:

true when a client has opened the stream within the specified period, false otherwise

4.30.5.6 `virtual bool BaslerCamera::GxStreamServerBase::WaitForStreamClose (unsigned long timeout_ms) [virtual, inherited]`

The client closes the stream by calling [GxClient::FinishGrab\(\)](#)

4.31 GenApi::IBoolean Struct Reference

Interface for Boolean properties.

Inherits VirtualDestructable.

Public Member Functions

- virtual void [SetValue](#) (bool Value)=0
Set node value.
- virtual void [operator=](#) (bool Value)
Set node value.
- virtual bool [GetValue](#) () const =0
Get node value.
- virtual bool [operator\(\)](#) () const
Get node value.

4.32 BaslerCamera::IDevice Struct Reference

Interface to be implemented by devices.

Inherited by BaslerCamera::DeviceImpl.

Public Member Functions

- virtual [~IDevice](#) ()
Ensure that destructors are virtual.
- virtual void [Open](#) ()=0
Open the device.
- virtual bool [IsOpen](#) () const =0
Check if a device has been opened.
- virtual void [Close](#) ()=0
Close the device.
- virtual const [DeviceInfo](#) & [GetDeviceInfo](#) () const =0 throw ()
Retrieve the [DeviceInfo](#) object used to create the device.

4.32.1 Member Function Documentation

4.32.1.1 virtual const [DeviceInfo](#)& BaslerCamera::IDevice::GetDeviceInfo () const throw () [pure virtual]

For example, to ask the device for its friendly device name, get its [DeviceInfo](#) object and query the [DeviceInfo](#) object for the friendly device name.

4.33 GenApi::IFloat Struct Reference

Interface for float properties.

Inherits VirtualDestructable.

Public Member Functions

- virtual void [SetValue](#) (double Value)=0
Set node value.
- virtual [IFloat](#) & [operator=](#) (double Value)=0
Set node value.
- virtual double [GetValue](#) ()=0
Get node value.
- virtual double [operator\(\)](#) ()=0
Get node value.
- virtual double [GetMin](#) ()=0
Get minimum value allowed.
- virtual double [GetMax](#) ()=0
Get maximum value allowed.
- virtual ERepresentation [GetRepresentation](#) ()=0
Get recommended representation.

4.34 BaslerCamera::IInDataStream Struct Reference

Interface to be implemented by devices grabbing frames from an (image) data stream.

Inherited by [BaslerCamera::CamT< TliDelegate, Apilmpl >](#).

Public Types

- enum [BufferStatus](#) { [bsOk](#) = 0, [bsTimeOut](#) = 1, [bsCancelled](#) = 2, [bsError](#) = 3 }
result type of [WaitForBuffer\(\)](#)

Public Member Functions

- virtual void [PrepareGrab](#) ()=0
Prepare data acquisition.
- virtual bool [IsReadyForGrab](#) () const =0
Are we prepared for grabbing?
- virtual void [QueueBuffer](#) (size_t nBytes, void *pBuffer, void *pUser)=0
Pass a buffer to be filled with data.
- virtual void [FlushBuffers](#) ()=0
Cancel all pending buffers.
- virtual [BufferStatus](#) [WaitForBuffer](#) (void **ppBuffer, void **ppUser, unsigned long timeout_ms)=0
Wait for the next buffer.
- virtual void [FinishGrab](#) ()=0
Finish image acquisition.

4.34.1 Member Enumeration Documentation

4.34.1.1 enum [BaslerCamera::IInDataStream::BufferStatus](#)

Enumerator:

- bsOk*** success
- bsTimeOut*** timeout occurred
- bsCancelled*** buffer has been cancelled by calling [IInDataStream::FlushBuffers](#)
- bsError*** error occurred

4.34.2 Member Function Documentation

4.34.2.1 virtual void [BaslerCamera::IInDataStream::PrepareGrab](#) () [pure virtual]

Necessary resource allocation will be performed within this function. [PrepareGrab\(\)](#) must be called before calling [QueueBuffer\(\)](#), [FlushBuffers\(\)](#), and [WaitForBuffer\(\)](#).

Implemented in [BaslerCamera::CamT< TliDelegate, ApIImpl >](#).

4.34.2.2 virtual void BaslerCamera::InDataStream::QueueBuffer (size_t *nBytes*, void * *pBuffer*, void * *pUser*) [pure virtual]

[QueueBuffer\(\)](#) enqueues the buffer into the input queue. For each buffer enqueued into the input queue by calling [QueueBuffer\(\)](#), one call to [WaitForBuffer\(\)](#) should be issued to get the processed buffer back from the output queue.

When grabbing data from a camera, [QueueBuffer\(\)](#) doesn't trigger the camera device to send images. Use device dependent functions like [OneShot\(\)](#) or [ContinuousShot\(\)](#) to cause the camera to send data. Normally one or more buffers should be enqueued in the input queue before triggering the camera.

Parameters:

- ← ***nBytes*** size of the buffer in bytes
- ← ***pBuffer*** pointer to the data buffer
- ← ***pUser*** additional context information that will be passed back to [WaitForBuffer\(\)](#) when the buffer is filled

Implemented in [BaslerCamera::CamT< TliDelegate, ApIImpl >](#).

4.34.2.3 virtual void BaslerCamera::InDataStream::FlushBuffers () [pure virtual]

All buffers enqueued with [QueueBuffer\(\)](#) will be cancelled and enqueued to the output queue. The cancelled buffers can be retrieved from the output queue by calling [WaitForBuffer\(\)](#).

Implemented in [BaslerCamera::CamT< TliDelegate, ApIImpl >](#).

4.34.2.4 virtual BufferStatus BaslerCamera::InDataStream::WaitForBuffer (void ** *ppBuffer*, void ** *ppUser*, unsigned long *timeout_ms*) [pure virtual]

[WaitForBuffer\(\)](#) retrieves the next buffer from the output queue. If there is no buffer available, [WaitForBuffer\(\)](#) blocks until the specified timeout expires or a buffer becomes available.

Parameters:

- ↔ ***ppBuffer*** stores the buffer containing the (image-)data
- ↔ ***ppUser*** stores the user provided context pointer
- ← ***timeout_ms*** Maximum period to wait for the buffer [in ms]

Implemented in [BaslerCamera::CamT< TliDelegate, ApIImpl >](#).

4.34.2.5 virtual void BaslerCamera::InDataStream::FinishGrab () [pure virtual]

All allocated resources will be freed. There shouldn't be any buffers enqueued when calling [FinishGrab\(\)](#). The implementation of [FinishGrab\(\)](#) will cancel all pending buffers and removes them from the output queue.

Ensure, that all buffers enqueued per [QueueBuffer\(\)](#) have been retrieved by calling [WaitForBuffer\(\)](#) before [FinishGrab\(\)](#) is called. Otherwise, [FinishGrab\(\)](#) will remove buffers from the output queue. These buffers are not passed to the client.

Don't call the [WaitForBuffer\(\)](#), [QueueBuffer\(\)](#), or [FlushBuffers\(\)](#) methods after calling [FinishGrab\(\)](#)!

Implemented in [BaslerCamera::CamT< TliDelegate, ApIImpl >](#).

4.35 GenApi::Integer Struct Reference

Interface for integer properties.

Inherits VirtualDestructable.

Public Member Functions

- virtual void [SetValue](#) (int64_t Value)=0
Set node value.
- virtual [Integer](#) & [operator=](#) (int64_t Value)=0
Set node value.
- virtual int64_t [GetValue](#) ()=0
Get node value.
- virtual int64_t [operator\(\)](#) ()=0
Get node value.
- virtual int64_t [GetMin](#) ()=0
Get minimum value allowed.
- virtual int64_t [GetMax](#) ()=0
Get maximum value allowed.
- virtual int64_t [GetInc](#) ()=0
Get increment.
- virtual ERepresentation [GetRepresentation](#) ()=0
Get recommended representation.

4.36 GenApi::InvalidArgumentException Class Reference

Exception fired when an argument is invalid.

Inherits [GenApi::GenericException](#).

Public Member Functions

- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std:exception).

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.37 BaslerCamera::IOutDataStream Struct Reference

Interface to be implemented by classes providing an (image) data stream.

Inherited by BaslerCamera::GxStreamServerBase.

Public Types

- enum [BufferStatus](#) { [bsOk](#) = 0, [bsStreamClosed](#) = 1 , [bsError](#) = 3 }
result type of [WaitForBuffer\(\)](#)

Public Member Functions

- virtual void [QueueBuffer](#) (size_t nBytes, void *pBuffer, void *pUser)=0
Initiate the transmission of a buffer.
- virtual void [FlushBuffers](#) ()=0
Cancel transmission of all buffers enqueued by [QueueBuffer\(\)](#).
- virtual [BufferStatus](#) [WaitForBuffer](#) (void **ppUser)=0
Waits until the transmission of a buffer has been finished.

4.37.1 Member Enumeration Documentation

4.37.1.1 enum [BaslerCamera::IOutDataStream::BufferStatus](#)

Enumerator:

- [bsOk](#)** success
- [bsStreamClosed](#)** stream has been closed by the client
- [bsError](#)** buffer has been cancelled

4.37.2 Member Function Documentation

4.37.2.1 virtual void [BaslerCamera::IOutDataStream::QueueBuffer](#) (size_t nBytes, void * [pBuffer](#), void * [pUser](#)) [pure virtual]

[QueueBuffer\(\)](#) accepts a buffer and initiates the transmission of the buffer. [QueueBuffer\(\)](#) doesn't block until the transmission has been completed. When the transmission of a buffer has been completed, the context pointer associated with the buffer can be retrieved by calling [WaitForBuffer\(\)](#). Don't touch a buffer enqueued by calling [QueueBuffer\(\)](#) before retrieving it's associated context pointer by calling [WaitForBuffer\(\)](#). You are allowed to use the buffer pointer itself as context information, for example,

```
Server.QueueBuffer(bsize, pBuffer, pBuffer)
```

To ensure a continuous stream of transferred buffers, you can enqueue several buffers in advance. The transmission is performed in the order the buffers are enqueued.

Parameters:

- ← **nBytes** size of the buffer in bytes
- ← **pBuffer** pointer to the data buffer
- ← **pUser** additional context information that will be returned when the transmission of the buffer has been finished

4.37.2.2 virtual void BaslerCamera::IOutDataStream::FlushBuffers () [pure virtual]

All buffers enqueued by calling [QueueBuffer\(\)](#) are cancelled when calling [FlushBuffers\(\)](#). For each cancelled buffer, the associated context pointer can be retrieved by calling [WaitForBuffer\(\)](#).

4.37.2.3 virtual BufferStatus BaslerCamera::IOutDataStream::WaitForBuffer (void ** ppUser)
[pure virtual]

The context pointer, passed in together with a buffer, will be returned.

Returns:

[IOutDataStream::bsOk](#) when the transmission was successful. Otherwise one of the other [IOutDataStream::BufferStatus](#) values is returned.

Parameters:

- ↔ **ppUser** points to a pointer to the user provided context pointer

4.38 BaslerCamera::StreamServer::IRegisterSet Struct Reference

Interface to be implemented by classes implementing a register set.

Inherited by BaslerCamera::GxStreamServerBase[*virtual*], and BaslerCamera::StreamServer::RegisterSet< Impl >[*virtual*].

Public Types

- typedef uint32_t [Address_t](#)
Type used for addresses.
- enum [Result](#)
Result type of IRegisterSet's methods.

Public Member Functions

- virtual [Result DoReadRegister](#) ([Address_t](#) address, uint32_t *pData)=0
Called when a register is to be read.
- virtual [Result DoWriteRegister](#) ([Address_t](#) address, uint32_t Data)=0

4.38.1 Member Function Documentation

4.38.1.1 virtual [Result](#) BaslerCamera::StreamServer::IRegisterSet::DoReadRegister ([Address_t](#) address, uint32_t * *pData*) [*pure virtual*]

Parameters:

- ← **address** Address to read from
- **pData** The register content will be written to this pointer

Returns:

IRegisterSet::rrOk on success

4.38.1.2 virtual [Result](#) BaslerCamera::StreamServer::IRegisterSet::DoWriteRegister ([Address_t](#) address, uint32_t Data) [*pure virtual*]

Parameters:

- ← **address** Address to write to
- ← **Data** Data to be written

Returns:

IRegisterSet::rrOk on success

4.39 GenApi::IString Struct Reference

Interface for string properties.

Inherits VirtualDestructable.

Public Member Functions

- virtual void [SetValue](#) (const std::string &Value)=0
Set node value.
- virtual const std::string & [operator=](#) (const std::string &Value)=0
Set node value.
- virtual std::string [GetValue](#) ()=0
Get node value.
- virtual std::string [operator\(\)](#) ()=0
Get node value.

4.40 GenApi::LogicalErrorException Class Reference

Exception thrown to indicate logical errors in the program flow.

Inherits [GenApi::GenericException](#).

Public Member Functions

- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std:exception).

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.41 GenApi::OutOfRangeException Class Reference

Exception fired if an argument is out of range.

Inherits [GenApi::GenericException](#).

Public Member Functions

- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std::exception).

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.42 GenApi::PropertyException Class Reference

Exception fired if a property access fails.

Inherits [GenApi::GenericException](#).

Public Member Functions

- const char * [GetDescription](#) () const throw ()
Get error description.
- const char * [GetSourceFileName](#) () const throw ()
Get filename in which the error occurred.
- unsigned int [GetSourceLine](#) () const throw ()
Get line number at which the error occurred.
- virtual const char * [what](#) () const throw ()
Get error description (overwrite from std:exception).

Protected Attributes

- unsigned int [m_SourceLine](#)
Line number at which the error occurred.
- std::string [m_SourceFileName](#)
Filename in which the error occurred.
- std::string [m_Description](#)
Error description.
- std::string [m_What](#)
Complete error message, including file name and line number.

4.43 BaslerCamera::PropertySet Class Reference

Inherited by [BaslerCamera::DeviceInfo](#).

Public Member Functions

- `std::string GetProperty (const std::string &key) const`
Returns a property.
- `void AddProperty (const std::string &key, const std::string &value)`
Adds a property.
- `bool HasProperty (const std::string &key) const`
Checks if the container contains a property.

4.43.1 Detailed Description

Container for key-value pairs

4.43.2 Member Function Documentation

4.43.2.1 `std::string BaslerCamera::PropertySet::GetProperty (const std::string & key) const`

Parameters:

key The name of the property

Returns:

the value of the property If the property set doesn't contain a value for the specified key, a Basler-Camera::RuntimeException will be thrown.

4.43.2.2 `void BaslerCamera::PropertySet::AddProperty (const std::string & key, const std::string & value)`

Parameters:

key Name of the property

value The value of the property

4.43.2.3 `bool BaslerCamera::PropertySet::HasProperty (const std::string & key) const`

Parameters:

key The name of the property to check

Returns:

true when the property exists, false otherwise.

4.44 GenApi::RuntimeException Class Reference

Runtime exception.

Inherits [GenApi::GenericException](#).

Public Member Functions

- `const char * GetDescription () const throw ()`
Get error description.
- `const char * GetSourceFileName () const throw ()`
Get filename in which the error occurred.
- `unsigned int GetSourceLine () const throw ()`
Get line number at which the error occurred.
- `virtual const char * what () const throw ()`
Get error description (overwrite from `std::exception`).

Protected Attributes

- `unsigned int m\_SourceLine`
Line number at which the error occurred.
- `std::string m\_SourceFileName`
Filename in which the error occurred.
- `std::string m\_Description`
Error description.
- `std::string m\_What`
Complete error message, including file name and line number.

Chapter 5

File Documentation

5.1 BaslerCam.h File Reference

The main include file to be included for the use of the Basler XCam library.

Namespaces

- namespace [BaslerCamera](#)

Defines

- #define [CAMERANAMESPACE](#) XCamInterface

The namespace containing the camera control interface and related enumeration types.

Typedefs

- typedef [XCam](#) [BaslerCamera::Camera](#)

The Camera object used to parameterize the camera and to capture images.

5.1.1 Detailed Description

Before including [BaslerCam.h](#), the define `USE_XCAM` or `USE_GXCAM` must be defined to ensure that the `Camera` typedef is available.

`USE_XCAM` should be defined when `Camera` should refer to the XCam device running on the eXcite camera.

`USE_GXCAM` should be defined when `Camera` should refer to the GxCam device used for remote access of an eXcite camera. On the eXcite, the `gxbaslercam` server application must be running.

5.2 CamT.h File Reference

Namespaces

- namespace [BaslerCamera](#)
- namespace **BaslerCameraPrivate**

Classes

- class [BaslerCamera::CamT](#) < TliDelegate, ApImpl >
Class template used to implement a camera device.

5.2.1 Detailed Description

5.3 DataStream.h File Reference

Definition of interfaces IInDataStream & IOutDataStream.

Namespaces

- namespace [BaslerCamera](#)

Classes

- struct [BaslerCamera::IInDataStream](#)
Interface to be implemented by devices grabbing frames from an (image) data stream.
- struct [BaslerCamera::IOutDataStream](#)
Interface to be implemented by classes providing an (image) data stream.

5.3.1 Detailed Description

5.4 Device.h File Reference

Declaration of the IDevice interface.

Namespaces

- namespace [BaslerCamera](#)

Classes

- class [BaslerCamera::DeviceIoException](#)
Exception class thrown to indicate device I/O related errors.
- struct [BaslerCamera::IDevice](#)
Interface to be implemented by devices.

Defines

- `#define        DEVICE\_IO\_EXCEPTION        GenApi::ExceptionReporter<BaslerCamera::DeviceIoException>(__FILE__, __LINE__).Report`
Fires an DeviceIoException.

5.4.1 Detailed Description

5.4.2 Define Documentation

5.4.2.1 `#define DEVICE_IO_EXCEPTION GenApi::ExceptionReporter<BaslerCamera::DeviceIoException>(__FILE__, __LINE__).Report`

Example:

```
throw DEVICE_IO_EXCEPTION("Failed to open device %s", deviceName)
```

5.5 Exception.h File Reference

Exception classes and helper macros.

Namespaces

- namespace **GenApi**

Classes

- class [GenApi::GenericException](#)
Base class for all exceptions thrown by the library.
- class [GenApi::InvalidArgumentException](#)
Exception fired when an argument is invalid.
- class [GenApi::OutOfRangeException](#)
Exception fired if an argument is out of range.
- class [GenApi::PropertyException](#)
Exception fired if a property access fails.
- class [GenApi::RuntimeException](#)
Runtime exception.
- class [GenApi::LogicalErrorException](#)
Exception thrown to indicate logical errors in the program flow.

Defines

- `#define GENERIC\_EXCEPTION GenApi::ExceptionReporter<GenApi::GenericException>(__FILE__, __LINE__).Report`
Fires an exception, e.g., throw `EXCEPTION("%ld too large", Value);`.
- `#define INVALID\_ARGUMENT\_EXCEPTION GenApi::ExceptionReporter<GenApi::InvalidArgumentException>(__FILE__, __LINE__).Report`
Fires an invalid argument exception, e.g., throw `INVALID_ARGUMENT("%ld too large", Value);`.
- `#define OUT\_OF\_RANGE\_EXCEPTION GenApi::ExceptionReporter<GenApi::OutOfRangeException>(__FILE__, __LINE__).Report`
Fires an out of range exception, e.g., throw `OUT_OF_RANGE_EXCEPTION("%ld too large", Value);`.
- `#define PROPERTY\_EXCEPTION GenApi::ExceptionReporter<GenApi::PropertyException>(__FILE__, __LINE__).Report`
Fires an property exception, e.g., throw `PROPERTY_EXCEPTION("%ld too large", Value);`.
- `#define RUNTIME\_EXCEPTION GenApi::ExceptionReporter<GenApi::RuntimeException>(__FILE__, __LINE__).Report`

Fires a runtime exception, e.g., throw `RUNTIME_EXCEPTION("buh!")`.

- #define `LOGICAL_ERROR_EXCEPTION` `GenApi::ExceptionReporter<GenApi::LogicalErrorException>(__FILE__, __LINE__).Report`

Fires a logical error exception, e.g., throw `LOGICAL_ERROR_EXCEPTION("Should never reach this point")`.

- #define `CHECK_RANGE_I64`(`_Value`, `_Min`, `_Max`)

Checks for range violations, e.g., throw `CHECK_RANGE_I64(Value, 0, 100)`;

5.5.1 Detailed Description

5.6 GxClient.h File Reference

Instantiation of the GxClient device, a device receiving streaming data from a GxStreamServer application.

Namespaces

- namespace [BaslerCamera](#)

Typedefs

- typedef CamT< GxDelegateT< [GxClientInterface::CGxClientInterface](#) >, [GxClientInterface::CGxClientInterface](#) > [BaslerCamera::GxClient](#)

A device class used to connect to a [BaslerCamera::GxStreamServer](#).

5.6.1 Detailed Description

5.7 GxClientInterface.h File Reference

Namespaces

- namespace [GxClientInterface](#)

Classes

- class [GxClientInterface::CGxClientInterface](#)
The device's control interface.

5.7.1 Detailed Description

5.8 GxStreamServer.h File Reference

Definition of the GxStreamServer related classes.

Namespaces

- namespace **BaslerCameraPrivate**
- namespace [BaslerCamera](#)
- namespace [BaslerCamera::StreamServer](#)

Classes

- struct [BaslerCamera::StreamServer::IRegisterSet](#)
Interface to be implemented by classes implementing a register set.
- class [BaslerCamera::GxStreamServer](#)
A customizable stream server class that allows sending a stream consisting of fixed size image data frames.

5.8.1 Detailed Description

5.9 IBoolean.h File Reference

Definition of the IBoolean interface.

Namespaces

- namespace **GenApi**

Classes

- struct [GenApi::IBoolean](#)
Interface for Boolean properties.

5.9.1 Detailed Description

5.10 IFloat.h File Reference

Definition of the IFloat interface.

Namespaces

- namespace **GenApi**

Classes

- struct [GenApi::IFloat](#)
Interface for float properties.

5.10.1 Detailed Description

5.11 Integer.h File Reference

Definition of the Integer interface.

Namespaces

- namespace **GenApi**

Classes

- struct [GenApi::Integer](#)
Interface for integer properties.

Typedefs

- typedef std::list< Integer * > [GenApi::IntegerList_t](#)
List of Integer-node references.

5.11.1 Detailed Description

5.12 IString.h File Reference

Definition of the IString interface.

Namespaces

- namespace **GenApi**

Classes

- struct [GenApi::IString](#)
Interface for string properties.

5.12.1 Detailed Description

5.13 XCam.h File Reference

Declares the XCam typedef library as transport layer.

Namespaces

- namespace [BaslerCamera](#)

Typedefs

- typedef CamT< XCamDelegateT< [XCamInterface::CXCamInterface](#) >, [XCamInterface::CXCamInterface](#) > [BaslerCamera::XCam](#)

The eXcite camera device.

5.13.1 Detailed Description

5.14 XCamInterface.h File Reference

Namespaces

- namespace [XCamInterface](#)

Classes

- class [XCamInterface::CEnumeration_ColorCodingEnums](#)
Enumeration class used for the ColorCoding parameter.
- class [XCamInterface::CEnumeration_TriggerModeEnums](#)
Enumeration class used for the TriggerMode parameter.
- class [XCamInterface::CEnumeration_TriggerPolarityEnums](#)
Enumeration class used for the TriggerPolarity parameter.
- class [XCamInterface::CEnumeration_TriggerSourceEnums](#)
Enumeration class used for the TriggerSource parameter.
- class [XCamInterface::CEnumeration_TestImageEnums](#)
Enumeration class used for the TestImage parameter.
- class [XCamInterface::CEnumeration_PioOut0SrcEnums](#)
Enumeration class used for the PioOut0Src parameter.
- class [XCamInterface::CEnumeration_PioOut0MonitorEnums](#)
Enumeration class used for the PioOut0Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut0SettingEnums](#)
Enumeration class used for the PioOut0Setting parameter.
- class [XCamInterface::CEnumeration_PioOut1SrcEnums](#)
Enumeration class used for the PioOut1Src parameter.
- class [XCamInterface::CEnumeration_PioOut1MonitorEnums](#)
Enumeration class used for the PioOut1Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut1SettingEnums](#)
Enumeration class used for the PioOut1Setting parameter.
- class [XCamInterface::CEnumeration_PioOut2SrcEnums](#)
Enumeration class used for the PioOut2Src parameter.
- class [XCamInterface::CEnumeration_PioOut2MonitorEnums](#)
Enumeration class used for the PioOut2Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut2SettingEnums](#)
Enumeration class used for the PioOut2Setting parameter.

- class [XCamInterface::CEnumeration_PioOut3SrcEnums](#)
Enumeration class used for the PioOut3Src parameter.
- class [XCamInterface::CEnumeration_PioOut3MonitorEnums](#)
Enumeration class used for the PioOut3Monitor parameter.
- class [XCamInterface::CEnumeration_PioOut3SettingEnums](#)
Enumeration class used for the PioOut3Setting parameter.
- class [XCamInterface::CEnumeration_Strobe0PolarityEnums](#)
Enumeration class used for the Strobe0Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe1PolarityEnums](#)
Enumeration class used for the Strobe1Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe2PolarityEnums](#)
Enumeration class used for the Strobe2Polarity parameter.
- class [XCamInterface::CEnumeration_Strobe3PolarityEnums](#)
Enumeration class used for the Strobe3Polarity parameter.
- class [XCamInterface::CEnumeration_VideoModeEnums](#)
Enumeration class used for the VideoMode parameter.
- class [XCamInterface::CXCamInterface](#)
The device's control interface.

5.14.1 Detailed Description

Revision History

Doc. ID Number	Date	Changes
DA00074901	21 June 2005	Initial release of the eXcite API Reference (for prototype cameras only).
DA00074902	30 Sept 2005	Initial release of the eXcite API reference for production cameras.
DA00074903	13 April 2006	Changed address in Singapore. Replaced "include" by "lib" in sample source code in Section 1.3.2 "Library Files". Removed the following: IBoolean & ExtDataStream, IBoolean & FrameCounter, IBoolean & CrcChecksum, IBoolean & DcamValues. Made minor changes to PioOut0Src_SerialTx. Added the following: IBoolean & ShutterTimeBase Enable, IFloat & ShutterTimeBase.

Feedback

Your feedback will help us improve our documentation. Please click the link below to access an online feedback form. Your input is greatly appreciated.

<http://www.baslerweb.com/umfrage/survey.html>

Index

- AddProperty
 - BaslerCamera::DeviceInfo, 77
 - BaslerCamera::PropertySet, 101
- BaslerCam.h, 103
- BaslerCamera, 37
- BaslerCamera::CamT, 43
 - bsCancelled, 44
 - bsError, 44
 - bsOk, 44
 - bsTimeout, 44
- BaslerCamera::CamT
 - BufferStatus, 44
 - FinishGrab, 45
 - FlushBuffers, 45
 - PrepareGrab, 45
 - QueueBuffer, 45
 - WaitForBuffer, 45
- BaslerCamera::DeviceInfo, 77
- BaslerCamera::DeviceInfo
 - AddProperty, 77
 - GetProperty, 77
 - HasProperty, 78
- BaslerCamera::DeviceIoException, 79
- BaslerCamera::DeviceManager, 80
- BaslerCamera::DeviceManager
 - CreateDevice, 80
 - DestroyDevice, 80
 - EnumerateDevices, 80
- BaslerCamera::GxStreamServer, 83
 - bsError, 85
 - bsOk, 85
 - bsStreamClosed, 85
- BaslerCamera::GxStreamServer
 - BufferStatus, 85
 - DoReadRegister, 85
 - DoWriteRegister, 85
 - GxStreamServer, 85
 - Open, 85
 - ReadHandler_t, 84
 - WaitForStreamClose, 86
 - WaitForStreamingEnabled, 86
 - WaitForStreamOpen, 86
 - WriteHandler_t, 84
- BaslerCamera::IDevice, 88
 - GetDeviceInfo, 88
- BaslerCamera::IInDataStream, 90
 - bsCancelled, 90
 - bsError, 90
 - bsOk, 90
 - bsTimeout, 90
- BaslerCamera::IInDataStream
 - BufferStatus, 90
 - FinishGrab, 91
 - FlushBuffers, 91
 - PrepareGrab, 90
 - QueueBuffer, 91
 - WaitForBuffer, 91
- BaslerCamera::IOutDataStream, 94
 - bsError, 94
 - bsOk, 94
 - bsStreamClosed, 94
- BaslerCamera::IOutDataStream
 - BufferStatus, 94
 - FlushBuffers, 95
 - QueueBuffer, 94
 - WaitForBuffer, 95
- BaslerCamera::PropertySet, 101
- BaslerCamera::PropertySet
 - AddProperty, 101
 - GetProperty, 101
 - HasProperty, 101
- BaslerCamera::StreamServer, 39
- BaslerCamera::StreamServer::IRegisterSet, 96
- BaslerCamera::StreamServer::IRegisterSet
 - DoReadRegister, 96
 - DoWriteRegister, 96
- bsCancelled
 - BaslerCamera::CamT, 44
 - BaslerCamera::IInDataStream, 90
- bsError
 - BaslerCamera::CamT, 44
 - BaslerCamera::GxStreamServer, 85
 - BaslerCamera::IInDataStream, 90
 - BaslerCamera::IOutDataStream, 94
- bsOk
 - BaslerCamera::CamT, 44
 - BaslerCamera::GxStreamServer, 85
 - BaslerCamera::IInDataStream, 90

BaslerCamera::IOutDataStream, [94](#)
bsStreamClosed
BaslerCamera::GxStreamServer, [85](#)
BaslerCamera::IOutDataStream, [94](#)
bsTimeout
BaslerCamera::CamT, [44](#)
BaslerCamera::IInDataStream, [90](#)
BufferStatus
BaslerCamera::CamT, [44](#)
BaslerCamera::GxStreamServer, [85](#)
BaslerCamera::IInDataStream, [90](#)
BaslerCamera::IOutDataStream, [94](#)

Camera, [29](#)
CamT, [30](#)
CamT.h, [104](#)
ColorCoding_Mono16
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_Mono8
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_Raw16
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_Raw8
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_RGB16
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_RGB8
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_SMono16
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_SRGB16
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_VendorSpecific0
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_VendorSpecific1
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_VendorSpecific2
XCamInterface::CEnumeration_
ColorCodingEnums, [48](#)
ColorCoding_VendorSpecific3
XCamInterface::CEnumeration_
ColorCodingEnums, [48](#)
ColorCoding_VendorSpecific4
XCamInterface::CEnumeration_
ColorCodingEnums, [48](#)
ColorCoding_VendorSpecific5
XCamInterface::CEnumeration_
ColorCodingEnums, [48](#)
ColorCoding_VendorSpecific6
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_YUV8_4_2_2
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCoding_YUV8_4_4_4
XCamInterface::CEnumeration_
ColorCodingEnums, [47](#)
ColorCodingEnums
XCamInterface::CEnumeration_ColorCoding-
Enums, [47](#)
CreateDevice
BaslerCamera::DeviceManager, [80](#)
CXCamInterface, [31](#)

DataStream.h, [105](#)
DestroyDevice
BaslerCamera::DeviceManager, [80](#)
Device Manager, [28](#)
Device.h, [106](#)
DEVICE_IO_EXCEPTION, [106](#)
DEVICE_IO_EXCEPTION
Device.h, [106](#)
DoReadRegister
BaslerCamera::GxStreamServer, [85](#)
BaslerCamera::StreamServer::IRegisterSet,
[96](#)
DoWriteRegister
BaslerCamera::GxStreamServer, [85](#)
BaslerCamera::StreamServer::IRegisterSet,
[96](#)

Enum types for the data members of 'Camera', [34](#)
EnumerateDevices
BaslerCamera::DeviceManager, [80](#)
Exception.h, [107](#)
Exceptions, [27](#)

FinishGrab
BaslerCamera::CamT, [45](#)
BaslerCamera::IInDataStream, [91](#)
FlushBuffers
BaslerCamera::CamT, [45](#)
BaslerCamera::IInDataStream, [91](#)
BaslerCamera::IOutDataStream, [95](#)

[GenApi::GenericException, 82](#)
[GenApi::IBoolean, 87](#)
[GenApi::IFloat, 89](#)
[GenApi::Integer, 92](#)
[GenApi::InvalidArgumentException, 93](#)
[GenApi::IString, 97](#)
[GenApi::LogicalErrorException, 98](#)
[GenApi::OutOfRangeException, 99](#)
[GenApi::PropertyException, 100](#)
[GenApi::RuntimeException, 102](#)
[GetDeviceInfo](#)
 [BaslerCamera::IDevice, 88](#)
[GetProperty](#)
 [BaslerCamera::DeviceInfo, 77](#)
 [BaslerCamera::PropertySet, 101](#)
[GxClient](#)
 [imgxfer_group, 36](#)
[GxClient.h, 109](#)
[GxClientInterface, 40](#)
[GxClientInterface.h, 110](#)
[GxClientInterface::CGxClientInterface, 70](#)
[GxStreamServer](#)
 [BaslerCamera::GxStreamServer, 85](#)
[GxStreamServer.h, 111](#)

[HasProperty](#)
 [BaslerCamera::DeviceInfo, 78](#)
 [BaslerCamera::PropertySet, 101](#)

[IBoolean.h, 112](#)
[IFloat.h, 113](#)
[Integer.h, 114](#)
[Image transfer from an eXcite to a PC, 36](#)
[imgxfer_group](#)
 [GxClient, 36](#)
[IString.h, 115](#)

[Non-enum types for the data members of 'Camera', 33](#)

[Open](#)
 [BaslerCamera::GxStreamServer, 85](#)

[PioOut0Monitor_High](#)
 [XCamlInterface::CEnumeration_-PioOut0MonitorEnums, 49](#)
[PioOut0Monitor_Low](#)
 [XCamlInterface::CEnumeration_-PioOut0MonitorEnums, 49](#)
[PioOut0MonitorEnums](#)
 [XCamlInterface::CEnumeration_PioOut0MonitorEnums, 49](#)
[PioOut0Setting_High](#)
 [XCamlInterface::CEnumeration_-PioOut0SettingEnums, 50](#)

[PioOut0Setting_Low](#)
 [XCamlInterface::CEnumeration_-PioOut0SettingEnums, 50](#)
[PioOut0SettingEnums](#)
 [XCamlInterface::CEnumeration_PioOut0SettingEnums, 50](#)
[PioOut0Src_IntegrationEnable](#)
 [XCamlInterface::CEnumeration_-PioOut0SrcEnums, 51](#)
[PioOut0Src_ReadyforTrigger](#)
 [XCamlInterface::CEnumeration_-PioOut0SrcEnums, 51](#)
[PioOut0Src_SerialTx](#)
 [XCamlInterface::CEnumeration_-PioOut0SrcEnums, 51](#)
[PioOut0Src_Strobe0](#)
 [XCamlInterface::CEnumeration_-PioOut0SrcEnums, 51](#)
[PioOut0Src_UserSet](#)
 [XCamlInterface::CEnumeration_-PioOut0SrcEnums, 51](#)
[PioOut0SrcEnums](#)
 [XCamlInterface::CEnumeration_PioOut0SrcEnums, 51](#)

[PioOut1Monitor_High](#)
 [XCamlInterface::CEnumeration_-PioOut1MonitorEnums, 52](#)
[PioOut1Monitor_Low](#)
 [XCamlInterface::CEnumeration_-PioOut1MonitorEnums, 52](#)
[PioOut1MonitorEnums](#)
 [XCamlInterface::CEnumeration_PioOut1MonitorEnums, 52](#)
[PioOut1Setting_High](#)
 [XCamlInterface::CEnumeration_-PioOut1SettingEnums, 53](#)
[PioOut1Setting_Low](#)
 [XCamlInterface::CEnumeration_-PioOut1SettingEnums, 53](#)
[PioOut1SettingEnums](#)
 [XCamlInterface::CEnumeration_PioOut1SettingEnums, 53](#)
[PioOut1Src_IntegrationEnable](#)
 [XCamlInterface::CEnumeration_-PioOut1SrcEnums, 54](#)
[PioOut1Src_ReadyforTrigger](#)
 [XCamlInterface::CEnumeration_-PioOut1SrcEnums, 54](#)
[PioOut1Src_SerialTx](#)
 [XCamlInterface::CEnumeration_-PioOut1SrcEnums, 54](#)
[PioOut1Src_Strobe1](#)
 [XCamlInterface::CEnumeration_-PioOut1SrcEnums, 54](#)

PioOut1Src_UserSet
 XCamInterface::CEnumeration_
 PioOut1SrcEnums, [54](#)

PioOut1SrcEnums
 XCamInterface::CEnumeration_PioOut1Src-
 Enums, [54](#)

PioOut2Monitor_High
 XCamInterface::CEnumeration_
 PioOut2MonitorEnums, [55](#)

PioOut2Monitor_Low
 XCamInterface::CEnumeration_
 PioOut2MonitorEnums, [55](#)

PioOut2MonitorEnums
 XCamInterface::CEnumeration_Pio-
 Out2MonitorEnums, [55](#)

PioOut2Setting_High
 XCamInterface::CEnumeration_
 PioOut2SettingEnums, [56](#)

PioOut2Setting_Low
 XCamInterface::CEnumeration_
 PioOut2SettingEnums, [56](#)

PioOut2SettingEnums
 XCamInterface::CEnumeration_Pio-
 Out2SettingEnums, [56](#)

PioOut2Src_IntegrationEnable
 XCamInterface::CEnumeration_
 PioOut2SrcEnums, [57](#)

PioOut2Src_ReadyforTrigger
 XCamInterface::CEnumeration_
 PioOut2SrcEnums, [57](#)

PioOut2Src_SerialTx
 XCamInterface::CEnumeration_
 PioOut2SrcEnums, [57](#)

PioOut2Src_Strobe2
 XCamInterface::CEnumeration_
 PioOut2SrcEnums, [57](#)

PioOut2Src_UserSet
 XCamInterface::CEnumeration_
 PioOut2SrcEnums, [57](#)

PioOut2SrcEnums
 XCamInterface::CEnumeration_PioOut2Src-
 Enums, [57](#)

PioOut3Monitor_High
 XCamInterface::CEnumeration_
 PioOut3MonitorEnums, [58](#)

PioOut3Monitor_Low
 XCamInterface::CEnumeration_
 PioOut3MonitorEnums, [58](#)

PioOut3MonitorEnums
 XCamInterface::CEnumeration_Pio-
 Out3MonitorEnums, [58](#)

PioOut3Setting_High
 XCamInterface::CEnumeration_
 PioOut3SettingEnums, [59](#)

PioOut3Setting_Low
 XCamInterface::CEnumeration_
 PioOut3SettingEnums, [59](#)

PioOut3SettingEnums
 XCamInterface::CEnumeration_Pio-
 Out3SettingEnums, [59](#)

PioOut3Src_IntegrationEnable
 XCamInterface::CEnumeration_
 PioOut3SrcEnums, [60](#)

PioOut3Src_ReadyforTrigger
 XCamInterface::CEnumeration_
 PioOut3SrcEnums, [60](#)

PioOut3Src_SerialTx
 XCamInterface::CEnumeration_
 PioOut3SrcEnums, [60](#)

PioOut3Src_Strobe3
 XCamInterface::CEnumeration_
 PioOut3SrcEnums, [60](#)

PioOut3Src_UserSet
 XCamInterface::CEnumeration_
 PioOut3SrcEnums, [60](#)

PioOut3SrcEnums
 XCamInterface::CEnumeration_PioOut3Src-
 Enums, [60](#)

PrepareGrab
 BaslerCamera::CamT, [45](#)
 BaslerCamera::InDataStream, [90](#)

QueueBuffer
 BaslerCamera::CamT, [45](#)
 BaslerCamera::InDataStream, [91](#)
 BaslerCamera::OutDataStream, [94](#)

ReadHandler_t
 BaslerCamera::GxStreamServer, [84](#)

Strobe0Polarity_HighActiveOutput
 XCamInterface::CEnumeration_
 Strobe0PolarityEnums, [61](#)

Strobe0Polarity_LowActiveOutput
 XCamInterface::CEnumeration_
 Strobe0PolarityEnums, [61](#)

Strobe0PolarityEnums
 XCamInterface::CEnumeration_
 Strobe0PolarityEnums, [61](#)

Strobe1Polarity_HighActiveOutput
 XCamInterface::CEnumeration_
 Strobe1PolarityEnums, [62](#)

Strobe1Polarity_LowActiveOutput
 XCamInterface::CEnumeration_
 Strobe1PolarityEnums, [62](#)

Strobe1PolarityEnums
 XCamInterface::CEnumeration_
 Strobe1PolarityEnums, [62](#)

Strobe2Polarity_HighActiveOutput
 XCamInterface::CEnumeration_
 Strobe2PolarityEnums, [63](#)
 Strobe2Polarity_LowActiveOutput
 XCamInterface::CEnumeration_
 Strobe2PolarityEnums, [63](#)
 Strobe2PolarityEnums
 XCamInterface::CEnumeration_
 Strobe2PolarityEnums, [63](#)
 Strobe3Polarity_HighActiveOutput
 XCamInterface::CEnumeration_
 Strobe3PolarityEnums, [64](#)
 Strobe3Polarity_LowActiveOutput
 XCamInterface::CEnumeration_
 Strobe3PolarityEnums, [64](#)
 Strobe3PolarityEnums
 XCamInterface::CEnumeration_
 Strobe3PolarityEnums, [64](#)

 TestImage_Disabled
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage1
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage2
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage3
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage4
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage5
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage6
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImage_TestImage7
 XCamInterface::CEnumeration_
 TestImageEnums, [65](#)
 TestImageEnums
 XCamInterface::CEnumeration_TestImage-
 Enums, [65](#)
 TriggerMode_TriggerMode0
 XCamInterface::CEnumeration_
 TriggerModeEnums, [66](#)
 TriggerMode_TriggerMode1
 XCamInterface::CEnumeration_
 TriggerModeEnums, [66](#)
 TriggerMode_TriggerMode2
 XCamInterface::CEnumeration_
 TriggerModeEnums, [66](#)
 TriggerModeEnums
 XCamInterface::CEnumeration_TriggerMode-
 Enums, [66](#)
 TriggerPolarity_HighActive
 XCamInterface::CEnumeration_
 TriggerPolarityEnums, [67](#)
 TriggerPolarity_LowActive
 XCamInterface::CEnumeration_
 TriggerPolarityEnums, [67](#)
 TriggerPolarityEnums
 XCamInterface::CEnumeration_Trigger-
 PolarityEnums, [67](#)
 TriggerSource_ExTrigPort0
 XCamInterface::CEnumeration_
 TriggerSourceEnums, [68](#)
 TriggerSource_ExTrigPort1
 XCamInterface::CEnumeration_
 TriggerSourceEnums, [68](#)
 TriggerSource_ExTrigPort2
 XCamInterface::CEnumeration_
 TriggerSourceEnums, [68](#)
 TriggerSource_ExTrigPort3
 XCamInterface::CEnumeration_
 TriggerSourceEnums, [68](#)
 TriggerSource_SoftTrig
 XCamInterface::CEnumeration_
 TriggerSourceEnums, [68](#)
 TriggerSourceEnums
 XCamInterface::CEnumeration_TriggerSource-
 Enums, [68](#)
 Types for the data members of 'Camera', [32](#)

 VideoMode_VideoMode0
 XCamInterface::CEnumeration_
 VideoModeEnums, [69](#)
 VideoMode_VideoMode1
 XCamInterface::CEnumeration_
 VideoModeEnums, [69](#)
 VideoMode_VideoMode2
 XCamInterface::CEnumeration_
 VideoModeEnums, [69](#)
 VideoMode_VideoMode3
 XCamInterface::CEnumeration_
 VideoModeEnums, [69](#)
 VideoMode_VideoMode4
 XCamInterface::CEnumeration_
 VideoModeEnums, [69](#)
 VideoMode_VideoMode5

`XCamInterface::CEnumeration_`
 `VideoModeEnums`, 69

`VideoMode_VideoMode6`
 `XCamInterface::CEnumeration_`
 `VideoModeEnums`, 69

`VideoMode_VideoMode7`
 `XCamInterface::CEnumeration_`
 `VideoModeEnums`, 69

`VideoModeEnums`
 `XCamInterface::CEnumeration_VideoMode-`
 `Enums`, 69

`WaitForBuffer`
 `BaslerCamera::CamT`, 45
 `BaslerCamera::InDataStream`, 91
 `BaslerCamera::OutDataStream`, 95

`WaitForStreamClose`
 `BaslerCamera::GxStreamServer`, 86

`WaitForStreamingEnabled`
 `BaslerCamera::GxStreamServer`, 86

`WaitForStreamOpen`
 `BaslerCamera::GxStreamServer`, 86

`WriteHandler_t`
 `BaslerCamera::GxStreamServer`, 84

`XCam.h`, 116

`XCamInterface`, 41

`XCamInterface.h`, 117

`XCamInterface::CEnumeration_ColorCodingEnums`
 `ColorCoding_Mono16`, 47
 `ColorCoding_Mono8`, 47
 `ColorCoding_Raw16`, 47
 `ColorCoding_Raw8`, 47
 `ColorCoding_RGB16`, 47
 `ColorCoding_RGB8`, 47
 `ColorCoding_SMono16`, 47
 `ColorCoding_SRGB16`, 47
 `ColorCoding_VendorSpecific0`, 47
 `ColorCoding_VendorSpecific1`, 47
 `ColorCoding_VendorSpecific2`, 48
 `ColorCoding_VendorSpecific3`, 48
 `ColorCoding_VendorSpecific4`, 48
 `ColorCoding_VendorSpecific5`, 48
 `ColorCoding_VendorSpecific6`, 48
 `ColorCoding_YUV8_4_1_1`, 47
 `ColorCoding_YUV8_4_2_2`, 47
 `ColorCoding_YUV8_4_4_4`, 47

`XCamInterface::CEnumeration_`
 `ColorCodingEnums`, 47

`XCamInterface::CEnumeration_ColorCodingEnums`
 `ColorCodingEnums`, 47

`XCamInterface::CEnumeration_`
 `PioOut0MonitorEnums`
 `PioOut0Monitor_High`, 49
 `PioOut0Monitor_Low`, 49

`XCamInterface::CEnumeration_`
 `PioOut0MonitorEnums`, 49

`XCamInterface::CEnumeration_PioOut0Monitor-`
 `Enums`
 `PioOut0MonitorEnums`, 49

`XCamInterface::CEnumeration_`
 `PioOut0SettingEnums`
 `PioOut0Setting_High`, 50
 `PioOut0Setting_Low`, 50

`XCamInterface::CEnumeration_`
 `PioOut0SettingEnums`, 50

`XCamInterface::CEnumeration_PioOut0Setting-`
 `Enums`
 `PioOut0SettingEnums`, 50

`XCamInterface::CEnumeration_PioOut0SrcEnums`
 `PioOut0Src_IntegrationEnable`, 51
 `PioOut0Src_ReadyforTrigger`, 51
 `PioOut0Src_SerialTx`, 51
 `PioOut0Src_Strobe0`, 51
 `PioOut0Src_UserSet`, 51

`XCamInterface::CEnumeration_PioOut0SrcEnums`, 51

`XCamInterface::CEnumeration_PioOut0SrcEnums`
 `PioOut0SrcEnums`, 51

`XCamInterface::CEnumeration_`
 `PioOut1MonitorEnums`
 `PioOut1Monitor_High`, 52
 `PioOut1Monitor_Low`, 52

`XCamInterface::CEnumeration_`
 `PioOut1MonitorEnums`, 52

`XCamInterface::CEnumeration_PioOut1Monitor-`
 `Enums`
 `PioOut1MonitorEnums`, 52

`XCamInterface::CEnumeration_`
 `PioOut1SettingEnums`
 `PioOut1Setting_High`, 53
 `PioOut1Setting_Low`, 53

`XCamInterface::CEnumeration_`
 `PioOut1SettingEnums`, 53

`XCamInterface::CEnumeration_PioOut1Setting-`
 `Enums`
 `PioOut1SettingEnums`, 53

`XCamInterface::CEnumeration_PioOut1SrcEnums`
 `PioOut1Src_IntegrationEnable`, 54
 `PioOut1Src_ReadyforTrigger`, 54
 `PioOut1Src_SerialTx`, 54
 `PioOut1Src_Strobe1`, 54
 `PioOut1Src_UserSet`, 54

`XCamInterface::CEnumeration_PioOut1SrcEnums`, 54

`XCamInterface::CEnumeration_PioOut1SrcEnums`
 `PioOut1SrcEnums`, 54

XCamInterface::CEnumeration_
 PioOut2MonitorEnums
 PioOut2Monitor_High, [55](#)
 PioOut2Monitor_Low, [55](#)
 XCamInterface::CEnumeration_
 PioOut2MonitorEnums, [55](#)
 XCamInterface::CEnumeration_PioOut2Monitor-
 Enums
 PioOut2MonitorEnums, [55](#)
 XCamInterface::CEnumeration_
 PioOut2SettingEnums
 PioOut2Setting_High, [56](#)
 PioOut2Setting_Low, [56](#)
 XCamInterface::CEnumeration_
 PioOut2SettingEnums, [56](#)
 XCamInterface::CEnumeration_PioOut2Setting-
 Enums
 PioOut2SettingEnums, [56](#)
 XCamInterface::CEnumeration_PioOut2SrcEnums
 PioOut2Src_IntegrationEnable, [57](#)
 PioOut2Src_ReadyforTrigger, [57](#)
 PioOut2Src_SerialTx, [57](#)
 PioOut2Src_Strobe2, [57](#)
 PioOut2Src_UserSet, [57](#)
 XCamInterface::CEnumeration_PioOut2SrcEnums,
 [57](#)
 XCamInterface::CEnumeration_PioOut2SrcEnums
 PioOut2SrcEnums, [57](#)
 XCamInterface::CEnumeration_
 PioOut3MonitorEnums
 PioOut3Monitor_High, [58](#)
 PioOut3Monitor_Low, [58](#)
 XCamInterface::CEnumeration_
 PioOut3MonitorEnums, [58](#)
 XCamInterface::CEnumeration_PioOut3Monitor-
 Enums
 PioOut3MonitorEnums, [58](#)
 XCamInterface::CEnumeration_
 PioOut3SettingEnums
 PioOut3Setting_High, [59](#)
 PioOut3Setting_Low, [59](#)
 XCamInterface::CEnumeration_
 PioOut3SettingEnums, [59](#)
 XCamInterface::CEnumeration_PioOut3Setting-
 Enums
 PioOut3SettingEnums, [59](#)
 XCamInterface::CEnumeration_PioOut3SrcEnums
 PioOut3Src_IntegrationEnable, [60](#)
 PioOut3Src_ReadyforTrigger, [60](#)
 PioOut3Src_SerialTx, [60](#)
 PioOut3Src_Strobe3, [60](#)
 PioOut3Src_UserSet, [60](#)
 XCamInterface::CEnumeration_PioOut3SrcEnums,
 [60](#)

XCamInterface::CEnumeration_PioOut3SrcEnums
 PioOut3SrcEnums, [60](#)
 XCamInterface::CEnumeration_
 Strobe0PolarityEnums
 Strobe0Polarity_HighActiveOutput, [61](#)
 Strobe0Polarity_LowActiveOutput, [61](#)
 XCamInterface::CEnumeration_
 Strobe0PolarityEnums, [61](#)
 XCamInterface::CEnumeration_Strobe0Polarity-
 Enums
 Strobe0PolarityEnums, [61](#)
 XCamInterface::CEnumeration_
 Strobe1PolarityEnums
 Strobe1Polarity_HighActiveOutput, [62](#)
 Strobe1Polarity_LowActiveOutput, [62](#)
 XCamInterface::CEnumeration_
 Strobe1PolarityEnums, [62](#)
 XCamInterface::CEnumeration_Strobe1Polarity-
 Enums
 Strobe1PolarityEnums, [62](#)
 XCamInterface::CEnumeration_
 Strobe2PolarityEnums
 Strobe2Polarity_HighActiveOutput, [63](#)
 Strobe2Polarity_LowActiveOutput, [63](#)
 XCamInterface::CEnumeration_
 Strobe2PolarityEnums, [63](#)
 XCamInterface::CEnumeration_Strobe2Polarity-
 Enums
 Strobe2PolarityEnums, [63](#)
 XCamInterface::CEnumeration_
 Strobe3PolarityEnums
 Strobe3Polarity_HighActiveOutput, [64](#)
 Strobe3Polarity_LowActiveOutput, [64](#)
 XCamInterface::CEnumeration_
 Strobe3PolarityEnums, [64](#)
 XCamInterface::CEnumeration_Strobe3Polarity-
 Enums
 Strobe3PolarityEnums, [64](#)
 XCamInterface::CEnumeration_TestImageEnums
 TestImage_Disabled, [65](#)
 TestImage_TestImage1, [65](#)
 TestImage_TestImage2, [65](#)
 TestImage_TestImage3, [65](#)
 TestImage_TestImage4, [65](#)
 TestImage_TestImage5, [65](#)
 TestImage_TestImage6, [65](#)
 TestImage_TestImage7, [65](#)
 XCamInterface::CEnumeration_TestImageEnums,
 [65](#)
 XCamInterface::CEnumeration_TestImageEnums
 TestImageEnums, [65](#)
 XCamInterface::CEnumeration_TriggerModeEnums
 TriggerMode_TriggerMode0, [66](#)
 TriggerMode_TriggerMode1, [66](#)

- TriggerMode_TriggerMode2, [66](#)
- TriggerMode_TriggerMode3, [66](#)
- XCamInterface::CEnumeration_ -
 - TriggerModeEnums, [66](#)
- XCamInterface::CEnumeration_TriggerModeEnums
 - TriggerModeEnums, [66](#)
- XCamInterface::CEnumeration_ -
 - TriggerPolarityEnums
 - TriggerPolarity_HighActive, [67](#)
 - TriggerPolarity_LowActive, [67](#)
- XCamInterface::CEnumeration_ -
 - TriggerPolarityEnums, [67](#)
- XCamInterface::CEnumeration_TriggerPolarity-
 - Enums
 - TriggerPolarityEnums, [67](#)
- XCamInterface::CEnumeration_ -
 - TriggerSourceEnums
 - TriggerSource_ExTrigPort0, [68](#)
 - TriggerSource_ExTrigPort1, [68](#)
 - TriggerSource_ExTrigPort2, [68](#)
 - TriggerSource_ExTrigPort3, [68](#)
 - TriggerSource_SoftTrig, [68](#)
- XCamInterface::CEnumeration_ -
 - TriggerSourceEnums, [68](#)
- XCamInterface::CEnumeration_TriggerSource-
 - Enums
 - TriggerSourceEnums, [68](#)
- XCamInterface::CEnumeration_VideoModeEnums
 - VideoMode_VideoMode0, [69](#)
 - VideoMode_VideoMode1, [69](#)
 - VideoMode_VideoMode2, [69](#)
 - VideoMode_VideoMode3, [69](#)
 - VideoMode_VideoMode4, [69](#)
 - VideoMode_VideoMode5, [69](#)
 - VideoMode_VideoMode6, [69](#)
 - VideoMode_VideoMode7, [69](#)
- XCamInterface::CEnumeration_VideoModeEnums,
 - [69](#)
- XCamInterface::CEnumeration_VideoModeEnums
 - VideoModeEnums, [69](#)
- XCamInterface::CXCamInterface, [71](#)