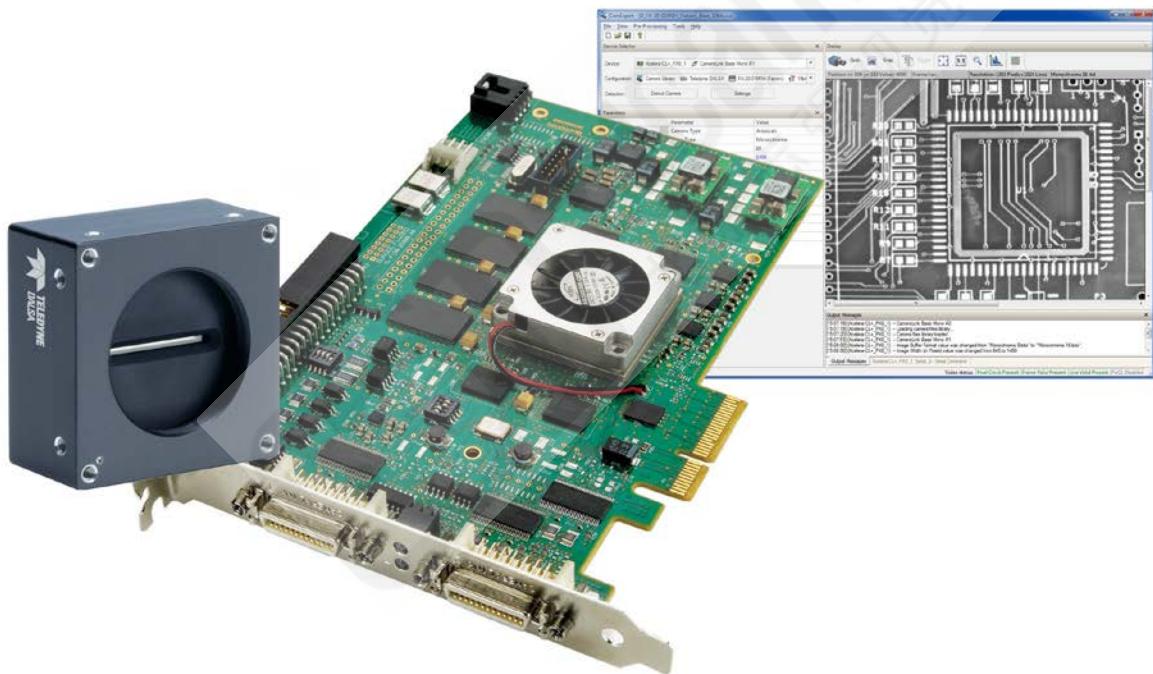


# Sapera LT ++™ 8.10

## Programmer's Manual

sensors | cameras | frame grabbers | processors | **software** | vision solutions



P/N: OC-SAPM- SPPP0  
[www.teledynedalsa.com](http://www.teledynedalsa.com)

 **TELEDYNE DALSA**  
Everywhereyoulook™

## **NOTICE**

© 2015 Teledyne DALSA, inc. All rights reserved.

This document may not be reproduced nor transmitted in any form or by any means, either electronic or mechanical, without the express written permission of TELEDYNE DALSA. Every effort is made to ensure the information in this manual is accurate and reliable. Use of the products described herein is understood to be at the user's risk. TELEDYNE DALSA assumes no liability whatsoever for the use of the products detailed in this document and reserves the right to make changes in specifications at any time and without notice.

Microsoft® is a registered trademark; Windows®, Windows® XP, Windows® Vista, Windows® 7, Windows® 8 are trademarks of Microsoft Corporation.

All other trademarks or intellectual property mentioned herein belongs to their respective owners.

Printed on September 28, 2015

Document Number: OC-SAPM-SPPPO

Printed in Canada

### **About This Manual**

This manual exists in Windows Help, and Adobe Acrobat® (PDF) formats (printed manuals are available as special orders). The Help and PDF formats make full use of hypertext cross-references. The Teledyne DALSA home page on the Internet, located at <http://www.teledynedalsa.com/imaging>, contains documents, software updates, demos, errata, utilities, and more.

### **About Teledyne DALSA**

Teledyne DALSA is an international high performance semiconductor and electronics company that designs, develops, manufactures, and markets digital imaging products and solutions, in addition to providing wafer foundry services.

Teledyne DALSA Digital Imaging offers the widest range of machine vision components in the world. From industry-leading image sensors through powerful and sophisticated cameras, frame grabbers, vision processors and software to easy-to-use vision appliances and custom vision modules.

# Contents

<b>GETTING STARTED.....</b>	<b>5</b>
ABOUT SAPERA LT ++ .....	5
SAPERA LT ARCHITECTURE .....	5
REQUIREMENTS .....	7
FILE LOCATIONS .....	7
<b>HIERARCHY CHARTS.....</b>	<b>8</b>
BASIC CLASS HIERARCHY CHART .....	8
<b>USING SAPERA LT ++.....</b>	<b>9</b>
HEADER FILES, LIBRARIES, AND DLLS.....	9
SAPERA LT ++ - CREATING AN APPLICATION .....	10
DEMONSTRATIONS AND EXAMPLES .....	11
<b>BASIC CLASS REFERENCE .....</b>	<b>12</b>
DATA CLASSES.....	12
SAPACQUISITION .....	17
SAPACQCALLBACKINFO .....	34
SAPACQDEVICE.....	38
SAPACQDEVICECALLBACKINFO .....	55
SAPBUFFER .....	59
SAPBUFFERROI .....	89
SAPBUFFERWITHTRASH.....	93
SAPCOLORCONVERSION .....	96
SAPDISPLAY .....	106
SAPFEATURE .....	112
SAPFLATFIELD .....	128
SAPGIO .....	141
SAPGIOCALLBACKINFO .....	148
SAPLOCATION .....	152
SAPLUT .....	154
SAPMANAGER .....	164
SAPMAN_CALLBACKINFO .....	179
SAPMETADATA .....	182
SAPPERFORMANCE .....	187
SAPPROCESSING .....	189
SAPP_CALLBACKINFO .....	194
SAPTRANSFER.....	195
SPECIALIZED TRANSFER CLASSES.....	209
SAPVIEW .....	211
SAPVIEWCALLBACKINFO .....	225
SAPXFERCALLBACKINFO .....	226
SAPXFERNODE .....	231
SAPXFERPAIR .....	234
SAPXFERPARAMS .....	241
<b>APPENDIX A: SAPERA LT AND GENICAM.....</b>	<b>245</b>
WHAT IS GENICAM? .....	245
USING SAPERA LT WITH GENICAM-COMPLIANT DEVICES .....	245
NOTES ON THE SAPERA LT GENICAM IMPLEMENTATION .....	246
GIGEVISION IN SAPERA LT .....	247

APPENDIX B: OBSOLETE CLASSES.....	250
CONTACT INFORMATION.....	251
SALES INFORMATION.....	251
TECHNICAL SUPPORT.....	252

# Getting Started

## About Sapera LT ++

Sapera™ LT is a software API for controlling image acquisition devices such as frame grabbers and camera. Sapera LT libraries support Teledyne DALSA cameras and frame grabbers as well as hundreds of 3rd party camera models across all common interfaces formats like GigE Vision®, Camera Link®, as well as emerging new image acquisition standards such as CLHS.



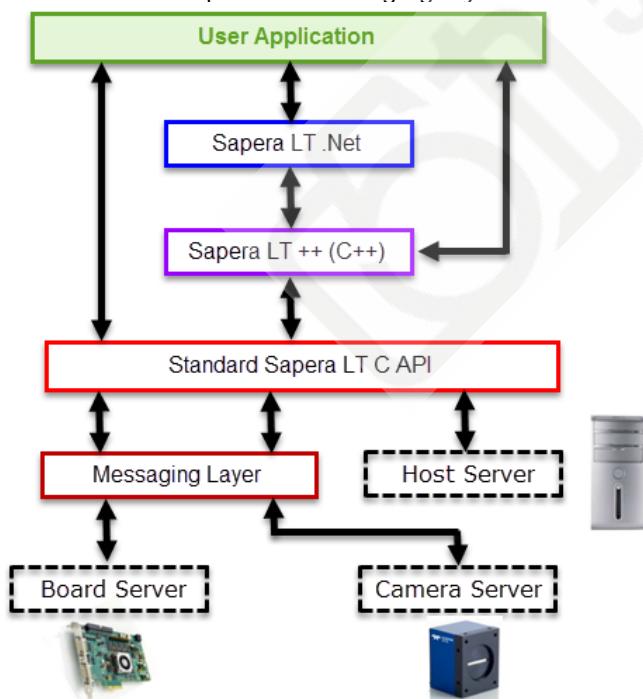
If your application requires image processing or GPU optimization, Sapera Essential, a full-featured image processing library, is available as a separate software package. For more information see [www.teledynedalsa.com/imaging/products/software/](http://www.teledynedalsa.com/imaging/products/software/). □

## Sapera LT Architecture

The following section describes application architecture, related terms, and illustrates Sapera LT's library architecture.

### Application Architecture

The Sapera LT modular architecture allows applications to be distributed on different Sapera LT servers. Each server can run either on the host computer or on a Teledyne DALSA device. Sapera LT calls are routed to different servers via the Sapera LT messaging layer in a fashion completely independent of the underlying hardware.



## What is a server?

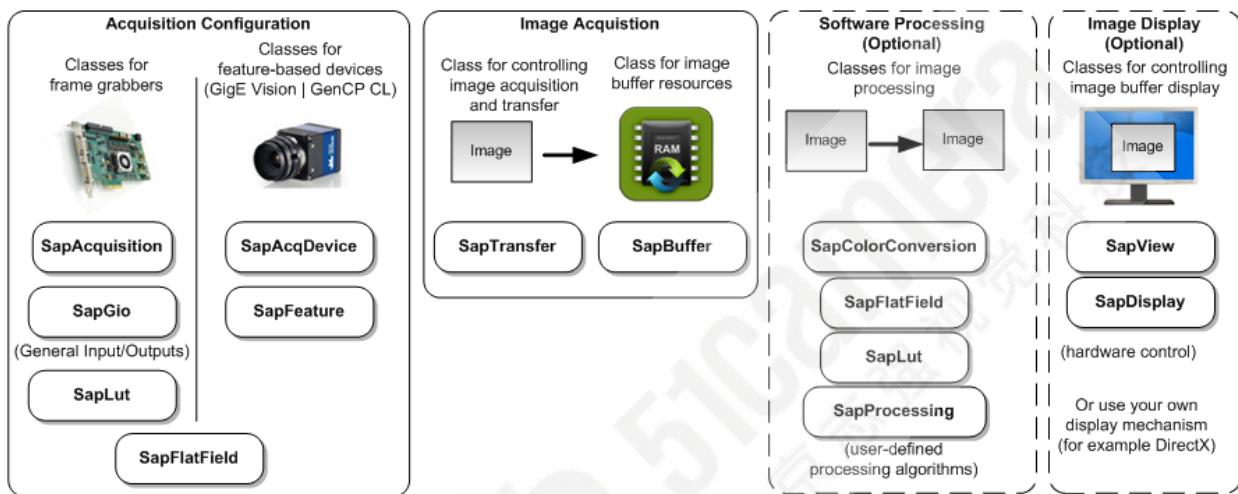
A Sapera Server is an abstract representation of a physical device like a frame grabber, a camera, or a desktop PC. In general, a Teledyne DALSA board is a server. Some processing boards, however, may contain several servers; this is true when using multi-processor boards.

A server allows Sapera applications to interact with the server's resources.

## Library Architecture

The typical machine vision application requires configuration of acquisition resources, image capture and transfer to memory buffers. These image buffers can then be processed or displayed, analyzed, with results determining subsequent processes. Events can also be monitored to trigger appropriate responses. The Sapera LT library architecture is organized around these basic machine vision functional blocks.

The following block diagram, while not exhaustive of all the classes available in Sapera LT, illustrates the major functional blocks with the corresponding classes.



The **Sapera LT User's Manual** provides explanations and multiple code snippets for typical application operations.



It is always recommended to use the source code provided with the demos and examples as both a learning tool and a starting point for your applications. For a complete list and description of the demos and examples included with Sapera LT see the Sapera LT Getting Started Manual.

# Requirements

For 32-bit development, Saper LT ++ currently supports the following compilers:

- Microsoft Visual C++ 2005 (with Service Pack 1)
- Microsoft Visual C++ 2008 (with Service Pack 1)
- Microsoft Visual C++ 2010
- Microsoft Visual C++ 2012
- Microsoft Visual C++ 2013
- Borland C++ Builder XE (versions XE1 to XE5) (*Basic Classes* only)

For 64-bit, development, it supports the following compilers:

- Microsoft Visual C++ 2005 (with Service Pack 1)
- Microsoft Visual C++ 2008 (with Service Pack 1)
- Microsoft Visual C++ 2010
- Microsoft Visual C++ 2012
- Microsoft Visual C++ 2013

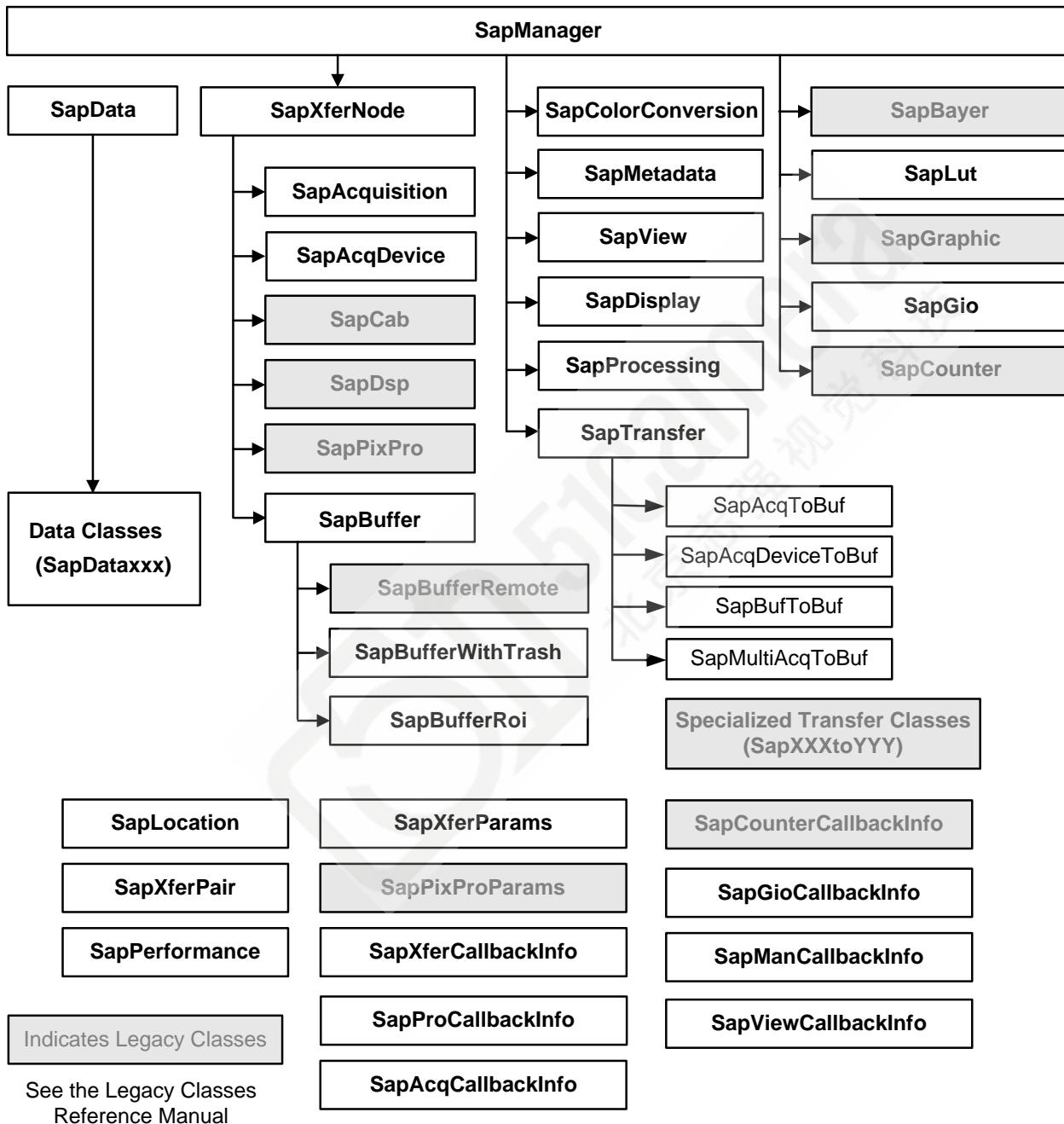
## File Locations

The table below shows the different file groups with their respective locations.

Description	Location
Basic Classes headers	Saper\Classes\Basic
GUI Classes headers and source code	Saper\Classes\Gui
32-bit import librairies (compiler independent)	Saper\Lib\Win32
32-bit import librairies (Visual C++ 2005)	Saper\Lib\Win32\VS2005
32-bit import librairies (Visual C++ 2008) (if compiled)	Saper\Lib\Win32\VS2008
32-bit import librairies (Visual C++ 2010) (if compiled)	Saper\Lib\Win32\VS2010
32-bit import librairies (Visual C++ 2012) (if compiled)	Saper\Lib\Win32\VS2012
32-bit import librairies (Visual C++ 2013) (if compiled)	Saper\Lib\Win32\VS2013
32-bit Import libraries (Borland C++ Builder XE1 to XE5)	Saper\Lib\Win32\Borland
64-bit import librairies (compiler independent)	Saper\Lib\Win64
64-bit import librairies (Visual C++ 2005)	Saper\Lib\Win64\VS2005
64-bit import librairies (Visual C++ 2008) (if compiled)	Saper\Lib\Win64\VS2008
64-bit import librairies (Visual C++ 2010) (if compiled)	Saper\Lib\Win64\VS2010
64-bit import librairies (Visual C++ 2012) (if compiled)	Saper\Lib\Win64\VS2012
64-bit import librairies (Visual C++ 2013) (if compiled)	Saper\Lib\Win64\VS2013
Dynamic-link libraries (DLLs)	Windows system directory (<windir>\System32)

# Hierarchy Charts

## Basic Class Hierarchy Chart



# Using Sapera LT ++

## Header Files, Libraries, and DLLs

The following files are provided with Sapera LT. Also, 'XX' refers to the current Sapera LT version number, for example, **SapClassBasic74.dll** for the version 7.40 Basic Classes DLL.

Starting from Sapera LT 7.50, only the SapClassGui.dll for 2005 is distributed since demos and examples are compiled with this version of Microsoft Visual Studio 2005.

SapClassGui source code is included with all demo Visual Studio solutions (2005-2013), therefore individual SapClassGui.dll files can be compiled depending on the selected platform. When compiled, the SapClassGui.dll is available in the respective Sapera\Lib\Win32\VS20xx or Sapera\Lib\Win64\VS20xx directories.

Note that library and DLL files with the 'D' suffix (for example, **SapClassGuID.lib**) denote debug versions.

File Name	Description	Location
<b>SapClassBasic.h</b>	Basic class header file	<b>Sapera\Classes\Basic</b>
<b>SapClassGui.h</b>	GUI class header file	<b>Sapera\Classes\Gui</b>
<b>SapClassBasic.lib</b>	Basic class libraries for all Visual C++ versions	<b>Sapera\Lib\Win32</b> <b>Sapera\Lib\Win64</b>
<b>SapClassGui.lib</b>	GUI class libraries for Visual C++ 2005	<b>Sapera\Lib\Win32\VS2005</b> <b>Sapera\Lib\Win64\VS2005</b>
<b>SapClassBasic.lib</b>	Basic class library for Borland C++ Builder XE1 to XE5	<b>Sapera\Lib\Win32\Borland</b>
<b>SapClassBasicXX.dll</b>	Basic class DLL	<windir>\System32
<b>SapClassGuiXX.dll</b>	GUI class DLLs for Visual C++ 2005	<windir>\System32
<b>SapClassGuiXX.NET_2008.dll</b>	GUI class DLLs for Visual C++ 2008	<windir>\System32
<b>SapClassGuiXX.NET_2010.dll</b>	GUI class DLLs for Visual C++ 2010	<windir>\System32
<b>SapClassGuiXX.NET_2012.dll</b>	GUI class DLLs for Visual C++ 2012	<windir>\System32
<b>SapClassBasicXX_b.dll</b>	Basic class DLL for Borland C++ Builder XE1 to XE5	<windir>\System32

# Sapera LT ++ - Creating an Application

The following sections describe how to create a Sapera LT ++ application in Visual C++ 2005/2008/2010/2012/2013 and Borland C++ Builder XE1 to XE5.

## Visual Studio 2005/2008/2010/2012/2013

Follow the steps below to compile and link an application that uses the *Basic Classes*:

- Include **SapClassBasic.h** in the program source code (it includes all other required headers)
- Add **\$(SAPERADIR)\Classes\Basic** and **\$(SAPERADIR)\Include** in *Project | Properties | C/C++ | General | Additional Include Directories*
- If you are building a 32-bit application, insert  
Insert **\$(SAPERADIR)\Lib\Win32\SapClassBasic.lib** in *Project | Add Existing Item ...*
- If you are building a 64-bit application, insert  
Insert **\$(SAPERADIR)\Lib\Win64\SapClassBasic.lib** in *Project | Add Existing Item ...*
- In *Project | Properties | C/C++ | Code Generation | Runtime Library*, choose the option *Multi-threaded DLL* (in release mode) or *Multi-threaded Debug DLL* (in debug mode)

Follow the additional steps below to compile and link an application that uses the *GUI Classes*:

- Include **SapClassGui.h** in the program source code (it includes all other required headers)
- Add **\$(SAPERADIR)\Classes\Gui** in *Project | Properties | C/C++ | General | Additional Include Directories*
- If you are building a 32-bit application, insert **\$(SAPERADIR)\Lib\Win32\VS2005\SapClassGui.lib** (or **VS2008/VS2010/VS2012/VS2013**) and **SapClassGUID.lib** in *Project | Add Existing Item ...*
- If you are building a 64-bit application, insert **\$(SAPERADIR)\Lib\Win64\VS2005\SapClassGui.lib** (or **VS2008/VS2010/VS2012/VS2013**) and **SapClassGUID.lib** in *Project | Add Existing Item ...*
- In *Project | Properties | General*, select *Not Set* for *Character Set*
- In *Project | Properties | General* for **SapClassGui.lib**, select *Excluded From Build* for *Debug*
- In *Project | Properties | General* for **SapClassGUID.lib**, select *Excluded From Build* for *Release*

If you also want to modify the source code for the *GUI Classes* and recompile the associated DLL in Debug or Release mode, the **SapClassGui** project is available from the **SapDemos\_2005.sln**, **SapDemos\_2008.sln**, **SapDemos\_2010.sln**, **SapDemos\_2012.sln**, and **SapDemos\_2013.sln** solution files in the **Sapera\Demos\Classes\Vc** directory.

## Updating Existing Visual Studio Projects

Here is a generic procedure for updating existing projects from an older version of Visual Studio to a newer version:

- Open the newer version OF Visual Studio
- Open the existing solution (or workspace) file with the project(s) to convert from the older version
- Follow the instructions for converting the old projects to the new project format
- Review the warnings (if any) listed in the conversion report, you may need to make changes to project properties as a result
- If the projects only target 32-bit Windows (Win32), you will need to add 64-bit targets (x64) if necessary
- Compile the converted projects, you may need to fix compiler/linker errors and warnings
- If a project uses the SapClassGui/SapClassGuid libraries, you must first recompile these
- If a project uses the SapClassGui/SapClassGuid libraries in its file list, delete these entries from the file list, and insert the libraries you just recompiled
- If a project uses the SapClassGui/SapClassGuid libraries in the linker options, check that the file paths correspond to the libraries you just recompiled

Note that this update procedure is more appropriate for versions of Visual Studio which are closer to one another. For versions which are completely different (for example, Visual Studio 6 to Visual Studio 2010), it is preferable to rewrite the projects from scratch.

## Borland C++ Builder XE1 to XE5

Follow the steps below to compile and link a 32-bit application that uses the *Basic Classes*:

- When creating an application, verify that the multi-threaded runtime option is enabled.
- Include **SapClassBasic.h** in the program source code (it includes all other required headers)
- Add **\$(SAPERADIR)\Classes\Basic** and **\$(SAPERADIR)\Include** in *Project | Options... | C++ Compiler | Paths and Defines | Include search path*.
- Insert **\$(SAPERADIR)\Lib\Win32\Borland\SapClassBasic.lib** in *Project | Add to Project ...*

The *GUI Classes* are not supported under C++ Builder XE.

## Notes on Using the Sapera LT ++ API

When using the Sapera LT ++ API, you must not have a static instance of a Sapera LT ++ object. Also, you must not allocate or free such an object from the DllMain function.

```
SapBuffer staticBuf;// Wrong !!
SapBuffer *pBuffer;// OK

DllMain()
{
pBuffer = new SapBuffer();// Wrong !!
delete pBuffer;// Wrong !!
}

MyFunction()
{
pBuffer = new SapBuffer();// OK
delete pBuffer;// OK
}
```

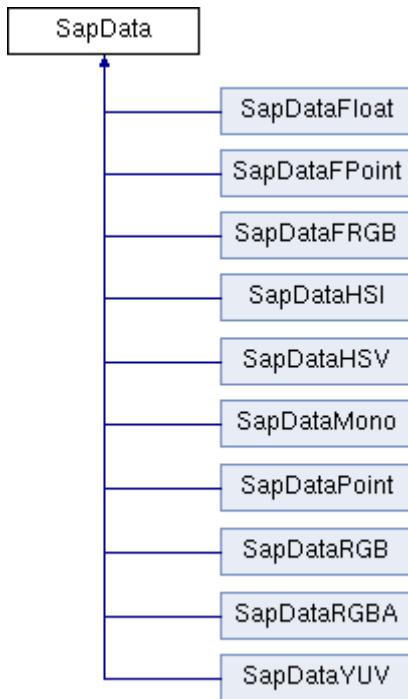
---

## Demos and Examples

Refer to the *Sapera LT User's Manual* for a description of the Sapera LT ++ demos as well as examples available in Sapera .NET.

# Basic Class Reference

## Data Classes



SapData and its derived classes act as wrappers for low-level Sapera LT data types, where each class encapsulates one data element of a specific type. They are used as method arguments or return values in various Sapera LT ++ classes.

### SapData Class

#### Purpose

This is the common base class for all other data classes. Though SapData objects may be directly instantiated, they serve no useful purpose.

**void Clear();**

Clears the data element to black, which almost always corresponds to the numeric value 0 (with a few exceptions, for example, the YUV color format).

**SapFormatType GetType();**

Identifies to which SapDataXxx class the current object is an instance. See the SapManager::GetFormatType method for the list of available types.

#### Demo/Example Usage

Not available

## SapDataFRGB Class

### Purpose

Encapsulates one element supporting Sapera floating-point RGB data types

**SapDataFRGB();**

**SapDataFRGB(float red, float green, float blue);**

Class constructor, where the *red*, *green*, and *blue* arguments specifies an initial value other than black

**float Red();**

Returns the red component of the current value of the data element

**float Green();**

Returns the green component of the current value of the data element

**float Blue();**

Returns the blue component of the current value of the data element

**void Set(float red, float green, float blue);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataHSI Class

### Purpose

Encapsulates one element supporting Sapera HSI data types

**SapDataHSI();**

**SapDataHSI(int h, int s, int i);**

Class constructor, where the *h*, *s*, and *i* arguments specify an initial value other than black

**int H();**

Returns the H component of the current value of the data element

**int S();**

Returns the S component of the current value of the data element

**int I();**

Returns the I component of the current value of the data element

**void Set(int h, int s, int i);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataHSV Class

### Purpose

Encapsulates one element supporting Sapera HSV data types

**SapDataHSV();**

**SapDataHSV(int h, int s, int v);**

Class constructor, where the *h*, *s*, and *v* arguments specify an initial value other than black

**int H();**

Returns the H component of the current value of the data element

**int S();**

Returns the S component of the current value of the data element

**int V();**

Returns the V component of the current value of the data element

**void Set(int h, int s, int v);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataFloat Class

### Purpose

Encapsulates one element supporting Sapera floating-point data types

**SapDataFloat();**

**SapDataFloat(float *flt*);**

Class constructor, where the *flt* argument specifies an initial value other than black

**int Float();**

Returns the current value of the data element

**void Set(float *flt*);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataFPoint Class

### Purpose

Encapsulates one element supporting Sapera data types representing floating-point (x, y) coordinate pairs

**SapDataFPoint();**

**SapDataFPoint(float *x*, float *y*);**

Class constructor, where the *x* and *y* arguments specify an initial value other than (0.0, 0.0)

**float X();**

Returns the X component of the current value of the data element

**float Y();**

Returns the Y component of the current value of the data element

**void Set(float *x*, float *y*);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataMono Class

### Purpose

Encapsulates one element supporting Sapera monochrome data types (excluding 64-bit)

**SapDataMono();**

**SapDataMono(int *mono*);**

Class constructor, where the *mono* argument specifies an initial value other than black

**int Mono();**

Returns the current value of the data element

**void Set(int *mono*);**

Specifies a new value for the data element

### Demo/Example Usage

Example Common Utilities

## SapDataPoint Class

### Purpose

Encapsulates one element supporting Sapera data types representing integer (x, y) coordinate pairs

**SapDataPoint();**

**SapDataPoint(int x, int y);**

Class constructor, where the *x* and *y* arguments specify an initial value other than (0, 0)

**int X();**

Returns the X component of the current value of the data element

**int Y();**

Returns the Y component of the current value of the data element

**void Set(int x, int y);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataRGB Class

### Purpose

Encapsulates one element supporting Sapera RGB data types

**SapDataRGB();**

**SapDataRGB(int red, int green, int blue);**

Class constructor, where the *red*, *green*, and *blue* arguments specify an initial value other than black

**int Red();**

Returns the red component of the current value of the data element

**int Green();**

Returns the green component of the current value of the data element

**int Blue();**

Returns the blue component of the current value of the data element

**void Set(int red, int green, int blue);**

Specifies a new value for the data element

### Demo/Example Usage

Example Common Utilities

## SapDataRGBA Class

### Purpose

Encapsulates one element supporting Sapera RGB with alpha channel data types

**SapDataRGBA(); SapDataRGBA(int red, int green, int blue, int alpha);**

Class constructor, where the *red*, *green*, *blue* and *alpha* arguments specify an initial value other than black

**int Red();**

Returns the red component of the current value of the data element

**int Green();**

Returns the green component of the current value of the data element

**int Blue();**

Returns the blue component of the current value of the data element

**int Alpha();**

Returns the alpha component of the current value of the data element

**void Set(int red, int green, int blue, int alpha);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

## SapDataYUV Class

### Purpose

Encapsulates one element supporting Sapera YUV data types

**SapDataYUV();**

**SapDataYUV(int y, int u, int v);**

Class constructor, where the *y*, *u*, and *v* arguments specify an initial value other than black

**int Y();**

Returns the Y component of the current value of the data element

**int U();**

Returns the U component of the current value of the data element

**int V();**

Returns the V component of the current value of the data element

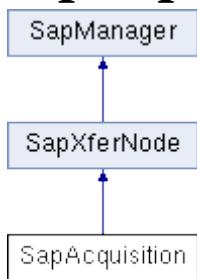
**void Set(int y, int u, int v);**

Specifies a new value for the data element

### Demo/Example Usage

Not available

# SapAcquisition



The SapAcquisition Class includes the functionality to manipulate an acquisition resource. It is used as a source transfer node to allow data transfers from an acquisition resource to another transfer node, such as a buffer.

```
#include <SapClassBasic.h>
```

**Note:** Genie cameras are not supported by this class. The SapAcqDevice class must be used in such cases.

## SapAcquisition Class Members

### Construction

[SapAcquisition](#)

Class constructor

[Create](#)

Allocates the low-level Sapera resources

[Destroy](#)

Releases the low-level Sapera resources

### Attributes

[GetConfigFile](#)

Gets/sets the name of the acquisition configuration file (CCF)

[SetConfigFile](#)

Gets a text description of the acquisition resource

[GetLabel](#)

Gets/sets the combination of registered acquisition event types

[SetEventType](#)

[SetCallbackInfo](#)

Sets the application callback method for acquisition events and the associated context

[GetCallback](#)

Gets the current application callback method for acquisition events

[GetContext](#)

Gets the application context associated with acquisition events

[GetCamSel](#)

Gets/sets the current camera selector value

[SetCamSel](#)

Gets the current LUT enable value

[IsLutEnabled](#)

Checks if the acquisition lookup table may be enabled/disabled

[GetFlipMode](#)

Gets/sets the flipping (that is, mirroring) mode for acquired images

[SetFlipMode](#)

[GetNumPlanarInputs](#)

Gets the number of cameras used for acquiring into vertical planar buffers

[GetPlanarInputs](#)

Gets/sets the current configuration for acquiring into vertical planar buffers

[SetPlanarInputs](#)

[IsFlatFieldAvailable](#)

Gets availability of hardware-based flat-field correction

[IsColorConversionAvailable](#)

Gets availability of hardware-based color conversion

[IsImageFilterAvailable](#)

Gets availability of hardware-based image filter

[IsImageFilterEnabled](#)

Gets the current image filter enable value

[IsTimeStampAvailable](#)

Gets availability of hardware-based timestamp

[IsWhiteBalanceAvailable](#)

Gets availability of hardware-based gains for white balance control

[GetTimeStampBase](#)

Gets/sets the timestamp base unit

<a href="#"><u>SetTimeStampBase</u></a>	
<a href="#"><u>GetSerialPortName</u></a>	Gets the name of the serial port attached to the current acquisition device
<b>Operations</b>	
<a href="#"><u>SaveParameters</u></a>	Saves the acquisition parameters to an acquisition configuration file (CCF)
<a href="#"><u>GetNumLut</u></a>	Gets the number of available acquisition look-up tables
<a href="#"><u>GetLut</u></a>	Gets an acquisition lookup table
<a href="#"><u>ApplyLut</u></a>	Reprograms an acquisition lookup table
<a href="#"><u>EnableLut</u></a>	Enables/disables the acquisition lookup tables
<a href="#"><u>EnableImageFilter</u></a>	Enables/disables the acquisition image filter
<a href="#"><u>GetImageFilter</u></a>	Gets/sets the values of the acquisition image filter
<a href="#"><u>SetImageFilter</u></a>	
<a href="#"><u>GetImageFilterKernelSize</u></a>	Gets the image filter kernel size
<a href="#"><u>LoadImageFilter</u></a>	Loads a hardware-based image filter kernel from file
<a href="#"><u>SaveImageFilter</u></a>	Saves a hardware-based image filter kernel to file
<a href="#"><u>IsSignalStatusAvailable</u></a>	Checks for availability of the status of input acquisition signals
<a href="#"><u>GetSignalStatus</u></a>	Gets the current status of input acquisition signals
<a href="#"><u>SoftwareTrigger</u></a>	Simulates a trigger to the acquisition device
<a href="#"><u>IsCapabilityValid</u></a>	Checks for the availability of a low-level Sapera C library capability
<a href="#"><u>IsParameterValid</u></a>	Checks for the availability of a low-level Sapera C library parameter
<a href="#"><u>GetCapability</u></a>	Gets the value of a low-level Sapera C library capability
<a href="#"><u>GetParameter</u></a>	Gets/sets the value of a low-level Sapera C library parameter
<a href="#"><u>SetParameter</u></a>	
<a href="#"><u>CustomCommand</u></a>	Issues a low-level custom command specific to the acquisition hardware
<a href="#"><u>ResetTimeStamp</u></a>	Resets the acquisition hardware timestamp counter to zero
<a href="#"><u>RegisterCallback</u></a>	Registers a callback function for the event associated with a specified name or index
<a href="#"><u>UnregisterCallback</u></a>	Unregisters a callback function on the event associated with a specified name or index

## SapAcquisition Member Functions

The following are members of the SapAcquisition Class.

---

### SapAcquisition::SapAcquisition

```

SapAcquisition(
    SapLocation loc = SapLocation::ServerSystem
);
SapAcquisition(
    SapLocation loc, const char *configFile,
    SapAcquisition::EventType eventType = SapAcquisition::EventNone,
    SapAcqCallback pCallback = NULL,
    void *pContext = NULL
);
SapAcquisition(
    SapLocation loc,
    const char *camfile,
    const char *vicfile,
    SapAcquisition::EventType eventType = SapAcquisition::EventNone,
    SapAcqCallback pCallback = NULL, void *pContext = NULL
);
SapAcquisition
(
    const SapAcquisition &acq,
```

```

SapAcquisition::EventType eventType,
SapAcqCallback pCallback,
void *pContext = NULL
);

```

## Parameters

<i>loc</i>	SapLocation object specifying the server where the acquisition resource is located and the index of the acquisition resource on this server.
<i>configFile</i>	Name of the acquisition configuration file (CCF) that describes all camera and frame grabber-related acquisition parameters. Use one of the standard CCF files provided with Sapera or create one using the CamExpert utility.
<i>camfile</i>	Name of the configuration file (CCA) that describes all camera related acquisition parameters (obsolete)
<i>vicfile</i>	Name of the configuration file (CVI) that describes all frame grabber-related acquisition parameters (obsolete)
<i>eventType</i>	Acquisition events for which the application callback function will be called. One or more of the following values may be combined together using a bitwise OR operation:  SapAcquisition::EventNone                          No events SapAcquisition::EventStartOfFrame                Start of frame SapAcquisition::EventStartOfField                Start of any field (odd or even) SapAcquisition::EventStartOfOdd                 Start of odd field SapAcquisition::EventStartOfEven                Start of even field SapAcquisition::EventEndOfFrame                End of frame SapAcquisition::EventEndOfField                End of any field (odd or even) SapAcquisition::EventEndOfOdd                 End of odd field SapAcquisition::EventEndOfEven                End of even field SapAcquisition::EventEndOfLine                After a specific line number  eventType = SapAcquisition::EventEndOfLine   <i>lineNum</i> After a specific number of lines (linescan cameras only) eventType = SapAcquisition::EventEndOfNLines   <i>numLines</i>  SapAcquisition::EventVirtualFrame                Equivalent to EventStartOfFrame for linescan cameras SapAcquisition::EventExternalTrigger            Received an external trigger that will then acquire at least one image. The maximum callback rate cannot be greater than the acquisition video frame rate.  SapAcquisition::EventVerticalSync                Vertical sync detected, even if not acquiring SapAcquisition::EventNoHSync                    Timeout due to a missing horizontal sync during live acquisition. You can set the timeout value by calling the SetParameter method for CORACQ_PRM_HSYNC_TIMEOUT. The event is only generated once, unless a new SapTransfer::Grab or SapTransfer::Snap command is issued or a new horizontal sync is detected.  SapAcquisition::EventNoVSync                    Timeout due to a missing horizontal sync during live acquisition. You can set the timeout value by calling the SetParameter method for CORACQ_PRM_VSYNC_TIMEOUT. The event is only generated once, unless a new SapTransfer::Grab or SapTransfer::Snap command is issued or a new horizontal sync is detected  SapAcquisition::EventNoPixelClk                No pixel clock detected. Generated only once, unless a new SapTransfer::Snap/Grab command is issued or the pixel clock is detected again and then lost.  SapAcquisition::EventPixelClk                    Pixel clock detected. Generated only once, unless a new SapTransfer::Snap/Grab command is issued or the pixel clock is lost again and then detected.  SapAcquisition::EventFrameLost                Lost a frame during live acquisition. This usually occurs

	SapAcquisition::EventDataOverflow	if there is not enough bandwidth to transfer images to host memory.
	SapAcquisition::EventExternalTriggerIgnored	Data overflow occurred during live acquisition. This usually occurs if the acquisition device cannot sustain the data rate of the incoming images.
	SapAcquisition::EventExternalTriggerTooSlow	Dropped an external trigger event. This usually occurs when the external trigger rate is faster than the acquisition frame rate.
	SapAcquisition::EventHsyncLock	The detected external line trigger rate is too slow for the hardware to process. This can usually occur when using the shaft encoder multiplier.
	SapAcquisition::EventHsyncUnlock	Detected a horizontal sync unlock to lock condition.
	SapAcquisition::EventVerticalTimeout	Detected a horizontal sync lock to unlock condition.
	SapAcquisition::EventLinkError	Detected a vertical timeout. You can set the timeout value by calling the SetParameter method for CORACQ_PRM_VERTICAL_TIMEOUT_DELAY.
	SapAcquisition::EventLineTriggerTooFast	Detected an error on the link between the camera and the frame grabber (for HSLink cameras only). The exact error condition may be one of the following: 8-bit/10-bit encoding, packet header error, CRC error, bad revision, or lost idle lock.
	SapAcquisition::EventShaftEncoderReverseCountOverflow	The detected line trigger rate is too fast for the hardware to process. This can occur when using the shaft encoder multiplier.
	SapAcquisition::EventLinkLock	Detected an overflow of the shaft encoder reverse counter.
	SapAcquisition::EventLinkUnlock	Detected all required lanes are locked (for HSLink and CLHS cameras only).
		Detected at least one of the required lanes lost the link lock (for HSLink and CLHS cameras only)
<i>pCallback</i>	Application callback function to be called each time one of the events specified above is received. If <i>eventType</i> is EventNone, this parameter is ignored.	
	The callback function must be declared as: void MyCallback(SapAcqCallbackInfo * <i>pInfo</i> );	
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL or <i>eventType</i> is EventNone, this parameter is ignored.	
<i>acq</i>	Existing acquisition object	

### Remarks

The SapAcquisition constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

The constructor with the *camFile* and *vicFile* arguments is now obsolete. However, you may use it for backward compatibility with older versions of Sapera LT in which CCA and CVI files were used instead of CCF files. The constructor with an acquisition object, event type, callback function and context is useful in one particular case. If you use the GUI class CAcqConfigDlg to load a configuration file, the resulting acquisition object is not configured to handle events. You can then use this constructor to complete the configuration.

The SapAcquisition object is used only for storing the acquisition resource parameters. To acquire data, use the SapTransfer Class (or one of its derived classes) and pass the SapAcquisition object as a parameter for the constructor. SapTransfer then handles the actual data transfer. You can also use the SapAcqToBuf specialized transfer class to simplify this task.

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example,

---

## SapAcquisition::ApplyLut

```
BOOL ApplyLut(BOOL enable = TRUE);  
BOOL ApplyLut(BOOL enable, int lutIndex);
```

### Parameters

*enable* Enable or disable the lookup table after reprogramming  
*lutIndex* Look-up table index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Reprograms an acquisition lookup table. The first version of this method reprograms the first (and often the only) LUT, whereas the second version allows a LUT index to be specified. Valid values for this index are from 0 to the value returned by the GetNumLut method, minus 1.

After getting the current LUT using the GetLut method, use the methods in the SapLut Class to manipulate it. Then use ApplyLut to apply the changes. You need to enable the LUT in order to affect acquired images. Note that some acquisition devices do not support enabling or disabling the LUT.

### Demo/Example Usage

Grab LUT Example

---

## SapAcquisition::CanEnableLut

```
BOOL CanEnableLut();
```

### Remarks

Checks if the acquisition lookup table may be enabled/disabled. The initial value for this attribute is FALSE. It is then set according to the current the acquisition device capability when calling the Create method.

### Demo/Example Usage

Not available

---

## SapAcquisition::Create

```
BOOL Create();  
bool Create();
```

### Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

### Remarks

Creates all the low-level Sapera resources needed by the acquisition object. Always call this method before SapTransfer::Create.

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

## SapAcquisition::CustomCommand

```
BOOL CustomCommand(int command, void *inData, int inDataSize, void *outData, int outDataSize);
```

### Parameters

*Command* Low-level command ID  
*inData* Memory area with input data  
*inDataSize* Number of bytes of input data  
*outData* Memory area to receive output data  
*outDataSize* Maximum number of bytes of output data

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Provides a way to directly call custom commands specific to the acquisition hardware.

You will rarely need to use this method since the functionality is usually customer or OEM specific.

#### Demo/Example Usage

Not available

---

### SapAcquisition::Destroy

**BOOL Destroy();**

#### Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

#### Remarks

Destroys all the low-level Sapera resources needed by the acquisition object. Always call this method after SapTransfer::Destroy.

#### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### SapAcquisition::EnableImageFilter

**BOOL EnableImageFilter(BOOL enable = TRUE);**

#### Parameters

*Enable*      TRUE to enable the acquisition lookup table, FALSE to disable it

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Enables or disables the acquisition image filter. When the image filter is disabled, it does not affect acquired images. However, its contents are not lost, so they may be used again without reprogramming the acquisition hardware. Note that some acquisition devices do not support this feature.

#### Demo/Example Usage

Not available

---

### SapAcquisition::EnableLut

**BOOL EnableLut(BOOL enable = TRUE);**

#### Parameters

*Enable*      TRUE to enable the acquisition lookup table, FALSE to disable it

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Enables or disables the acquisition lookup table. When the LUT is disabled, it does not affect acquired images. However, its contents are not lost, so they may be used again without reprogramming the acquisition hardware. Note that some acquisition devices do not support this feature.

#### Demo/Example Usage

Grab LUT Example

---

### SapAcquisition::GetCallback

**SapAcqCallback GetCallback();**

#### Remarks

Gets the current application callback method for acquisition events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapAcquisition constructor for more details.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetCamSel, SapAcquisition::SetCamSel

```
int GetCamSel();
BOOL SetCamSel(int camSel);
```

#### Remarks

Specifies the zero-based index of the camera input from which the acquisition device grabs images. The maximum value allowed depends on the acquisition hardware and the current data format. The initial value for this attribute is 0. It is then set according to the current acquisition device value when calling the Create method.

You cannot call SetCamSel before the Create method or during live acquisition, that is, when the SapTransfer::IsGrabbing method returns TRUE.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetCapability

```
BOOL GetCapability(int cap, void *pValue);
```

#### Parameters

*Cap* Low-level Sapera C library capability to read

*pValue* Pointer to capability value to read back

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

This method allows direct read access to low-level Sapera C library capabilities for the acquisition module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

Note that this method is rarely needed. The SapAcquisition class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

Grab Demo

---

### SapAcquisition::GetConfigFile, SapAcquisition::SetConfigFile

```
const char *GetConfigFile();
BOOL SetConfigFile(const char *configFile);
```

#### Remarks

Gets/sets the name of the acquisition configuration file (CCF).

You normally set the initial value for this attribute in the SapAcquisition constructor. If you use the default constructor, then this value is NULL.

You can only call SetConfigFile before the Create method.

#### Demo/Example Usage

Grab Demo

---

### SapAcquisition::GetContext

```
void *GetContext();
```

#### Remarks

Gets the application context associated with acquisition events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapAcquisition constructor for more details.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetEventType, SapAcquisition::SetEventType

```
SapAcquisition::EventType GetEventType();
```

```
BOOL SetEventType(SapAcquisition::EventType eventType);
```

#### Remarks

Gets/sets the combination of registered acquisition event types. The initial value for this attribute is EventNone, unless you specify another value in the constructor.

You can only call SetEventType before the Create method. See the SapAcquisition constructor for possible values for *eventType*.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetFlipMode, SapAcquisition::SetFlipMode

```
SapAcquisition::FlipMode GetFlipMode();  
BOOL SetFlipMode(SapAcquisition::FlipMode flipMode);
```

#### Parameters

<i>flipMode</i>	SapAcquisition::FlipNone	No flipping
	SapAcquisition::FlipHorizontal	Acquired images are flipped horizontally
	SapAcquisition::FlipVertical	Acquired images are flipped vertically

#### Remarks

Gets/sets the flipping (that is, mirroring) mode for acquired images. The initial value for this attribute is FlipNone.

You can only call SetFlipMode after the Create method.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetImageFilter, SapAcquisition::SetImageFilter

```
BOOL GetImageFilter(int filterIndex, SapBuffer *pKernel);  
BOOL SetImageFilter(int filterIndex, SapBuffer *pKernel);
```

#### Parameters

<i>filterIndex</i>	Kernel filter index.
<i>pKernel</i>	Pointer to SapBuffer object containing the kernel values.

#### Remarks

Gets/sets the image filter kernel values. With an appropriate choice of kernel values, the image filter can perform such operations as smoothing, edge or peak enhancement, or position shifting on the image.

Use the SapAcquisition::IsImageFilterAvailable to check if the acquisition device supports hardware-based image filters.

The image filter values are specified in a SapBuffer object with SapFormatInt32 (signed values). The size of the image filter is retrieved using the SapAcquisition::GetImageFilterKernelSize function. The values can be accessed using the SapBuffer::ReadElement and SapBuffer::WriteElement functions.

Use the SetImageFilter function to update the hardware image filter kernel with the values contained in the specified SapBuffer object. When the kernel is applied to the image, each pixel is multiplied by the corresponding value in the kernel matrix (divided by the divisor), and the center pixel is replaced by the sum of the resulting pixel values in the matrix.

Note: The actual weight of a pixel is the value in the buffer divided by the divisor. For example, if the divisor is 16384, a value of 24576 in the kernel provides a weight of 1.5 (that is, 24576/16384). Thus for a 3x3 low pass filter with all kernel filter elements with an effective weight of 1, each kernel entry in the buffer would have a value of (1/9) \* CORACQ\_CAP\_IMAGE\_FILTER\_DIVISOR.

You can only call SetImageFilter after the Create method.

Note, currently available hardware only supports a single filter (*filterIndex* = 0).

#### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

### SapAcquisition::GetImageFilterKernelSize

```
BOOL GetImageFilterKernelSize(int filterIndex, ImageFilterKernelSize *pKernelSize);
```

*filterIndex* Kernel filter index.

*pKernelSize* Kernel size. Possible values are:

- SapAcquisition::ImageFilterSize1x1
- SapAcquisition::ImageFilterSize2x2
- SapAcquisition::ImageFilterSize3x3
- SapAcquisition::ImageFilterSize4x4
- SapAcquisition::ImageFilterSize5x5
- SapAcquisition::ImageFilterSize6x6
- SapAcquisition::ImageFilterSize7x7

### Remarks

Gets acquisition hardware image filter kernel size.

Note, currently available hardware only supports a single filter (*filterIndex* = 0).

### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

## SapAcquisition::GetLabel

```
econst char* GetLabel();
```

### Remarks

Gets a text description of the acquisition resource. This attribute is initially set to an empty string. After a successful call to the Create method, it is composed of the name of the server where the acquisition resource is located and the name of this resource: *ServerName* [*ResourceName*].

Example: "Xcelera-CL\_PX4\_1 [ CameraLink Full Mono #1 ]"

After the label is initialized, its value never changes again.

### Demo/Example Usage

Not available

---

## SapAcquisition::GetLut

```
SapLut *GetLut(int lutIndex = 0);
```

### Remarks

Gets an acquisition lookup table. All available LUTs on the acquisition device are automatically created and initialized when calling the Create method. You can manipulate the LUT through the methods in the SapLut Class, and reprogram it using the ApplyLut method.

Valid values for the *lutIndex* argument are from 0 to the value returned by the GetNumLut method, minus 1.

GetLut returns NULL if the current acquisition device does not support lookup tables.

### Demo/Example Usage

Grab LUT Demo

---

## SapAcquisition::GetNumLut

```
int GetNumLut();
```

### Remarks

Gets the number of available acquisition look-up tables, where a value of 0 means that the current acquisition device has no LUTs. The returned value is only meaningful after you call the Create method.

### Demo/Example Usage

Grab LUT Demo

---

## SapAcquisition::GetNumPlanarInputs

```
int GetNumPlanarInputs();
```

### Remarks

Gets the number of cameras used for acquiring into vertical planar buffers, where a value of 1 means that

planar mode is disabled. All cameras must be synchronized together. The returned value is only meaningful after you call the Create method.

### Demo/Example Usage

Not available

---

## SapAcquisition::GetParameter, SapAcquisition::SetParameter

```
BOOL GetParameter(int param, void *pValue);
BOOL SetParameter(int param, int value, BOOL updateNow = TRUE);
BOOL SetParameter(int param, void *pValue, BOOL updateNow = TRUE);
```

### Parameters

*param* Low-level Sapera C library parameter to read or write  
*paramValue* Pointer to parameter value to read back or to write  
*value* New parameter value to write  
*updateNow* Allows delayed updating of acquisition parameters

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the acquisition module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value that is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

By default, *updateNow* is TRUE, therefore calling SetParameter programs the acquisition hardware with the new value immediately. However, some parameters should not be set individually, as this may result in inconsistencies and error conditions in the acquisition resource.

If *updateNow* is FALSE, new parameter values are accumulated internally. The next time SetParameter is called with *updateNow* set to TRUE, all the new values are sent in one operation to the acquisition hardware, thus avoiding the problems just described.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported by the SapAcquisition Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Bayer Demo, FlatField Demo, Sequential Grab Demo

---

## SapAcquisition::GetPlanarInputs, SapAcquisition::SetPlanarInputs

```
BOOL GetPlanarInputs(BOOL *pCamEnable);
BOOL SetPlanarInputs(BOOL *pCamEnable, int numCameras);
```

### Parameters

*pCamEnable* Camera configuration array, must have at least 32 entries  
*numCameras* Number of cameras to configure for planar acquisition

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets/sets the current configuration for synchronous acquisition into vertical planar buffers, where all cameras are synchronized together.

Individual entries in the *pCamEnable* array are set to TRUE if the corresponding camera is enabled for planar acquisition; otherwise, they are set to FALSE. The entry at index 0 in *pCamEnable* corresponds to the first camera, the entry at index 1 corresponds to the second camera, and so on. If planar mode is disabled, then only the entry at index 0 is set.

You can only call GetPlanarInputs and SetPlanarInputs after the Create method.

### Demo/Example Usage

Not available

---

### SapAcquisition::GetSerialPortName

BOOL **GetSerialPortName**(char \**serialPortName*);

#### Parameters

*serialPortName* Memory area large enough to receive the text for the serial port name (at least 64 bytes)

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the name of the serial port attached to the current acquisition device.

You can only call GetSerialPortName after the Create method.

#### Demo/Example Usage

Not available

---

### SapAcquisition::GetSignalStatus

BOOL **GetSignalStatus**(SapAcquisition::SignalStatus *signalStatus*, BOOL \**pIsActive*);

BOOL **GetSignalStatus**(SapAcquisition::SignalStatus \**pSignalStatus*,  
SapAcqCallback *pCallback* = NULL, void \**pContext* = NULL);

#### Parameters

*signalStatus* Combination of status signals to inquire. See the IsSignalStatusAvailable method for a list of possible values.

*pIsActive* Set upon return to TRUE if the specified status signals have been detected, FALSE otherwise

*pSignalStatus* Set upon return to the combination of detected status signals. See the IsSignalStatusAvailable method for a list of possible values.

*pCallback* Application callback function to be called each time the combination of detected signal status changes.

The callback function must be declared as:

```
void MyCallback(SapAcqCallbackInfo *pInfo);
```

*pContext* Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reports the status of input signals connected to the acquisition device. Use the first form of GetSignalStatus for a one-time inquiry. Since many signals may be detected at the same time, values are usually combined together using a bitwise OR operation.

The second form allows asynchronous notification of application code whenever the combination of status signals changes. This may happen, for example, when an input cable is accidentally disconnected. First call the method as follows:

```
GetSignalStatus(&currentStatus, MyCallback, &myContext);
```

This first reads the current value of the signal status. An internal mechanism then periodically checks for signal status changes, and notifies the application program using the callback function. You must call GetSignalStatus again with a NULL argument to disable the application callback function:

```
GetSignalStatus(&currentStatus, NULL);
```

#### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

---

### SapAcquisition::GetTimeStampBase, SapAcquisition::SetTimeStampBase

TimeStampBase **GetTimeStampBase**;

BOOL **SetTimeStampBase**(TimeStampBase *timeStampBase*);

#### Parameters

*timeStampBase* SapAcquisition::TimeStamp100NanoSec The time base is in 100 nano seconds.

SapAcquisition::TimeStampFrameValid	The time base is in frame valid signals received.
SapAcquisition::TimeStampLineTrigger	The time base is in external line trigger or shaft encoder pulse (after drop/multiply operation).
SapAcquisition::TimeStampLineValid	The time base is in line valid signals received.
SapAcquisition::TimeStampMicroSec	The time base is in micro seconds.
SapAcquisition::TimeStampMilliSec	The time base is in milli seconds.
SapAcquisition::TimeStampNanoSec	The time base is in nano seconds.
SapAcquisition::TimeStampNone	Time base is not available.
SapAcquisition::TimeStampPixelClock	The time base is in camera pixel clock.
SapAcquisition::TimeStampShaftEncoder	The time base is in external line trigger or shaft encoder pulse (before drop/multiply operation).

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets/sets the acquisition device timestamp base units.

#### Demo/Example Usage

Sequential Grab Demo

### SapAcquisition::IsCapabilityValid

**BOOL IsCapabilityValid(int cap);**

#### Parameters

*cap* Low-level Sapera C library capability to be checked

#### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library capability for the acquisition module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapAcquisition class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

Not available

### SapAcquisition::IsColorConversionAvailable

**BOOL IsColorConversionAvailable();**

#### Remarks

Gets availability of hardware-based color conversion. You can only call IsColorConversionAvailable after the Create method.

#### Demo/Example Usage

ColorConverions Demo, FlatField Demo

### SapAcquisition::IsFlatFieldAvailable

**BOOL IsFlatFieldAvailable();**

#### Remarks

Gets availability of hardware-based flat-field correction. You can only call IsFlatFieldAvailable after the Create method.

## Demo/Example Usage

FlatField Demo

---

### SapAcquisition::IsImageFilterAvailable

**BOOL IsImageFilterAvailable();**

#### Remarks

Gets availability of hardware-based image filter. You can only call IsImageFilterAvailable after the Create method.

#### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

### SapAcquisition::IsImageFilterEnabled

**BOOL IsImageFilterEnabled();**

#### Remarks

Sets the enable state of the hardware acquisition image filter. To check if image filter is supported by the acquisition device use the SapAcquisition::IsImageFilterAvailable.

#### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

### SapAcquisition::IsLutEnabled

**BOOL IsLutEnabled();**

#### Remarks

Gets the current LUT enable value. The initial value for this attribute is FALSE. It is then set according to the current the acquisition device value when calling the Create method.

#### Demo/Example Usage

Not available

---

### SapAcquisition::IsParameterValid

**BOOL IsParameterValid(int param);**

#### Parameters

*param* Low-level Sapera C library parameter to be checked

#### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library parameter for the acquisition module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note tha this method is rarely needed. The SapAcquisition class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

FlatField Demo

---

### SapAcquisition::IsSignalStatusAvailable

**BOOL IsSignalStatusAvailable();**

**BOOL IsSignalStatusAvailable(SapAcquisition::SignalStatus signalStatus);**

#### Parameters

*signalStatus* Status signal to inquire. One or more of the following values may be ORed together.

SapAcquisition::SignalNone	No signal
SapAcquisition::SignalHSyncPresent	Horizontal sync signal (analog video source) or line

SapAcquisition::SignalVSyncPresent	valid (digital video source)
SapAcquisition::SignalPixelClkPresent / SapAcquisition::SignalPixelClk1Present	Vertical sync signal (analog video source) or frame valid (digital video source)
SapAcquisition::SignalPixelClk2Present	Pixel clock signal. For CameraLink devices, this status returns true if a clock signal is detected on the base cable.
SapAcquisition::SignalPixelClk3Present	Pixel clock signal. For CameraLink devices, this status returns true if a clock signal is detected on the medium cable.
SapAcquisition::SignalPixelClkAllPresent	Pixel clock signal. For Camera Link devices, true if all required pixel clock signals have been detected by the acquisition device based on the CameraLink configuration selected.
SapAcquisition::SignalChromaPresent	Color burst signal (valid for NTSC and PAL)
SapAcquisition::SignalHSyncLock	Successful lock to an horizontal sync signal, for an analog video source
SapAcquisition::SignalVSyncLock	Successful lock to a vertical sync signal, for an analog video source
SapAcquisition::SignalPowerPresent	Power is available for a camera. This does not necessarily mean that power is used by the camera, it only indicates that power is available at the camera connector, where it might be supplied from the board PCI bus or from the board PC power connector. The returned value value is FALSE if the circuit fuse is blown, therefore power cannot be supplied to any connected camera.
SapAcquisition::SignalPoCLActive	Power to the camera is present on the Camera Link cable
SapAcquisition::SignalPixelLinkLock	Lane lock signal. For HSLink and CLHS devices, true if all required lane lock signals have been detected by the acquisition device based on the HSLink or CLHS configuration selected.

### Return Value

Returns TRUE if the acquisition device can detect the specified status signals, FALSE otherwise

### Remarks

Reports the availability of the status of input signals connected to the acquisition device. Use the first form of IsSignalStatusAvailable to inquire about all input signals. Use the second form to narrow the inquiry down to specific signals only.

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

## SapAcquisition::IsTimeStampAvailable

**BOOL IsTimeStampAvailable();**

### Return Value

Returns TRUE if the acquisition device has a hardware-based timestamp, FALSE otherwise

### Remarks

Reports the availability of a hardware timestamp on the acquisition device. In general, a hardware timestamp is more accurate than one associated with a buffer transfer event to the host. The timestamp is retrieved from a buffer using SapBuffer::GetCounterStamp.

### Demo/Example Usage

Sequential Grab Demo

## SapAcquisition::IsWhiteBalanceAvailable

```
BOOL IsWhiteBalanceAvailable();
```

#### Remarks

Gets availability of hardware-based gain values for white balance control. You can only call IsWhiteBalanceAvailable after the Create method.

#### Demo/Example Usage

ColorConverions Demo, FlatField Demo

---

### SapAcquisition::LoadImageFilter

```
BOOL LoadImageFilter(UINT32 filterIndex, const char *file);
```

#### Parameters

*filterIndex* Filter index into which to load the kernel.

*file* Image filter kernel file to load.

#### Remarks

Loads a image filter kernel from file for hardware-based image filtering. The kernel file format uses the .crc extension. Use SapAcquisition::SaveImageFilter to save kernels to file.

#### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

### SapAcquisition::RegisterCallback

```
BOOL RegisterCallback(EventType eventType, SapAcqCallback callback, void *context = NULL);
```

#### Parameters

*eventType* Event type. See the SapAcquisition constructor for a list a possible values.

*callback* Address of a user callback function of the following form:

```
void MyCallback(SapAcquisitionCallbackInfo* pInfo)
{}
```

*context* Pointer to a user storage (that is, variable, structure, buffer, etc). Can be NULL.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Registers an event by associating a callback function for the specified type. When the event occurs in the acquisition device, this callback function is called. It provides information on the corresponding event using a SapAcqCallbackInfo object. Refer to this class for more details.

The context pointer is also returned by the callback function, allowing for the exchange application specific information.

#### Example

```
void MyCallback(SapAcqCallbackInfo* pInfo)
{
    // Access information using functions of SapAcqCallbackInfo class
    // ...
}

main()
{
    // ...
    acq.RegisterCallback("FeatureValueChanged", MyCallback, NULL);
    // ...
    acq.UnregisterCallback("FeatureValueChanged");
    // ...
}
```

#### Demo/Example Usage

Grab Console example

---

### SapAcquisition::ResetTimeStamp

```
BOOL ResetTimeStamp();
```

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Resets the acquisition hardware timestamp counter to zero.

#### Demo/Example Usage

Not available

---

### SapAcquisition::SaveImageFilter

**BOOL SaveImageFilter(UINT32 filterIndex, const char \*file);**

#### Parameters

*filterIndex* Filter index into which to load the kernel.

*file* Image filter kernel file to load.

#### Remarks

Saves a hardware-based image filter kernel to file. The kernel file format uses the .crci extension. Use SapAcquisition::LoadImageFilter to load previously saved kernels.

#### Demo/Example Usage

See SapClassGui::CimageFilterEditorDlg

---

### SapAcquisition::SaveParameters

**BOOL SaveParameters(const char \*configFile);**

#### Parameters

*configFile* Name of the acquisition configuration file (CCF) for saving camera and frame grabber related acquisition parameters

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Saves the current values of acquisition module parameters to the specified file.

#### Demo/Example Usage

Not available

---

### SapAcquisition::SetCallbackInfo

**BOOL SetCallbackInfo(SapAcqCallback pCallback, void \*pContext = NULL);**

#### Remarks

Sets the application callback method for acquisition events and the associated context.

You can only call SetCallbackInfo before the Create method. See the SapAcquisition constructor for more details.

#### Demo/Example Usage

Not available

---

### SapAcquisition::SoftwareTrigger

**BOOL SoftwareTrigger(SapAcquisition::SoftwareTriggerType triggerType);**

#### Parameters

*triggerType* Trigger type may be one of the following values

SapAcquisition::SoftwareTriggerExtI External trigger

SapAcquisition::SoftwareTriggerExtFrame External frame trigger

SapAcquisition::SoftwareTriggerExtLine External line trigger

#### Return Value

Returns TRUE if successful, FALSE otherwise.

#### Remarks

Simulates a trigger to the acquisition device. Use SoftwareTrigger for testing purposes when the actual hardware trigger is not available.

Note that in order for this feature to work, external trigger must be enabled. This can be done either through CamExpert or by calling the SetParameter method for the CORACQ\_PRM\_EXT\_TRIGGER\_ENABLE parameter.

Also, this feature may not be implemented on the current acquisition device. To find out if it is, call the GetCapability method for the CORACQ\_CAP\_SOFTWARE\_TRIGGER capability.

### Demo/Example Usage

Not available

---

## SapAcquisition::UnregisterCallback

```
BOOL UnregisterCallback(const char* eventName);
BOOL UnregisterCallback(int eventIndex);
```

### Parameters

*eventName* Event name. See the acquisition device User's Manual for the list of supported events.

*eventIndex* Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Unregisters a callback function on the event associated with a specified name or index. Use this function in a loop to unregister all the callback functions previously registered.

### Example

```
// Unregisters all the callback functions
//
UINT32 eventCount, eventIndex;
acq.GetEventCount(&eventCount);
for (eventIndex = 0; eventIndex < eventCount; eventIndex++)
{
    BOOL isRegistered;
    acq.IsCallbackRegistered(eventIndex, &isRegistered);
    if (isRegistered)
    {
        acq.UnregisterCallback(eventIndex);
    }
}
```

### Demo/Example Usage

Grab Console Example

# SapAcqCallbackInfo

The SapAcqCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapAcquisition Class.

```
#include <SapClassBasic.h>
```

## SapAcqCallbackInfo Class Members

### Construction

[SapAcqCallbackInfo](#) Class constructor

### Attributes

<u><a href="#">GetAcquisition</a></u>	Gets the SapAcquisition object associated with acquisition events or signal status reporting
<u><a href="#">GetContext</a></u>	Gets the application context associated with acquisition events or signal status reporting
<u><a href="#">GetCustomData</a></u>	Gets the data associated with a custom event
<u><a href="#">GetCustomSize</a></u>	Gets the size of the custom data returned by GetCustomData
<u><a href="#">GetEventType</a></u>	Gets the acquisition events that triggered the call to the application callback
<u><a href="#">GetEventCount</a></u>	Gets the current count of acquisition events
<u><a href="#">GetEventInfo</a></u>	Gets the low-level Sapera handle of the event info resource.
<u><a href="#">GetSignalStatus</a></u>	Gets the input signal status that triggered the call to the application callback
<u><a href="#">GetGenericParam0</a></u>	Gets generic parameters supported by some events
<u><a href="#">GetGenericParam1</a></u>	
<u><a href="#">GetGenericParam2</a></u>	
<u><a href="#">GetGenericParam3</a></u>	
<u><a href="#">GetAuxiliaryTimestamp</a></u>	Gets the auxiliary timestamp associated with acquisition events or signal status reporting
<u><a href="#">GetHostTimestamp</a></u>	Gets the host timestamp associated with acquisition events or signal status reporting

## SapAcqCallBackInfo Member Functions

The following are members of the SapAcqCallbackInfo Class.

### SapAcqCallbackInfo::GetAcquisition

SapAcquisition\* **GetAcquisition()**;

### Remarks

Gets the SapAcquisition object context associated with acquisition events or signal status reporting. See the SapAcquisition constructor for more details.

### Demo/Example Usage

Not available

### SapAcqCallbackInfo::GetAuxiliaryTimestamp

BOOL **GetAuxiliaryTimestamp**(UINT64 \*auxTimestamp);

### Parameters

*auxTimestamp* Address of a pointer to receive the auxiliary timestamp

### Remarks

Gets the auxiliary timestamp associated with acquisition events or signal status reporting. When a registered event is raised, the auxiliary timestamp is generated internally by the device (to retrieve the host timestamp generated by the host CPU see the SapAcqCallbackInfo::GetHostTimestamp function).

Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::SapAcqCallbackInfo

```
SapAcqCallbackInfo(  
    SapAcquisition* pAcq,  
    void* pContext,  
    SapAcquisition::EventType eventType,  
    int eventCount  
);  
  
SapAcqCallbackInfo(  
    SapAcquisition* pAcq,  
    void* pContext,  
    SapAcquisition::SignalStatus signalStatus  
);  
  
SapAcqCallbackInfo(  
    SapAcquisition *pAcq,  
    void *pContext,  
    COREVENTINFO eventInfo  
);
```

### Parameters

<i>pAcq</i>	SapAcquisition object that calls the callback function.
<i>pContext</i>	Pointer to the application context.
<i>eventType</i>	Combination of acquisition events. See the SapAcquisition constructor for a list a possible values.
<i>eventCount</i>	Current acquisition event count.
<i>signalStatus</i>	Combination of signal status values. See SapAcquisition::IsSignalStatusAvailable for a list a possible values.
<i>eventInfo</i>	Low-level Sapera handle of the event info resource

### Remarks

SapAcquisition objects create an instance of this class before each call to the acquisition callback method in order to combine all function arguments into one container.

SapAcquisition uses this class for two different purposes. The first case applies to reporting acquisition events. The *pContext* parameter takes the value specified in the SapAcquisition class constructor; *eventType* identifies the combination of events that triggered the call to the callback function; and *eventCount* increments by one at each call, starting at 1.

The second case applies to reporting signal status changes. The *pContext* parameter takes the value specified in the SapAcquisition::GetSignalStatus method, and *signalStatus* identifies the new signal status that triggered the call to the callback function.

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

---

## SapAcqCallbackInfo::GetContext

```
void* GetContext();
```

### Remarks

Gets the application context associated with acquisition events or signal status reporting. See the SapAcquisition constructor and SapAcquisition::GetSignalStatus for more details.

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::GetCustomData

```
BOOL GetCustomData(void** customData);
```

### Parameters

*customData* Address of a pointer to receive the address to the data buffer

### Remarks

Gets the address of a buffer containing the data associated with a custom event. You must not free the buffer after you are finished using it.

This functionality is usually not supported, except for special versions of certain acquisition devices. See the device User's Manual for more information on availability.

### Example

```
void MyCallback(SapAcqCallbackInfo* pInfo)
{
    // Retrieve the data buffer
    void* pCustomData;
    pInfo->GetCustomData(&pCustomData);

    // Use the data buffer
    //...
}
```

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::GetCustomSize

**BOOL GetCustomSize(int\* *customSize*);**

### Parameters

*customSize* Address of an integer to return the value

### Remarks

Gets the size of the custom data returned by the GetCustomData method.

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::GetEventCount

**int GetEventCount();**

**BOOL GetEventCount(int \* *eventCount*);**

### Parameters

*eventCount* Pointer to the variable to hold the event count

### Remarks

Gets the current count of acquisition events. The initial value is 1 and increments after every call to the acquisition callback function.

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::GetEventInfo

**COREVENTINFO GetEventInfo();**

### Remarks

Gets the low-level Sapera handle of the event info resource. You should not use this method unless you need a handle to the low-level C API to access some functionality not exposed in the C++ API.

### Demo/Example Usage

Not available

---

## SapAcqCallbackInfo::GetEventType

**SapAcquisition::EventType GetEventType();**

**BOOL GetEventType(SapAcquisition::EventType \* *eventType*);**

### Parameters

*eventType* Pointer to the integer variable to hold the event type

### Remarks

Gets the combination of acquisition events that triggered the call to the application callback. Since it is possible for multiple events to trigger one such call, GetEventType may actually return a combination of many events, using a bitwise OR operator. See the SapAcquisition constructor for the list of possible values.

Note that, when the event type is SapAcquisition::EndOfLine or SapAcquisition::EndOfNLines, the line number for which the acquisition callback function is called is not returned through this function, the corresponding bits are always set to 0.

#### Demo/Example Usage

Not available

---

#### SapAcqCallbackInfo::GetGenericParam0

#### SapAcqCallbackInfo::GetGenericParam1

#### SapAcqCallbackInfo::GetGenericParam2

#### SapAcqCallbackInfo::GetGenericParam3

```
BOOL GetGenericParam0(int* paramValue);
BOOL GetGenericParam1(int* paramValue);
BOOL GetGenericParam2(int* paramValue);
BOOL GetGenericParam3(int* paramValue);
```

#### Parameters

*paramValue* Address of an integer where the parameter value is written

#### Remarks

Gets any of the four generic parameters supported by some events. You should use aliases instead when they are available. For example, the 'Feature Info Changed' event of the SapAcquisition class uses the GetFeatureIndex method as an alias to GetGenericParam0. See the acquisition device User's Manual for a list of events using generic parameters.

#### Demo/Example Usage

Not available

---

#### SapAcqCallbackInfo::GetHostTimestamp

```
BOOL GetHostTimestamp(UINT64 *hostTimestamp);
```

#### Parameters

*hostTimestamp* Address of a pointer to receive the host timestamp

#### Remarks

Gets the host timestamp associated with acquisition events or signal status reporting. When a registered event is raised, the host timestamp is retrieved from the host CPU at the kernel level before the callback function executes at the application level.

Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

#### Demo/Example Usage

Not available

---

#### SapAcqCallbackInfo::GetSignalStatus

SapAcquisition::SignalStatus **GetSignalStatus()**;

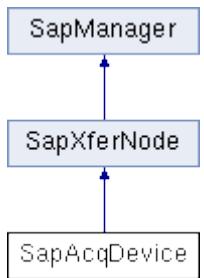
#### Remarks

Gets the input signal status that triggered the call to the application callback. See SapAcquisition::GetSignalStatus for the list of possible values.

#### Demo/Example Usage

Not available

# SapAcqDevice



The SapAcqDevice Class provides the functionality for reading/writing features from/to devices such as Teledyne DALSA GigE Vision cameras. The class also contains functions for sending commands and registering events to devices.

This class is used as a source transfer node to allow data transfers from an acquisition device to another transfer node, such as a buffer.

**Note:** Frame-grabber devices are not supported by this class. The SapAcquisition class must be used in such cases.

```
#include <SapClassBasic.h>
```

## SapAcqDevice Class Members

### Construction

<a href="#"><u>SapAcqDevice</u></a>	Class constructor
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources
<b>General Parameters</b>	
<a href="#"><u>GetConfigFile</u></a>	Gets/sets the name of the acquisition configuration file (CCF)
<a href="#"><u>SetConfigFile</u></a>	
<a href="#"><u>GetReadOnly</u></a>	Gets/sets whether or not the class has read-only access to the device
<a href="#"><u>SetReadOnly</u></a>	
<a href="#"><u>GetUpdateFeatureMode</u></a>	Gets/sets the mode by which features are written to the device
<a href="#"><u>SetUpdateFeatureMode</u></a>	
<a href="#"><u>GetLabel</u></a>	Gets a text description of the acquisition device
<a href="#"><u>GetConfigName</u></a>	Gets/sets the configuration name to be used when saving the device features using the <i>SaveFeatures</i> method
<a href="#"><u>SetConfigName</u></a>	
<a href="#"><u>GetModeName</u></a>	Gets/sets the mode name to be used when saving the device features using the <i>SaveFeatures</i> method
<a href="#"><u>SetModeName</u></a>	
<a href="#"><u>UpdateLabel</u></a>	Updates the device label.

### Parameter Access

<a href="#"><u>IsCapabilityValid</u></a>	Checks for the availability of a low-level Sapera C library capability
<a href="#"><u>GetCapability</u></a>	Gets the value of a low-level Sapera C library capability
<a href="#"><u>IsParameterValid</u></a>	Checks for the availability of a low-level Sapera C library parameter
<a href="#"><u>GetParameter</u></a>	Gets/sets the value of a low-level Sapera C library parameter
<a href="#"><u>SetParameter</u></a>	

### Feature Access

<a href="#"><u>GetFeatureCount</u></a>	Returns the number of features supported by the acquisition device
<a href="#"><u>GetFeatureNameByIndex</u></a>	Returns the name of a feature associated with a specified index
<a href="#"><u>GetFeatureIndexByName</u></a>	Returns the index of a feature associated with a specified name
<a href="#"><u>IsFeatureAvailable</u></a>	Returns whether or not a feature is supported by the acquisition device
<a href="#"><u>GetFeatureInfo</u></a>	Returns information on a feature associated with a specified name or index

<u><a href="#">GetFeatureValue</a></u>	Returns the value of a feature associated with a specified name or index
<u><a href="#">SetFeatureValue</a></u>	Sets the value of a feature associated with a specified name or index
<u><a href="#">UpdateFeaturesFromDevice</a></u>	Gets all the features from the acquisition device at once
<u><a href="#">UpdateFeaturesToDevice</a></u>	Sets all the features to the acquisition device at once
<u><a href="#">LoadFeatures</a></u>	Loads all the features from a configuration file
<u><a href="#">SaveFeatures</a></u>	Saves all (or a subset of) features to a configuration file
<u><a href="#">IsFlatFieldAvailable</a></u>	Gets availability of camera-based flat-field correction
<u><a href="#">GetCategoryCount</a></u>	Returns the number of unique feature category names
<u><a href="#">GetCategoryPath</a></u>	Returns the full path name of a unique feature category
<b>Bayer Management</b>	
<u><a href="#">IsRawBayerOutput</a></u>	Returns whether or not the acquisition device output is raw Bayer
<b>Event Management</b>	
<u><a href="#">GetEventCount</a></u>	Returns the number of events supported by the acquisition device
<u><a href="#">GetEventNameByIndex</a></u>	Returns the name of an event associated with a specified index
<u><a href="#">GetEventIndexByName</a></u>	Returns the index of an event associated with a specified name
<u><a href="#">IsEventAvailable</a></u>	Returns whether or not an event is supported by the acquisition device
<u><a href="#">RegisterCallback</a></u>	Registers a callback function for the event associated with a specified name or index
<u><a href="#">UnregisterCallback</a></u>	Unregisters a callback function on the event associated with a specified name or index
<u><a href="#">IsCallbackRegistered</a></u>	Returns whether or not a callback function was registered on the event associated with a specified name or index
<b>File Management</b>	
<u><a href="#">GetFileCount</a></u>	Returns the number of files supported by the acquisition device
<u><a href="#">GetFileNameByIndex</a></u>	Returns the name of a device file associated with a specified index
<u><a href="#">GetFileIndexByName</a></u>	Returns the index of a device file associated with a specified name
<u><a href="#">Is FileAccess Available</a></u>	Gets availability of file access by the acquisition device
<u><a href="#">GetFileProperty</a></u>	Gets a property of a specific file on the acquisition device
<u><a href="#">WriteFile</a></u>	Writes a file to an acquisition device
<u><a href="#">ReadFile</a></u>	Reads a file from an acquisition device
<u><a href="#">DeleteDeviceFile</a></u>	Deletes a file from the acquisition device

## SapAcqDevice Member Functions

The following are members of the SapAcqDevice Class.

---

### SapAcqDevice::SapAcqDevice

**SapAcqDevice**(SapLocation *location* = SapLocation::ServerSystem, BOOL *readOnly* = FALSE);

**SapAcqDevice**(SapLocation *location*, const char \**configFile*);

#### Parameters

<i>location</i>	SapLocation object specifying the server where the acquisition device is located and the index of the acquisition device on this server.
<i>readOnly</i>	TRUE to force read-only access to the device. If another application is already accessing the device (through this class) use this function to obtain read-only access to the device. To know what functions of the SapAcqDevice class are accessible with this option, refer to the function documentation.
<i>configFile</i>	Name of the acquisition configuration file (CCF) that describes all the acquisition parameters. A CCF file can be created using the <i>CamExpert</i> utility.

#### Remarks

The SapAcqDevice constructor does not actually create the low-level Sapera resources. To do this, you must

call the `SapAcqDevice::Create` method.

The first constructor is used when no configuration file is required. In such a case the default parameters of the acquisition device are used. You can optionally obtain read-only access to the device. This option is useful only when another application has already obtained a read-write access to the same device.

The second constructor allows you to load a configuration file (CCF) previously created by the CamExpert tool or by your own application.

The `SapAcqDevice` object is used only for storing the acquisition device parameters. To acquire data, use the `SapTransfer` Class (or one of its derived classes) and pass the `SapAcqDevice` object as a parameter for the constructor. `SapTransfer` then handles the actual data transfer. You can also use the `SapAcqDeviceToBuf` specialized transfer class class to simplify this task.

#### **Demo/Example Usage**

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Camera Events Example, Camera Features Example, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example

---

### **SapAcqDevice::Create**

**BOOL Create();**

#### **Return Value**

Returns TRUE if successful, FALSE otherwise.

#### **Remarks**

Creates all the low-level Sapera resources needed by the acquisition object. Always call this method before `SapTransfer::Create`.

#### **Demo/Example Usage**

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Camera Events Example, Camera Features Example, Camera Files Example, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example

---

### **SapAcqDevice::DeleteDeviceFile**

**BOOL DeleteDeviceFile(const char \*deviceFileName);**

**BOOL DeleteDeviceFile(int deviceFileIndex);**

#### **Parameters**

`deviceFileName` Name of the device file. See the acquisition device User's Manual for the list of supported files.

`deviceFileIndex` Index of the file. All indices in the range from 0 to the value returned by the `GetFileCount` method, minus 1, are valid.

#### **Return Value**

Returns TRUE if successful, FALSE otherwise.

#### **Remarks**

Deletes the specified file on the device.

To find out which device files names are available, use the `GetFileCount` function together with the `GetFileNameByIndex` function.

In order to use this function with an `deviceFileIndex` argument, you first need to call the `GetFileIndexByName` function to retrieve the index corresponding to the file you want to delete.

#### **Demo/Example Usage**

Camera Files Example

---

### **SapAcqDevice::Destroy**

**BOOL Destroy();**

#### **Return Value**

Returns TRUE if successful, FALSE otherwise.

#### **Remarks**

Destroys all the low-level Sapera resources needed by the acquisition object. Always call this method after `SapTransfer::Destroy`.

## Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Camera Events Example, Camera Features Example, Camera Files Example, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example

---

## SapAcqDevice::GetCapability

**BOOL GetCapability(int cap, void \*pValue);**

### Parameters

*cap* Low-level Sapera C library capability to read

*pValue* Pointer to capability value to read back

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

This method allows direct read access to low-level Sapera C library capabilities for the acquisition device module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

Note that this method is rarely needed. The SapAcqDevice class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

## Demo/Example Usage

Not available

---

## SapAcqDevice::GetCategoryCount

**BOOL GetCategoryCount(int \*categoryCount);**

### Parameters

*categoryCount* Number of feature categories available on the acquisition device

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Returns the number of unique feature category names. This is equivalent to getting the information for all available features (by calling GetFeatureCount followed by GetFeatureInfo), retrieving the category name for each (by calling SapFeature::GetCategory), and then counting the unique category names.

After calling this function, you can call GetCategoryPath to retrieve full path names for individual features, using a category index which can be any value in the range [0... *categoryCount* -1].

## Demo/Example Usage

Not available

---

## SapAcqDevice::GetCategoryPath

**BOOL GetCategoryPath(int categoryIndex, char\* path, int pathSize);**

### Parameters

*categoryIndex* Index of the category. All indices from 0 to the value returned by the GetCategoryCount function, minus 1, are valid.

*path* Returns the full path name of the category associated with the specified index

*pathSize* Size (in bytes) of the buffer pointed to by *path*

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Returns the full path name of a feature category at a specified index, following a call to the GetCategoryCount function to get the total number of categories. The returned path name is formatted according to the following rules:

All path names begin with "\Root" or "\SaperaRoot"

Top level categories are returned as "\Root\CategoryName"

Second level categories are returned as "\Root\CategoryName\SubCategoryName"

and so on...

This allows parsing of category path names so that these can be shown using a hierarchical view in a GUI based application.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::GetConfigFile, SapAcqDevice::SetConfigFile

```
const char* GetConfigFile();
BOOL SetConfigFile(const char* configfile);
```

#### Parameters

*configFile* Name of the configuration file

#### Remarks

Gets/sets the name of the acquisition configuration file (CCF) to be loaded at creation, that is, when the Create method is called.

You normally set the initial value for this attribute in the SapAcqDevice constructor. If you use the default constructor, then this value is NULL.

You can only call SetConfigFile before the Create method.

#### Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo

---

### SapAcqDevice::GetConfigName, SapAcqDevice::SetConfigName

```
const char* GetConfigName();
BOOL SetConfigName(const char* configName);
```

#### Parameters

*configName* Name of the configuration to be written to the CCF file. The length of the string must not exceed 64 characters.

#### Remarks

Gets/sets the configuration name to be used when saving the device features using the SaveFeatures method. It is then possible to uniquely identify different configuration files when the company name, camera model name, and mode name are the same. For example, 'High Contrast' might be used as configuration name.

When loading a configuration file using the LoadFeatures method, this parameter is automatically updated.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::GetEventCount

```
BOOL GetEventCount(int *eventCount);
```

#### Parameters

*eventCount* Number of events supported by the acquisition device

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Returns the number of events supported by the acquisition device. Devices do not necessarily support the same event set. For instance you can use this function to retrieve the number of events and then get the name of those event using GetEventNameByIndex, using an event index which can be any value in the range 0 to the value returned by this function, minus 1.

#### Demo/Example Usage

Camera Events Example

---

### SapAcqDevice::GetEventIndexByName

**BOOL GetEventIndexByName**(const char\* *eventName*, int\* *eventIndex*);

**Parameters**

*eventName* Event name. See the acquisition device User's Manual for the list of supported events.

*eventIndex* Returns the index of the event associated with the specified name

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the index of an event associated with a specified name. This function is useful in building a list of indexes associated with the event names you commonly use. You can then access those events by index to increase performance.

**Demo/Example Usage**

Not available

---

### SapAcqDevice::GetEventNameByIndex

**BOOL GetEventNameByIndex**(int *eventIndex*, char\* *eventName*, int *eventNameSize*);

**Parameters**

*eventIndex* Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.

*eventName* Returns the name of the event associated with the specified index

*eventNameSize* Size (in bytes) of the buffer pointed to by *eventName*

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the name of an event associated with a specified index. This method is especially useful when converting an event index (retrieved from your callback information) to the corresponding name.

**Demo/Example Usage**

Camera Events Example

---

### SapAcqDevice::GetFeatureCount

**BOOL GetFeatureCount**(int\* *featureCount*);

**Parameters**

*featureCount* Number of features supported by the acquisition device

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the number of features supported by the acquisition device. Different devices do not necessarily support the same feature set. You can get information about each feature by calling the GetFeatureInfo method, using an index which can be any value in the range from 0 to the value returned by this method, minus 1.

The returned value is only meaningful after calling the Create method.

**Demo/Example Usage**

Camera Events Example, Camera Features Example

---

### SapAcqDevice::GetFeatureIndexByName

**BOOL GetFeatureIndexByName**(const char\* *featureName*, int\* *featureIndex*);

**Parameters**

*featureName* Name of the feature. See the acquisition device User's Manual for the list of supported features.

*featureIndex* Returns the index of the feature associated with the specified name

**Return Value**

Returns TRUE if successful, FALSE otherwise

### Remarks

Returns the index of a feature associated with a specified name. This function is useful in building a list of indexes associated with the feature names you commonly use. Then you can access those features by index to increase performance.

### Demo/Example Usage

Not available

---

## SapAcqDevice::GetFeatureInfo

```
BOOL GetFeatureInfo(const char* featureName, SapFeature* feature);  
BOOL GetFeatureInfo(int featureIndex, SapFeature* feature);
```

### Parameters

<i>featureName</i>	Name of the feature. See the acquisition device User's Manual for the list of supported features.
<i>featureIndex</i>	Index of the feature. All indices from 0 to the value returned by the GetFeatureCount method, minus 1, are valid.
<i>Feature</i>	Pointer to a SapFeature object to store the feature information

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Returns information on a feature associated with a specified name or index. All information about the feature is stored in a SapFeature object. This object contains the attributes of the feature such as name, type, range, and so forth. See the SapFeature class for more details.

Note that you must call the Create method for the SapFeature object before calling this method.

### Demo/Example Usage

Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, Grab Console Example

---

## SapAcqDevice::GetFeatureNameByIndex

```
BOOL GetFeatureNameByIndex(int featureIndex, char* featureName, int featureNameSize);
```

### Parameters

<i>featureIndex</i>	Index of the feature. All indices from 0 to the value returned by the GetFeatureCount method, minus 1, are valid.
<i>featureName</i>	Returns the name of the feature associated with the specified index
<i>featureNameSize</i>	Size (in bytes) of the buffer pointed to by <i>featureName</i>

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Returns the name of a feature associated with a specified index. For instance you can use this function to display the names of all features supported by the device.

### Demo/Example Usage

Camera Features Example

---

## SapAcqDevice::GetFeatureValue

```
bool GetFeatureValue(const char* featureName, INT32* featureValue);  
bool GetFeatureValue(const char* featureName, UINT32* featureValue);  
bool GetFeatureValue(const char* featureName, INT64* featureValue);  
bool GetFeatureValue(const char* featureName, UINT64* featureValue);  
bool GetFeatureValue(const char* featureName, float* featureValue);  
bool GetFeatureValue(const char* featureName, double* featureValue);  
bool GetFeatureValue(const char* featureName, BOOL* featureValue);  
bool GetFeatureValue(const char* featureName, char* featureString, int featureStringSize);  
bool GetFeatureValue(const char* featureName, SapBuffer* featureBuffer);  
bool GetFeatureValue(const char* featureName, SapLut* featureLut);
```

```

bool GetFeatureValue(int featureIndex, INT32* featureValue);
bool GetFeatureValue(int featureIndex, UINT32* featureValue);
bool GetFeatureValue(int featureIndex, INT64* featureValue);
bool GetFeatureValue(int featureIndex, UINT64* featureValue);
bool GetFeatureValue(int featureIndex, float* featureValue);
bool GetFeatureValue(int featureIndex, double* featureValue);
bool GetFeatureValue(int featureIndex, BOOL* featureValue);
bool GetFeatureValue(int featureIndex, char* featureString, int featureStringSize);
bool GetFeatureValue(int featureIndex, SapBuffer* featureBuffer);
bool GetFeatureValue(int featureIndex, SapLut* featureLut);

```

#### Parameters

<i>featureName</i>	Name of the feature. See the acquisition device User's Manual for the list of supported features.
<i>featureIndex</i>	Index of the feature. All indices from 0 to the value returned by the GetFeatureCount method, minus 1, are valid.
<i>featureValue</i>	Returns the value of the specified feature. You must choose the which function overload to use according to the feature type.
<i>featureString</i>	Returns the contents of a string feature
<i>featureStringSize</i>	Size (in bytes) of the buffer pointed to by <i>featureString</i>
<i>featureBuffer</i>	SapBuffer object for retrieving a buffer feature
<i>featureLut</i>	SapLut object for retrieving a LUT feature

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Returns the value of a feature associated with a specified name or index.

To find out which overloaded function to use, you must obtain the type of the feature by calling the GetFeatureInfo method, followed by SapFeature::GetType. In the case of a class type (such as SapBuffer or SapLut), you must call the Create method for that object before calling GetFeatureValue. To find out if the feature is readable, use SapFeature::GetAccessMode.

Note that, except for unitless features, each feature has its specific native unit, for example, milliseconds, KHz, tenth of degree, etc. This information is obtained through the SapFeature::GetSiUnit and SapFeature::GetSiToNativeExp10 functions.

When dealing with enumerations, it is recommended to always use the string representation to read the value. The actual integer value corresponding to the enumeration string can vary from one acquisition device to another, but the string representation is guaranteed to always represent the same setting, even across manufacturers.

#### Demo/Example Usage

Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, Grab CameraLink Example

### SapAcqDevice::GetFileCount

```
BOOL GetFileCount(int* fileCount);
```

#### Parameters

<i>fileCount</i>	Number of files supported by the acquisition device
------------------	---

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Returns the number of files supported by the acquisition device. Use the returned value together with the GetFileNameByIndex function to get a list of supported device file names.

#### Demo/Example Usage

Camera Files Example

### SapAcqDevice::GetFileIndexByName

**BOOL GetFileIndexByName**(const char\* *fileName*, int\* *fileIndex*);

**Parameters**

<i>fileName</i>	Name of the device file. See the acquisition device User's Manual for the list of supported files.
<i>fileIndex</i>	Returned index of the device file associated with the specified name

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the index of a device file associated with a specified name. This function is useful in building a list of indexes associated with the device file names you commonly use. You can then access those device files by index to increase performance.

**Demo/Example Usage**

Not available

---

### SapAcqDevice::GetFileNameByIndex

**BOOL GetFileNameByIndex**(int *fileIndex*, char\* *fileName*, int *fileNameSize*);

**Parameters**

<i>fileIndex</i>	Index of the device file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
<i>fileName</i>	Returned name of the device file associated with the specified index
<i>fileNameSize</i>	Size (in bytes) of the buffer pointed to by <i>fileName</i>

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the name of a device file associated with a specified index. Use this function together with the GetFileCount function to find out which device files names are available.

**Demo/Example Usage**

Camera Files Example

---

### SapAcqDevice::GetProperty

**BOOL GetGetProperty**(int *fileIndex*, SapAcqDevice::FileProperty *propertyType*,  
UINT64\* *filePropertyValue*);

**BOOL GetGetProperty**(const char\* *fileName*, SapAcqDevice::FileProperty *propertyType*,  
UINT64\* *filePropertyValue*);

**Parameters**

<i>fileIndex</i>	Index of the device file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
<i>fileName</i>	Name of the device file
<i>propertyType</i>	Device file property to inquire, can be one of the following: SapAcqDevice::FilePropertyAccessMode      Access mode for the device file. SapAcqDevice::FilePropertySize      Device file size, in bytes
<i>filePropertyValue</i>	Returned property value.

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Returns the value for the specified property type for the device file. When inquiring the file access mode, the possible values are:

SapAcqDevice::FileAccessModeNone  
SapAcqDevice::FileAccessModeReadOnly  
SapAcqDevice::FileAccessModeWriteOnly

## SapAcqDevice::FileAccessModeReadWrite

To find out which device files names are available, use the GetFileCount function together with the GetFileNameByIndex function.

In order to use this function with a *fileIndex* argument, you first need to call the GetFileIndexByName function to retrieve the index corresponding to the file you want.

### Demo/Example Usage

Camera Files Example

---

## SapAcqDevice::GetLabel

```
const char* GetLabel();
```

### Remarks

Gets a text description of the acquisition device resource. This attribute is initially set to an empty string. After a successful call to the Create method, it is composed of the name of the server where the acquisition device resource is located and the name of this resource: *ServerName [ResourceName]*.

Example: "Genie\_HM1400\_1 [ UserName ]"

The part of the label inside the square brackets actually corresponds to the value of the 'DeviceUserID' feature, which can be modified by the application. When this happens, the label is automatically updated, and the application callback function for the SapManager::EventResourceInfoChanged event is invoked (if registered using the SapManager::RegisterServerCallback function).

### Demo/Example Usage

Not available

---

## SapAcqDevice::GetModeName, SapAcqDevice::SetModeName

```
const char* GetModeName();
BOOL SetModeName(const char* modeName);
```

### Parameters

*modeName* Name of the camera mode to be written to the CCF file. The length of the string must not exceed 64 characters.

### Remarks

Gets/sets the mode name to be used when saving the device features using the SaveFeatures method. It is then possible to uniquely identify different modes when the company name and camera model name are the same. For example, 'Single-Channel, Free-Running' might be used as mode name.

When loading a configuration file using the LoadFeatures method, this parameter is automatically updated.

### Demo/Example Usage

Not available

---

## SapAcqDevice::GetParameter, SapAcqDevice::SetParameter

```
BOOL GetParameter(int param, void *pValue);
BOOL SetParameter(int param, int value);
BOOL SetParameter(int param, void *pValue);
```

### Parameters

*param* Low-level Sapera C library parameter to read or write

*paramValue* Pointer to parameter value to read back or to write

*value* New parameter value to write

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the acquisition device module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value that is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not

already supported by the SapAcqDevice Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Acquisition Parameters Reference Manual* and *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::GetReadOnly, SapAcqDevice::SetReadOnly

```
BOOL GetReadOnly();  
BOOL SetReadOnly(BOOL readOnly);
```

#### Parameters

*readOnly*      TRUE to force read-only access to the device

#### Remarks

Gets/sets whether or not the class has read-only access to the device. See the SapAcqDevice constructor for more detail on this option. You can only call SetReadOnly before the Create method.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode

```
UpdateFeatureMode GetUpdateFeatureMode();  
BOOL SetUpdateFeatureMode(UpdateFeatureMode mode);
```

#### Parameters

*mode*      The mode can be one of the following values:

SapAcqDevice::UpdateFeatureAuto	New feature values are immediately sent to the acquisition device
SapAcqDevice::UpdateFeatureManual	New feature values are temporarily cached before being sent to the acquisition device

#### Remarks

Gets/sets the mode by which features are written to the device. In the automatic mode, every time a feature value is modified using the SetFeatureValue method, it is immediately sent to the device. In the manual mode, each feature value is temporarily cached until the UpdateFeaturesToDevice method is called to send all values to the device at once.

Note, for devices not using the Genie Framework, only the SapAcqDevice::UpdateFeatureAuto mode is implemented; setting the update mode to SapAcqDevice::UpdateFeatureManual has no effect. Consequently, the SapAcqDevice::UpdateFeaturesFromDevice and SapAcqDevice::UpdateFeaturesToDevice functions are not implemented.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::IsCallbackRegistered

```
BOOL IsCallbackRegistered(const char* eventName, BOOL* isRegistered);  
BOOL IsCallbackRegistered(int eventIndex, BOOL* isRegistered);
```

#### Parameters

*eventName*      Name of the event. See the acquisition device User's Manual for the list of supported events.

*eventIndex*      Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.

*isRegistered*      Returns TRUE if a callback function was registered on this event. FALSE otherwise

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Checks whether or not a callback function was registered on the event associated with a specified name or index. For example, you may use this function in a loop to find out if the callback function associated with the current event index has to be unregistered.

## Demo/Example Usage

Camera Events Example, Camera Features Example

---

### SapAcqDevice::IsCapabilityValid

**BOOL IsCapabilityValid(int cap);**

#### Parameters

*cap* Low-level Sapera C library capability to be checked

#### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library capability for the acquisition device module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapAcqDevice class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

## Demo/Example Usage

Not available

---

### SapAcqDevice::IsEventAvailable

**BOOL IsEventAvailable(const char\* eventName, BOOL\* isAvailable);**

#### Parameters

*eventName* Name of the event. See the acquisition device User's Manual for the list of supported events.

*isAvailable* Returns TRUE if the event is supported by the acquisition device. FALSE otherwise

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Checks whether or not an event is supported by the acquisition device. This function is useful when an application supports several acquisition devices, each having a different event set.

## Demo/Example Usage

GigE FlatField Demo

---

### SapAcqDevice::IsFeatureAvailable

**BOOL IsFeatureAvailable(const char \*featureName, BOOL \*isAvailable);**

#### Parameters

*featureName* Name of the feature. See device User's Manual for the list of supported features.

*isAvailable* TRUE if the feature is supported by the device. FALSE otherwise

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Returns whether or not a feature is supported by the acquisition device. This function is useful when an application supports several acquisition devices each having a different feature set.

## Demo/Example Usage

GigE FlatField Demo, GigE Sequential Grab Demo, Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE Camera LUT Example, Grab CameraLink Example

---

### SapAcqDevice::IsFileAccessAvailable

**BOOL IsFileAccessAvailable();**

#### Remarks

Gets availability of file access by the acquisition device. If this function returns FALSE, then you should not use

the GetFileCount, GetFileNameByIndex, GetFileIndexByName, GetFileProperty, WriteFile, ReadFile, and DeleteDeviceFile functions.

### Demo/Example Usage

Not available

---

## SapAcqDevice::IsFlatFieldAvailable

**BOOL IsFlatFieldAvailable();**

### Remarks

Gets availability of hardware-based flat-field correction. You can only call IsFlatFieldAvailable after the Create method.

### Demo/Example Usage

GigE FlatField Demo

---

## SapAcqDevice::IsParameterValid

**BOOL IsParameterValid(int param);**

### Parameters

*param* Low-level Sapera C library parameter to be checked

### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

### Remarks

Checks for the availability of a low-level Sapera C library parameter for the acquisition device module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapAcqDevice class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapAcqDevice::IsRawBayerOutput

**BOOL IsRawBayerOutput();**

### Remarks

Returns whether or not the current pixel format in the acquisition device is of the 'raw Bayer' type, and thus can be processed using software Bayer conversion.

### Demo/Example Usage

Not available

---

## SapAcqDevice::LoadFeatures

**BOOL LoadFeatures(const char\* configFile);**

### Parameters

*configFile* Name of the configuration file (CCF) to load the features from

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Loads all the features from a Sapera LT camera configuration file (CCF), and writes them to the acquisition device. This CCF file is generated by the CamExpert utility provided with Sapera LT, or by calling the SaveFeatures method.

For devices that support hardware persistence storage (for example, Genie cameras), loading a CCF file is not mandatory. For other devices, you must load a CCF file to ensure the device is in a usable state. See your acquisition device User's Manual to find out which category a specific acquisition device belongs to.

Note that you cannot call this method if the current object was constructed with read-only access. See the SapAcqDevice constructor for details.

## Demo/Example Usage

Not available

---

### SapAcqDevice::ReadFile

BOOL **ReadFile**(const char \* *deviceFileName*, const char \* *localFilePath*);  
BOOL **ReadFile**(int *deviceFileIndex*, const char \* *localFilePath*);

#### Parameters

<i>deviceFileName</i>	Name of the device file. See the acquisition device User's Manual for the list of supported files.
<i>deviceFileIndex</i>	Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
<i>localFilePath</i>	Full directory path and filename on the host computer to save the file.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reads the specified file from the device and saves it in the specified location on the host computer.

To find out which device files names are available, use the GetFileCount function together with the GetFileNameByIndex function.

In order to use this function with an *deviceFileIndex* argument, you first need to call the GetFileIndexByName function to retrieve the index corresponding to the file you want to delete.

## Demo/Example Usage

Camera Files Example

---

### SapAcqDevice::RegisterCallback

BOOL **RegisterCallback**(const char\* *eventName*, SapAcqDeviceCallback *callback*, void\* *context*);  
BOOL **RegisterCallback**(int *eventIndex*, SapAcqDeviceCallback *callback*, void\* *context*);

#### Parameters

<i>eventName</i>	Event name. See the acquisition device User's Manual for the list of supported events.
<i>eventIndex</i>	Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.
<i>callback</i>	Address of a user callback function of the following form:
	void MyCallback(SapAcqDeviceCallbackInfo* pInfo) { } Pointer to a user storage (that is, variable, structure, buffer, etc). Can be NULL.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Registers an event by associating a callback function for the specified name or index. When the event occurs in the acquisition device, this callback function is called. It provides information on the corresponding event using a SapAqcDeviceCallbackInfo object. Refer to this class for more details.

The context pointer is also returned by the callback function, allowing for the of exchange application specific information.

#### Example

```
void MyCallback(SapAcqDeviceCallbackInfo* pInfo)
{
    // Access information using functions of SapAcqDeviceCallbackInfo class
    // ...
}

main()
{
    // ...
    acqDevice.RegisterCallback("FeatureValueChanged", MyCallback, NULL);
    // ...
    acqDevice.UnregisterCallback("FeatureValueChanged");
    // ...
}
```

}

## Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

---

### SapAcqDevice::SaveFeatures

BOOL **SaveFeatures**(const char\* *configFile*);

#### Parameters

*configFile* Name of the configuration file (CCF) to save the features to

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Saves acquisition device features to a Sapera LT camera configuration file (CCF). Not all features are saved. For example, read-only features are not saved by default. Use the **SapFeature::IsSavedToConfigFile** and **SetSavedToConfigFile** methods to control whether each individual feature is saved or not.

This method is useful for acquisition devices that do not support hardware persistence storage in order to retrieve the feature values at a later time. See your acquisition device User's Manual to find out if hardware persistence storage is supported.

## Demo/Example Usage

Not available

---

### SapAcqDevice::SetFeatureValue

BOOL **SetFeatureValue**(const char \**featureName*, INT32 *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, UINT32 *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, INT64 *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, UINT64 *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, float *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, double *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, BOOL *featureValue*);  
BOOL **SetFeatureValue**(const char \**featureName*, const char \**featureString*);  
BOOL **SetFeatureValue**(const char \**featureName*, SapBuffer\* *featureBuffer*);  
BOOL **SetFeatureValue**(const char \**featureName*, SapLut\* *featureLut*);

BOOL **SetFeatureValue**(int *featureIndex*, INT32 *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, UINT32 *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, INT64 *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, UINT64 *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, float *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, double *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, BOOL *featureValue*);  
BOOL **SetFeatureValue**(int *featureIndex*, const char \**featureString*);  
BOOL **SetFeatureValue**(int *featureIndex*, SapBuffer\* *featureBuffer*);  
BOOL **SetFeatureValue**(int *featureIndex*, SapLut\* *featureLut*);

#### Parameters

*featureName* Name of the feature. See the acquisition device User's Manual for the list of supported features.

*featureIndex* Index of the feature. All indices from 0 to the value returned by the **GetFeatureCount** method, minus 1, are valid.

*featureValue* Feature value to write. You must choose which function overload to use according to the feature type.

*featureString* String feature to write

*featureBuffer* SapBuffer object to write

*featureLut* SapLut object to write

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Writes the value of a feature associated with a specified name or index.

To find out which overloaded function to use, you must obtain the type of the feature by calling the `GetFeatureInfo` method, followed by `SapFeature::GetType`. In the case of a class type (such as `SapBuffer` or `SapLut`), you must call the `Create` method for that object before calling `GetFeatureValue`. To find out if the feature is writable, use `SapFeature::GetAccessMode`.

Note that, except for unitless features, each feature has its specific native unit, for example, milliseconds, KHz, tenth of degree, etc. This information is obtained through the `SapFeature::GetSIUnit` and `SapFeature::GetSIToNativeExp10` functions.

Note that you cannot call this method if the current object was constructed with read-only access. See the `SapAcqDevice` constructor for details.

When dealing with enumerations, it is recommended to always use the string representation (`featureString` argument) to set the value. The actual integer value corresponding to the enumeration string can vary from one acquisition device to another, but the string representation is guaranteed to always represent the same setting, even across manufacturers.

### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example

---

## SapAcqDevice::UnregisterCallback

**BOOL UnregisterCallback(const char\* eventName);**  
**BOOL UnregisterCallback(int eventIndex);**

### Parameters

`eventName` Event name. See the acquisition device User's Manual for the list of supported events.  
`eventIndex` Index of the event. All indices in the range from 0 to the value returned by the `GetEventCount` method, minus 1, are valid.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Unregisters a callback function on the event associated with a specified name or index. Use this function in a loop to unregister all the callback functions previously registered.

### Example

```
// Unregisters all the callback functions
//
UINT32 eventCount, eventIndex;
acqDevice.GetEventCount(&eventCount);
for (eventIndex = 0; eventIndex < eventCount; eventIndex++)
{
    BOOL isRegistered;
    acqDevice.IsCallbackRegistered(eventIndex, &isRegistered);
    if (isRegistered)
    {
        acqDevice.UnregisterCallback(eventIndex);
    }
}
```

### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

---

## SapAcqDevice::UpdateFeaturesFromDevice

**BOOL UpdateFeaturesFromDevice();**

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets all the features from the acquisition device at once.

This method can only be used when the feature update mode is set to manual (see `SapAcqDevice::GetUpdateFeatureMode`, `SapAcqDevice::SetUpdateFeatureMode`). In this mode, writing individual features using the `SetFeatureValue` method is done to an internal cache. Calling this method resets the internal cache to the values currently present in the device. This is useful when a certain number of features have been written to the internal cache but you want to undo those settings.

Note that you cannot call this method if the current object was constructed with read-only access. See the

SapAcqDevice constructor for details.

This method is only implemented for acquisition devices which are supported through the Genie Framework.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::UpdateFeaturesToDevice

**BOOL UpdateFeaturesToDevice();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Writes all the features to the acquisition device at once.

This method can only be used when the feature update mode is set to manual (see SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode). In this mode, writing individual features using the SetFeatureValue method is done to an internal cache. After all the required features have been written, call this method to update the acquisition device.

Note that you cannot call this method if the current object was constructed with read-only access. See the SapAcqDevice constructor for details.

This method is only implemented for acquisition devices which are supported through the Genie Framework.

#### Demo/Example Usage

Not available

---

### SapAcqDevice::UpdateLabel

**BOOL UpdateLabel();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Updates the acquisition device label. This function can be used if the device label is changed after creation of the first Sapera object (device parameters are populated at this time with locally persisted values).

#### Demo/Example Usage

Camera Files Example

---

### SapAcqDevice::WriteFile

**BOOL WriteFile(const char \*localFilePath, const char \*deviceFileName);**

**BOOL WriteFile(const char \*localFilePath, int deviceFileIndex);**

#### Parameters

*localFilePath* Full directory path and filename on the host computer of the file to write to the device

*deviceFileName* Name of the device file. See the acquisition device User's Manual for the list of supported files.

*deviceFileIndex* Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reads the specified file from the specified location on the host computer and writes it to the device.

To find out which device files names are available, use the GetFileCount function together with the GetFileNameByIndex function.

In order to use this function with an *deviceFileIndex* argument, you first need to call the GetFileIndexByName function to retrieve the index corresponding to the file you want to delete.

#### Demo/Example Usage

Camera Files Example

# SapAcqDeviceCallbackInfo

The SapAcqDeviceCallbackInfo class acts as a container for storing all arguments to the callback function for the SapAcqDevice class.

```
#include <SapClassBasic.h>
```

## SapAcqDeviceCallbackInfo Class Members

### Construction

[SapAcqDeviceCallbackInfo](#) Class constructor

### Attributes

<a href="#"><u>GetAcqDevice</u></a>	Gets the SapAcqDevice object associated with acquisition device events
<a href="#"><u>GetContext</u></a>	Gets the application context associated with acquisition device events
<a href="#"><u>GetEventInfo</u></a>	Gets the low-level Sapera handle of the event info resource
<a href="#"><u>GetEventCount</u></a>	Gets the current count of acquisition device events
<a href="#"><u>GetEventIndex</u></a>	Gets the index of the event that triggered the call to the application callback
<a href="#"><u>GetHostTimeStamp</u></a>	Gets the timestamp corresponding to the moment when the event occurred on the host
<a href="#"><u>GetAuxiliaryTimeStamp</u></a>	Gets the timestamp corresponding to the moment when the event occurred on the acquisition device
<a href="#"><u>GetCustomData</u></a>	Gets the data associated with a custom event
<a href="#"><u>GetCustomSize</u></a>	Gets the size of the custom data returned by GetCustomData
<a href="#"><u>GetGenericParam0</u></a>	Gets generic parameters supported by some events
<a href="#"><u>GetGenericParam1</u></a>	
<a href="#"><u>GetGenericParam2</u></a>	
<a href="#"><u>GetGenericParam3</u></a>	
<a href="#"><u>GetFeatureIndex</u></a>	Gets the index of the feature associated with the event

## SapAcqDeviceCallbackInfo Member Functions

The following are members of the SapAcqDeviceCallbackInfo Class.

### SapAcqDeviceCallbackInfo::SapAcqDeviceCallbackInfo

[SapAcqDeviceCallbackInfo\(SapAcqDevice\\* \*pAcqDevice\*, void\\* \*context\*, COREVENTINFO \*eventInfo\*\);](#)

#### Parameters

<i>pAcqDevice</i>	SapAcqDevice object which called the callback function.
<i>context</i>	Pointer to the application context.
<i>eventInfo</i>	Low-level Sapera handle of the event info resource

#### Remarks

SapAcqDevice objects create an instance of this class before each call to the acquisition callback method in order to combine all function arguments into one container.

The *context* parameter takes the value specified when calling the SapAcqDevice::RegisterCallback method. The *eventInfo* handle is automatically created by Sapera LT.

Although it is possible to retrieve callback related parameters through *eventInfo*, you should rely on the other parameter retrieval methods in this class instead, like GetFeatureIndex.

#### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

### SapAcqDeviceCallbackInfo::GetAcqDevice

[SapAcqDevice\\* \*\*GetAcqDevice\(\)\*\*](#);

#### Remarks

Gets the SapAcqDevice object associated with acquisition events. See the SapAcqDevice constructor for more details.

#### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

---

### SapAcqDeviceCallbackInfo::GetAuxiliaryTimeStamp

BOOL **GetAuxiliaryTimeStamp**(UINT64\* *auxTimeStamp*);

#### Parameters

*auxTimeStamp* Address of a 64-bit integer to return the timestamp value

#### Remarks

Gets the timestamp corresponding to the moment when the event occurred on the acquisition device. Note that not all devices support this timestamp, and that this value is specific to the device. See the device User's Manual for more information on the availability of this value and the associated unit.

#### Demo/Example Usage

Not available

---

### SapAcqDeviceCallbackInfo::GetContext

void \***GetContext**();

#### Remarks

Gets the application context associated with acquisition events. See the SapAcqDevice::RegisterCallback function for more details.

#### Demo/Example Usage

Not available

---

### SapAcqDeviceCallbackInfo::GetCustomData

BOOL **GetCustomData**(void\*\* *customData*);

#### Parameters

*customData* Address of a pointer to receive the address to the data buffer

#### Remarks

Gets the address of a buffer containing the data associated with a custom event. You must not free the buffer after you are finished using it.

This functionality is usually not supported, except for special versions of certain acquisition devices. See the device User's Manual for more information on availability.

#### Example

```
void MyCallback(SapAcqDeviceCallbackInfo* pInfo)
{
    // Retrieve the data buffer
    void* pCustomData;
    pInfo->GetCustomData(&pCustomData);

    // Use the data buffer
    //...
}
```

#### Demo/Example Usage

Not available

---

### SapAcqDeviceCallbackInfo::GetCustomSize

BOOL **GetCustomSize**(int\* *customSize*);

#### Parameters

*customSize* Address of an integer to return the value

#### Remarks

Gets the size of the custom data returned by the GetCustomData method.

#### Demo/Example Usage

Not available

---

## SapAcqDeviceCallbackInfo::GetEventCount

BOOL **GetEventCount**(int\* *eventCount*);

### Parameters

*eventCount* Address of an integer where the count is written

### Remarks

Gets the current count of acquisition device events. The initial value is 1 and increments after every call to the acquisition callback function.

### Demo/Example Usage

Camera Events Example

---

## SapAcqDeviceCallbackInfo::GetEventIndex

BOOL **GetEventIndex**(int\* *eventIndex*);

### Parameters

*eventIndex* Address of an integer where the event index is written

### Remarks

Gets the index of the current event. Use this index to retrieve the name of the event using the SapAcqDevice::GetEventNameByIndex method.

### Demo/Example Usage

Not available

---

## SapAcqDeviceCallbackInfo::GetEventInfo

COREVENTINFO **GetEventInfo**();

### Remarks

Gets the low-level Saper handle of the event info resource. You should not use this method unless you need a handle to the low-level C API to access some functionality not exposed in the C++ API.

### Demo/Example Usage

Not available

---

## SapAcqDeviceCallbackInfo::GetFeatureIndex

BOOL **GetFeatureIndex**(int\* *featureIndex*);

### Parameters

*featureIndex* Address of an integer where the feature index is written

### Remarks

Gets the index of the feature associated with the event. For example, it is used by the 'Feature Info Changed' event of the SapAcqDevice class. In this case it represents the index of the feature whose attributes have changed. This index ranges from 0 to the value returned by the SapAcqDevice::GetFeatureCount method, minus 1.

### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

---

## SapAcqDeviceCallbackInfo::GetGenericParam0

## SapAcqDeviceCallbackInfo::GetGenericParam1

## SapAcqDeviceCallbackInfo::GetGenericParam2

## SapAcqDeviceCallbackInfo::GetGenericParam3

BOOL **GetGenericParam0**(int\* *paramValue*);

BOOL **GetGenericParam1**(int\* *paramValue*);

BOOL **GetGenericParam2**(int\* *paramValue*);

BOOL **GetGenericParam3**(int\* *paramValue*);

### Parameters

*paramValue* Address of an integer where the parameter value is written

### Remarks

Gets any of the four generic parameters supported by some events. You should use aliases instead when they are available. For example, the 'Feature Info Changed' event of the SapAcqDevice class use the GetFeatureIndex method as an alias to GetGenericParam0. See the acquisition device User's Manual for a list of events using generic parameters.

#### Demo/Example Usage

Not available

---

#### SapAcqDeviceCallbackInfo::GetHostTimeStamp

BOOL GetHostTimeStamp(UINT64\* *hostTimeStamp*);

#### Parameters

*hostTimeStamp* Address of a 64-bit integer where the timestamp value is written

#### Remarks

Gets the timestamp corresponding to the moment when the event occurred on the host. When a registered event is raised, the host timestamp is retrieved from the host CPU at the kernel level before the callback function executes at the application level.

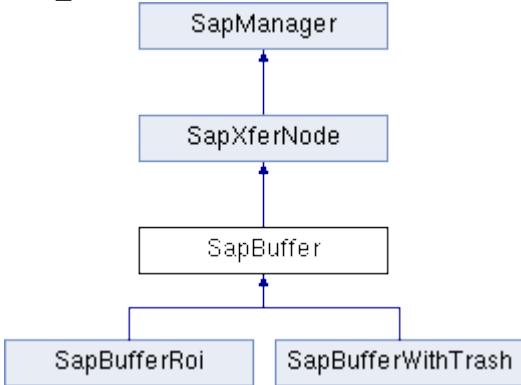
Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

#### Demo/Example Usage

Not available

# SapBuffer



The SapBuffer Class includes the functionality to manipulate an array of buffer resources. The array contains buffer resources with the same dimensions, format, and type.

The buffer object can be used as a destination transfer node to allow transferring data from a source node (such as acquisition or another buffer) to a buffer resource. It can also be used as a source transfer node to allow transferring data from a buffer resource to another buffer. The array of buffers allows a transfer to cycle throughout all the buffers.

The buffer object can be displayed using SapView Class and processed using the SapProcessing Class.

For more information on using buffers, see the Working with Buffers section of the Saper LT Programmer's Manual.

```
#include <SapClassBasic.h>
```

## SapBuffer Class Members

### Construction

<u>SapBuffer</u>	Class constructor
<u>Create</u>	Allocates the low-level Saper resources
<u>Destroy</u>	Releases the low-level Saper resources

### Attributes

<u>GetCount</u>	Gets/sets the number of buffer resources in the array
<u>SetCount</u>	
<u>GetWidth</u>	Gets/sets the width (in pixels) of all the buffer resources
<u>SetWidth</u>	
<u>GetHeight</u>	Gets/sets the height (in lines) of all the buffer resources
<u>SetHeight</u>	
<u>GetFormat</u>	Gets/sets the data format of all the buffer resources
<u>SetFormat</u>	
<u>IsMultiFormat</u>	Checks if the buffer resources are of multiformat type.
<u>GetNumPages</u>	Gets the number of pages in a buffer resource
<u>GetPageFormat</u>	Gets the format of the active page of the current buffer resource
<u>GetType</u>	Gets/sets the type of all the buffer resources
<u>SetType</u>	
<u>SetVirtualAddress</u>	Sets the virtual addresses to use for creating buffer resources
<u>SetPhysicalAddress</u>	Sets the physical addresses to use for creating buffer resources
<u>GetParameters</u>	Gets/sets the count, width, height, format, and type of all the buffer resources
<u>SetParameters</u>	
<u>SetParametersFromFile</u>	Sets the attributes of all the buffer resources from an existing file storing a Saper buffer
<u>GetPixelDepth</u>	Gets/sets the number of significant bits of all the buffer resources

<a href="#">SetPixelDepth</a>	
<a href="#">GetFrameRate</a>	Gets/sets the frame rate of all the buffer resources
<a href="#">SetFrameRate</a>	
<a href="#">SetFrameRate</a>	
<a href="#">GetBytesPerPixel</a>	Gets the number of bytes required to store a single buffer element
<a href="#">GetPitch</a>	Gets the number of bytes between two consecutive lines of all the buffer resources
<a href="#">GetBufName</a>	Gets the name of a buffer object that is shared between multiple processes
<a href="#">GetHandles</a>	Gets the array of low-level Sapera handles to all the buffer resources
<a href="#">GetHandle operator[]</a>	Gets the low-level Sapera handle of a specified buffer resource
<a href="#">GetIndex,</a>	Gets/sets the index of the current buffer resource
<a href="#">SetIndex</a>	
<b>Operations</b>	
<a href="#">ResetIndex</a>	Initializes the current buffer index
<a href="#">Next</a>	Increments the current buffer index
<a href="#">Clear</a>	Clears the content of all the buffers
<a href="#">Load</a>	Loads an image file into the current buffer resource
<a href="#">Save</a>	Saves the current buffer resource to an image file
<a href="#">Copy</a>	Copies contents of a single buffer resource from another SapBuffer object
<a href="#">CopyAll</a>	Copies contents of all the buffer resources from another SapBuffer object
<a href="#">CopyRect</a>	Copies a rectangular area from a single buffer resource to another buffer resource
<a href="#">SplitComponents</a>	Splits a color or multiformat buffer into its individual monochrome components
<a href="#">MergeComponents</a>	Merges individual monochrome components into a color buffer or individual multiformat components into a multiformat buffer.
<a href="#">ColorConvert</a>	Converts a color image (for example, Bayer format) to RGB format
<a href="#">ColorWhiteBalance</a>	Calculates RGB white balance coefficients for a color image (for example, Bayer format) to be used when converting to RGB format.
<a href="#">Read</a>	Reads a consecutive series of pixel values in the current buffer resource
<a href="#">ReadElement</a>	Reads the pixel value at a specified position in the current buffer resource
<a href="#">ReadLine</a>	Reads a series of linearly positioned pixel values in the current buffer resource
<a href="#">ReadRect</a>	Reads a series of pixel values from a rectangular area in the current buffer resource
<a href="#">Write</a>	Writes a consecutive series of pixel values in the current buffer resource
<a href="#">WriteElement</a>	Writes the pixel value at a specified position in the current buffer resource
<a href="#">WriteLine</a>	Writes a series of linearly positioned pixel values to the current buffer resource
<a href="#">WriteRect</a>	Writes a series of pixel values to a rectangular area in the current buffer resource
<a href="#">GetState</a>	Gets the empty/full state of the current buffer resource
<a href="#">SetState</a>	Sets the empty/full state of the current buffer resource
<a href="#">SetAllState</a>	Gets the empty/full state of all the buffer resources
<a href="#">GetPage</a>	Gets the active page of the current buffer resource for planar or multiformat buffer types
<a href="#">SetPage</a>	Sets the active page of the current buffer resource for planar or multiformat buffer types
<a href="#">SetAllPage</a>	Sets the active page of all the buffer resources for planar or multiformat buffer types
<a href="#">GetAddress</a>	Initiates direct address to buffer resource data by a pointer
<a href="#">ReleaseAddress</a>	End direct buffer resource data access
<a href="#">IsMapped</a>	Indicates if there currently exists a valid virtual data address for a buffer resource
<a href="#">GetCounterStamp</a>	Gets a unique identifier associated with a buffer resource
<a href="#">GetHostCounterStamp</a>	Gets the host counter timestamp at which a specific event occurred.
<a href="#">GetSpaceUsed</a>	Gets the number of data bytes actually stored in a buffer resource

<u>IsBufferTypeSupported</u>	Checks if an acquisition resource supports data transfers to a specific buffer type
<u>IsCapabilityValid</u>	Checks for the availability of a low-level Sapera C library capability
<u>IsParameterValid</u>	Checks for the availability of a low-level Sapera C library parameter
<u>GetCapability</u>	Gets the value of a low-level Sapera C library capability
<u>GetParameter</u>	Gets/sets the value of a low-level Sapera C library parameter
<u>SetParameter</u>	

## SapBuffer Member Functions

The following are members of the SapBuffer Class.

---

### SapBuffer::SapBuffer

```

SapBuffer(
    int count = 1,
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeScatterGather,
    SapLocation loc = SapLocation::ServerSystem
);

SapBuffer(
    int count,
    ULONG_PTR physAddress[]
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeContiguous,
);

SapBuffer(
    int count,
    void* virtAddress[]
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeScatterGather,
);

SapBuffer(
    int count,
    SapXferNode* pSrcNode,
    SapBuffer::Type type = SapBuffer::TypeScatterGather,
    SapLocation loc = SapLocation::ServerSystem
);

SapBuffer(
    const char* fileName,
    SapBuffer::Type type = SapBuffer::TypeScatterGather
    SapLocation loc = SapLocation::ServerSystem
);

SapBuffer(
    int count,
    const char* bufName
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeScatterGather,
    SapLocation loc = SapLocation::ServerSystem
);

SapBuffer(
    int count,
    const char* bufName
);

```

```

SapXferNode* pSrcNode,
SapBuffer::Type type = SapBuffer::TypeScatterGather,
SapLocation loc = SapLocation::ServerSystem
);
SapBuffer(
const char* bufName
int startIndex
int count,
SapBuffer::Type type = SapBuffer::TypeVirtual,
SapLocation loc = SapLocation::ServerSystem
);
SapBuffer(
int count,
SapDisplay* pDisplay
int width = 640,
int height = 480,
SapFormat format = SapFormatMono8,
SapBuffer::Type type = SapBuffer::TypeScatterGather,
);

```

```

SapBuffer(
int count,
SapDisplay* pDisplay
SapXferNode* pSrcNode,
SapBuffer::Type type = SapBuffer::TypeScatterGather,
);

```

## Parameters

<i>count</i>	Number of buffer resources
<i>width</i>	Width (in pixels) of all the buffer resources
<i>height</i>	Height (in lines) of all the buffer resources
<i>format</i>	Data format of all the buffer resources, can be one of the following values:
<b>Monochrome (unsigned)</b>	
SapFormatMono1	1-bit
SapFormatMono8	8-bit
SapFormatMono16	16-bit
SapFormatMono32	32-bit
<b>Monochrome (signed)</b>	
SapFormatInt8	8-bit
SapFormatInt16	16-bit
SapFormatInt32	32-bit
<b>RGB Color</b>	
SapFormatRGB5551	16-bit (5 for each of red/green/blue, 1 for alpha)
SapFormatRGB565	16-bit (5 for red, 6 for green, 5 for blue)
SapFormatRGB888	24-bit (8 for red, 8 for green, 8 for blue), blue component is stored first
SapFormatRGB8888	32-bit (8 for each of red/green/blue, 8 for alpha)
SapFormatRGB101010	32-bit (10 for each of red/green/blue, 2 unused)
SapFormatRGB161616	48-bit (16 for each of red/green/blue)
SapFormatRGB16161616	64-bit (16 for each of red/green/blue/alpha)
SapFormatRGBP8	8-bit planar
SapFormatRGBP16	16-bit planar
SapFormatRGBR888	24-bit (8 for red, 8 for green, 8 for blue), red component is stored first
<b>Bi Color</b>	
SapFormatBICOLOR88	8-bits per component, 32 total.
SapFormatBICOLOR1616	16-bits per component, 64 total
For both bicolor formats, 1 pixel is generated for 2 components (RG or BG) therefore the buffer width is twice the size of the resulting image.	
<b>YUV Color</b>	
SapFormatUYVY	16-bit, 4:2:2 subsampled
SapFormatYUY2	16-bit, 4:2:2 subsampled
SapFormatYVYU	16-bit, 4:2:2 subsampled

	SapFormatYUYV	16-bit, 4:2:2 subsampled
	SapFormatY411	12-bit, 4:1:1 subsampled
	SapFormatY211	8-bit, 4:2:2 subsampled
	SapFormatYUV	32-bit (8 for each of Y/U/V, 8 for alpha)
<b>LAB Color</b>		
	SapFormatLAB	32-bit (8 for each component, 8 unused)
	SapFormatLABP8	8-bit Planar(8 for each component, 8 unused)
	SapFormatLABP16	16-bit Planar (16 for each component, 8 unused)
	SapFormatLAB101010	32-bit (10 for each of red/green/blue, 2 unused)
	SapFormatLAB161616	48-bit (16 for each component, 16 unused)
<b>Other Formats</b>		
	SapFormatHSV	32-bit HSV (8 for each component, 8 unused)
	SapFormatHSI	32-bit HSI (8 for each component, 8 unused)
	SapFormatHSIP8	8-bit HSI planar
	SapFormatFloat	32-bit signed floating point
	SapFormatPoint	64-bit (32-bit signed integer for both X and Y components)
	SapFormatFPoint	64-bit (32-bit signed floating-point for both X and Y components)
<b>Multiformat</b>		
	SapFormatRGB888_MONO8	32-bit (8 for each of red/green/blue, IR)
	SapFormatRGB161616_MONO16	64-bit (16 for each of red/green/blue/IR)  For each line in a buffer, the first $\frac{3}{4}$ (left side) represents the RGB data and the last $\frac{1}{4}$ (right side) represents the monochrome (IR) data.  Note: Multiformat buffer types do not support color conversion; the RGB component must be extracted into a supported RGB format. For load and save operations, only the CRC and RAW formats are supported.
		See also the SapData classes for Sapera data elements described in this document
<b>type</b>	Type of all buffer resources can be one of the following values:	
	SapBuffer::TypeContiguous	Buffers are allocated in Sapera Contiguous Memory, which is one large chunk of non-pageable and non-moveable memory reserved by Sapera at boot time. Buffer data is thus contained in a single memory block (not segmented). These buffers may be used as source and destination for transfer resources.
	SapBuffer::TypeScatterGather	Buffers are allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list may be built. This allows allocation of very large buffers to be used as source and destination for transfer resources. The maximum amount of memory that may be allocated depends on available memory, the operating system, and the application(s) used. If the amount of system memory exceeds 4 GBytes, Sapera automatically uses TypeScatterGatherUnmapped instead.
	SapBuffer::TypeVirtual	Similar to TypeScatterGather, except that the memory pages are not locked. This allows allocation of very large buffers, but they cannot be used as source or destination for transfer resources.
	SapBuffer::TypeOffscreen	Buffers are allocated in system memory. SapView objects created using these buffers may use display adapter hardware to copy from the buffer to video memory. System memory offscreen buffers may be created using any pixel format, but calling the SapView::Show method will take longer to execute if the display hardware does not efficiently support its pixel format.
	SapBuffer::TypeOffscreenVideo	Buffers are allocated in offscreen video memory. SapView objects created using these buffers use display adapter hardware to perform a fast copy in video memory. These buffers are typically used when a graphical element is reused for several consecutive frames without modification. In this case, it is more efficient to keep this element in video memory and use display hardware capabilities.
	SapBuffer::TypeOverlay	Buffers are allocated in video memory. Once you create SapView objects using these buffers and call their Show method once, the display adapter overlay hardware will keep updating the display with the buffer contents with no additional calls. The pixel format of overlay buffers must be supported by the display hardware. Typically, overlay buffers support more pixel formats (like YUV) than offscreen buffers. Also, color keying is supported for overlays. The SapView Class determines the behavior of the overlay regarding key colors.

	SapBuffer::TypeDummy	Dummy buffers do not have any data memory. They may be used as placeholders by transfer resources when there is no physical data transfer.
	SapBuffer::TypeUnmapped	Buffers are allocated as a series of non-contiguous chunks of physical memory. You may not access their data until they have been mapped to virtual memory addresses using the GetAddress method. This type of buffer is useful if the total amount of needed buffer data exceeds the amount of available virtual memory addresses (2 GBytes under 32-bit Windows). To avoid a shortage of virtual memory addresses, use the ReleaseAddress method as soon as you are done accessing their data. Note that you cannot acquire images into these buffers. This buffer type is neither supported nor needed in Sapera LT for 64-bit Windows.
	SapBuffer::TypeScatterGatherUnmapped	These buffers are similar to TypeUnmapped, except that you can acquire images into them. This buffer type is neither supported nor needed in Sapera LT for 64-bit Windows.
	SapBuffer::TypeScatterGatherPhysical	These buffers are needed in 64-bit Windows for some frame grabbers (for example, X64-CL iPro) which feature DMA transfers to the host using 32-bit addresses. These frame grabbers do not support acquisition in regular scatter-gather buffers (SapBuffer::TypeScatterGather), because they require all physical addresses used during DMA transfers to be limited to 32-bit values.
<i>loc</i>	SapLocation object specifying the server on which the buffer resources are to be created. The resource index of the location object is ignored.	
<i>physAddress</i>	Array of physical addresses to use when creating buffer resources. This is intended for cases when you do not want Sapera to allocate buffer memory (in the Create method), and you already know the physical addresses where you want buffers to be located. These addresses typically correspond to hardware devices in the system.	
<i>virtAddress</i>	Array of virtual addresses to use when creating buffer resources. This is intended for cases when you do not want Sapera to allocate buffer memory (in the Create method), but you want to control the allocation and free memory in the application program instead. Memory thus remains available even after calling the Destroy method.	
<i>pSrcNode</i>	Source node object. The <i>width</i> , <i>height</i> , and <i>format</i> parameters are extracted automatically from this object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.	
<i>fileName</i>	Name of a Sapera image file from which to extract the <i>count</i> , <i>width</i> , <i>height</i> , and <i>format</i> parameters	
<i>bufName</i>	Name identifying the buffer object so that it may be shared between multiple processes	
<i>startIndex</i>	Starting index of buffer resource when using a shared buffer object created in another process	
<i>pDisplay</i>	SapDisplay object for creating a compatible buffer object	

## Remarks

The SapBuffer constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

The *count* parameter specifies the number of buffer resources, all of which have the same *width*, *height*, *format*, and *type*. Constructing the object using *physAddress* or *virtAddress* tells Sapera not to perform memory allocation itself in the Create method, but rather to rely on the supplied addresses.

Constructing the object using *pSrcNode* allows Sapera to automatically extract the *width*, *height*, and *format* from the source node to ensure transfer compatibility.

Constructing the object using *fileName* allows Sapera to automatically extract the *count*, *width*, *height*, and *format* from the file to ensure buffer compatibility. You must then use the Load method after calling Create.

The *loc* argument allows the creation of buffer resources on a remote server.

Constructing the object using *bufName* allows sharing of a buffer object between multiple processes. The first process that calls the constructor creates the actual buffer resources. The other processes that call the constructor with the same name automatically use the same resources. You may use the *startIndex* and *count* arguments to use only a subset of all the shared resources in the buffer object.

To transfer data to/from the buffer object, you must use the SapTransfer class (or one of its derived classes) and specify the SapBuffer object as a parameter. The data transfer is then controlled by the SapTransfer class.

On acquisition hardware with an integrated display controller (for example, Bandit 3), it is also possible to

create a buffer object from a display object using the *pDisplay* argument. This is useful, for example, when we need a buffer object of overlay type with a data format which is compatible with the display.

To use this functionality, you must perform the following steps in order:

- call the SapDisplay constructor with a SapLocation object for the acquisition hardware
- call one of the SapBuffer constructors with a SapDisplay object
- call the SapView constructor with a SapDisplay object
- call the Create methods for the display, buffer, and then the view object

Note, for Bayer acquisition the buffer format is either SapFormatMono8 or SapFormatMono16; refer to the SapColorConversion Member Functions class for more information on manipulating Bayer buffers.

For more information on using buffers, see the Working with Buffers section of the Sapera LT Programmer's Manual.

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example, Grab MFC Example

---

## SapBuffer::Clear

```
BOOL Clear();
BOOL Clear(int index);
BOOL Clear(SapData value);
BOOL Clear(int index, SapData value);
```

### Parameters

*index* Buffer resource index  
*value* New value for all buffer elements. See the SapData Class and its derived classes for more details.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Clears the content of a specified buffer resource in the array. If no value is specified, then black (usually 0) is assumed. If no index is specified, all buffers are cleared.

For multiformat buffers (for example, SapFormatRGB888\_MONO8 or RGB161616\_MONO16) use a SapDataRGBA object,

### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo

---

## SapBuffer::ColorConvert

```
BOOL ColorConvert(SapBuffer *pSrc, ColorAlign align, ColorMethod method, SapDataFRGB wbCoef, SapLut
*pLut = NULL);
BOOL ColorConvert(SapBuffer *pSrc, int srcIndex, int dstIndex, ColorAlign align, ColorMethod method,
SapDataFRGB wbCoef, SapLut *pLut = NULL);
```

### Parameters

*pSrc* Buffer object to convert. The input buffer format must be one of the following:  
    SapFormatUint8  
    SapFormatUint16  
*srcIndex* Source buffer resource index  
*dstIndex* Destination buffer resource index in the current object  
*align* Specifies the pixel alignment for the color filter. The alignment mode must correspond to the upper left 2x2 square of your camera's color scheme for Bayer conversion; 1x4 line for Bicolor conversion. If the input buffer is a child, the alignment mode is internally recalculated with respect to the upper left corner. Possible values are:

SapBuffer::ColorAlignGBGR



SapBuffer::ColorAlignBGGR	
SapBuffer::ColorAlignRGGB	
SapBuffer::ColorAlignGRBG	
SapBuffer::ColorAlignRGBG	
SapBuffer::ColorAlignBGRG	
<i>method</i>	Specifies the conversion method. Possible values are:
SapBuffer::ColorMethod1	This technique, based on 3x3 bi linear interpolation, is fast but tends to smooth image edges.
SapBuffer::ColorMethod2	This advanced technique is better for preserving image edges. However it works well only when the image has a strong green content. If not, a little amount of noise may be visible in objects.
SapBuffer::ColorMethod3	This advanced technique is almost as good as method 2 for preserving the edges but is independent of the image green content. Little color artifacts of 1 pixel may be visible in edges.
SapBuffer::ColorMethod4	This technique, based on 2x2 interpolation, is the simplest and fastest. Compared to 3x3 it is better at preserving edge sharpness but introduces a slight jitter in pixel position. In practice it is a good choice for image display but less recommended than 3x3 for accurate image processing.
SapBuffer::ColorMethod5	This technique, based on a set of linear filters, works under the main assumption that edges have much stronger luminance than chrominance component.
SapBuffer::ColorMethod6	Reserved.
SapBuffer::ColorMethod7	Support for bi-color conversion for use with the Teledyne DALSA Piranha 4 camera.
<i>wbCoef</i>	White balance coefficients. Can be calculated by SapBuffer::ColorWhiteBalance or set manually as follows:  SapDataFRGB wb; wb.frgb.red = <Red Gain> wb.frgb.green = <Green Gain> wb.frgb.blue = <Blue Gain>
	If no white balance is required, all gains must be set to 1.0.
<i>pLut</i>	LUT handle. Color lookup table applied after the filtering for color adjustment, for example, gamma correction. The number of entries required by the LUT must be 2N, where N is the buffer's pixel depth. The LUT format must be one of the following according to the output format: SapFormatColorNI8 or SapFormatColorNI16.

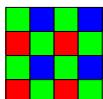
### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Converts images from the color image format to RGB format. The color format assigns each pixel in a monochrome image the value of one color channel. RGB images are created by using neighbouring pixel values to get the two missing color channels at each pixel.

Pixels in a row of a color image alternate between the green channel value and either the red or the blue channel value. The default scheme is shown below.



The missing color channel values are determined using neighbouring pixel values for the color channel in question either by linear interpolation (`SapBuffer::ColorMethod1`) or by one of the advanced methods (`SapBuffer::ColorMethod1` or `ColorMethod3`). The advanced methods are more computationally expensive than the interpolation method but give better image quality when the input image contains many strong edges.

If the input image is 16-bit and the significant bits are stored in the lower bits (for example, 10-bit camera) the buffer's pixel depth (`CORBUFFER_PRM_PIXEL_DEPTH`) must be set to the number of significant bits.

The white balance coefficients (`wbCoef`) are the R, G, and B gains applied to the input image before the filtering. These gains are used to balance the three color components so that a pure white at the input gives a pure white at the output.

The output lookup table (`lut`) may be used to apply a color correction after the filtering. A commonly used correction is gamma (`SapLut::Gamma` function of the `LUT` class).

### Demo/Example Usage

Not available

---

## **SapBuffer::ColorWhiteBalance**

```
BOOL ColorWhiteBalance (ColorAlign align, SapDataFRGB *pWbCoef)  
BOOL ColorWhiteBalance (int index, ColorAlign align, SapDataFRGB *pWbCoef);
```

### Parameters

<i>index</i>	Index of buffer object to convert. The input buffer format must be one of the following: <code>SapFormatUInt8</code> <code>SapFormatUInt16</code>
<i>align</i>	Specifies the pixel alignment for the color filter. The alignment mode must correspond to the upper left 2x2 square of your camera's color scheme for Bayer conversion; 1x4 line for Bicolor conversion. If the input buffer is a child, the alignment mode is internally recalculated with respect to the upper left corner. Possible values are:  <code>SapBuffer::ColorAlignGBGR</code>  <code>SapBuffer::ColorAlignBGGR</code>  <code>SapBuffer::ColorAlignRGGB</code>  <code>SapBuffer::ColorAlignGRBG</code>  <code>SapBuffer::ColorAlignRGBG</code>  <code>SapBuffer::ColorAlignBGRG</code> 
<i>pWbCoef</i>	Pointer to memory location to store calculated white balance coefficients. Coefficients are calculated for the R, G, and B color channels.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Calculates the white balance coefficients used by `SapBuffer::ColorConvert` on a color-encoded input image. The first prototype functions on the current buffer object (buffer index  $o = 0$ ). The input buffer should be a region-of-interest (ROI) of a color-encoded image containing a uniformly illuminated white region. The intensity of the pixels should be as high as possible but not saturated. The coefficients are calculated as follows:

$$G_R = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{R}$$

$$G_G = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{G}$$

$$G_B = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{B}$$

where  $\overline{R}$ ,  $\overline{G}$  and  $\overline{B}$  are the average value of each color component calculated on all the pixels of the input image.

### Demo/Example Usage

Not available

---

## SapBuffer::Copy

BOOL **Copy**(SapBuffer\* *pSrc*);  
BOOL **Copy**(SapBuffer\* *pSrc*, int *srcIndex*, int *dstIndex*);

### Parameters

*pSrc* Buffer object to copy from  
*srcIndex* Source buffer resource index  
*dstIndex* Destination buffer resource index in the current object

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Copies the contents of a single buffer resource from a source buffer object to the current object. If no source index is specified, the current source buffer index is assumed. If no destination index is specified, the current destination buffer index is assumed.

When the source buffer is larger than the destination buffer in the current object, only the section of the source that fits into the destination is copied.

If the source and destination buffer objects have different formats, automatic data conversion takes place whenever possible.

For multiformat buffer types (for example, SapFormatRGB888\_MONO8 or RGB161616\_MONO16) the copy function can be used to extract either the RGB or mono component to a MONO8/RGB888/RGB8888 or MONO16/RGB161616/RGB16161616 buffer.

### Demo/Example Usage

Not available

---

## SapBuffer::CopyAll

BOOL **CopyAll**(SapBuffer\* *pSrc*);

### Parameters

*pSrc* Buffer object to copy from

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Copies the contents of all buffer resources from a source buffer object to the current object. If the two have different buffer counts, the smaller of the two counts is used.

If the source and destination buffer objects have different formats, automatic data conversion takes place whenever possible.

### Demo/Example Usage

Not available

---

## SapBuffer::CopyRect

BOOL **CopyRect**(SapBuffer\* *pSrc*, int *srcIndex*, int *xSrc*, int *ySrc*, int *width*, int *height*, int *dstIndex*, int *xDest*, int *yDest*);

### Parameters

*pSrc* Buffer object to copy from  
*srcIndex* Source buffer resource index  
*xSrc* Left coordinate of source rectangle origin  
*ySrc* Top coordinate of source rectangle origin  
*width* Source rectangle width  
*height* Source rectangle height  
*dstIndex* Destination buffer resource index in the current object  
*xDest* Left coordinate of destination rectangle

*yDest* Top coordinate of destination rectangle

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Copies a rectangular area from a single buffer resource to another buffer resource. If the source area is too large for the destination buffer resource in the current object, only the section of the source that fits into the destination is copied.

The source and destination buffer objects must have the same format since there is no automatic data conversion as in the SapBuffer::Copy method.

#### Demo/Example Usage

Not available

---

### SapBuffer::Create

BOOL Create();

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Creates all the low-level Sapera resources needed by the buffer object. If it is used together with an acquisition and a transfer object, then you must call this method after SapAcquisition::Create, but before SapTransfer::Create.

#### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example, Grab MFC Example

---

### SapBuffer::Destroy

BOOL Destroy();

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Destroys all the low-level Sapera resources needed by the buffer object. Always call this method after SapTransfer::Destroy.

#### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example, Grab MFC Example

---

### SapBuffer::GetAddress

```
BOOL GetAddress(void** pData);
BOOL GetAddress(void* virtualBaseAddress, void** pData);
BOOL GetAddress(int index, void** pData);
BOOL GetAddress(int index, void* virtualBaseAddress, void** pData);
BOOL GetAddress(int offset, int size, void** pData);
BOOL GetAddress(int offset, int size, void* virtualBaseAddress, void** pData);
BOOL GetAddress(int index, int offset, int size, void** pData);
BOOL GetAddress(int index, int offset, int size, void* virtualBaseAddress, void** pData);
```

#### Parameters

*pData* Pointer to returned buffer data address

*virtualBaseAddress* Starting address of a memory area already reserved by the application

*index* Buffer resource index

*offset* Byte offset from beginning of buffer data for partial mapping

**size** Number of bytes of buffer data to access for partial mapping

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the virtual address where buffer data is stored. Call GetAddress when you need to process buffers in the application itself. Since the Read and Write methods are too slow for this purpose, you need direct access through a pointer. In order to correctly interpret the raw data, you also need to use some or all of the following methods: GetWidth, GetHeight, GetFormat, GetPixelDepth, GetBytesPerPixel, and GetPitch.

Accessing buffer data in video memory may be very slow. In this case, you must call the ReleaseAddress method as soon as possible when you are finished, since getting the address prevents the display hardware from accessing buffer data. This may result in image display problems.

When dealing with buffers that are TypeUnmapped or TypeScatterGatherUnmapped, you should call the ReleaseAddress method as soon as possible when you are done. Getting the data address causes the actual physical to virtual memory mapping to occur. Releasing the address ends the memory mapping and may prevent exhaustion of virtual memory resources in the operating system.

When dealing with very large buffers, you may want to map the buffer data area one section at a time, since fully mapping a very large amount of memory can consume a large amount of system resources. In this case, use the offset and size arguments to specific the partial area to map, and call the ReleaseAddress method before mapping another section.

If you need control over the addresses where the buffer mapping occurs, then use the *virtualBaseAddress* argument. It allows you to specify an address of memory that has already been reserved by the application as the base address for memory mapping.

For buffer types other than those mentioned above, you do not need to call ReleaseAddress after accessing buffer data.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Color Split Example

---

## SapBuffer::GetBufName

const char\* **GetBufName()**;

### Remarks

Gets the name of a buffer object that is shared between multiple processes. If the SapBuffer object was not created using one of the constructors with shared buffers, the value of this attribute is an empty string.

### Demo/Example Usage

Not available

---

## SapBuffer::GetBytesPerPixel

int **GetBytesPerPixel()**;

### Remarks

Gets the number of bytes required to store a single buffer element of all the buffer resources.

You can only read the value of this attribute after calling the Create method.

### Demo/Example Usage

Not available

---

## SapBuffer::GetCapability

BOOL **GetCapability**(int *cap*, void\* *pValue*);  
BOOL **GetCapability**(int *index*, int *cap*, void\* *pValue*);

### Parameters

*cap* Low-level Sapera C library capability to read

*pValue* Pointer to capability value to read back

*Index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

This method allows direct read access to low-level Sapera C library capabilities for the buffer module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer. If no index is specified, the current buffer index is assumed.

Note that you will rarely need to use GetCapability. The Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

### Demo/Example Usage

Not available

---

## SapBuffer::GetCount, SapBuffer::SetCount

```
int GetCount();
BOOL SetCount(int count);
```

### Remarks

Gets/sets the number of buffer resources. The initial value for this attribute is 1, unless you specify another value in the constructor.

You can only call SetCount before the Create method.

### Demo/Example Usage

Sequential Grab Demo, GigE Sequential Grab Demo, Color Split Example

---

## SapBuffer::GetCounterStamp

```
BOOL GetCounterStamp(int* pCounterStamp);
BOOL GetCounterStamp (int index, int* pCounterStamp);
```

### Parameters

*pCounterStamp* Pointer to the returned counter value for the specified buffer resource  
*index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets a unique value associated with a buffer resource. This value is normally expressed in microseconds. It has no meaning by itself; however, subtracting counter stamp values for two buffer resources gives the amount of time elapsed between a common reference point for their respective data transfers.

Even though the returned value is a signed integer, you should convert it to an unsigned integer before using it, since the actual hardware timestamp is unsigned. This is especially important if you need to compare counter stamp values from two different buffers. The counter stamp value may also be expressed in other units. See the SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase method for details.

Note that some transfer devices do not support this feature.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Sequential Grab Demo

---

## SapBuffer::GetFormat, SapBuffer::SetFormat

```
SapFormat GetFormat();
BOOL SetFormat(SapFormat format);
```

### Remarks

Gets/sets the format of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is SapFormatUnknown, and is then set correctly from the transfer node object after calling the Create method.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer object with a starting index and count, then this value is SapFormatUnknown. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to SapFormatMono8.

You can only call SetFormat before the Create method. See the SapBuffer constructor for possible values for *format* (other than SapFormatUnknown).

#### Demo/Example Usage

Not available

---

### SapBuffer::GetFrameRate, SapBuffer::SetFrameRate

```
float GetFrameRate();  
void SetFrameRate(float frameRate);
```

#### Remarks

Gets/sets the frame rate in the buffer object. This value is used when loading or saving a sequence of buffers from/to a file (for example in AVI format).

When loading a buffer sequence the frame rate is restored from the file and can then be obtained through a call to *GetFrameRate*.

When saving a buffer sequence you may optionally save the frame rate. To do so you must specify the frame rate using the *SetFrameRate* function before saving the file. Note that in such a case the you must compute the frame rate yourself.

The frame rate information is irrelevant when the file format does not support sequences of buffers (for example BMP or TIFF formats).

#### Demo/Example Usage

Sequential Grab Demo, GigE Sequential Grab Demo

---

### SapBuffer::GetHandle, SapBuffer::operator[]

```
CORHANDLE GetHandle();  
CORHANDLE GetHandle(int index);  
CORHANDLE operator[](int index);
```

#### Parameters

*index* Index of the required buffer resource handle

#### Remarks

Returns the low-level Sapera handle of the specified buffer resource, which you may then use from low-level Sapera functionality. If no index is specified, the current buffer index is assumed. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

#### Demo/Example Usage

Not available

---

### SapBuffer::GetHandles

```
CORHANDLE* GetHandles();
```

#### Remarks

Gets the low-level Sapera handles of all the buffer resources, which you may then use from low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

#### Demo/Example Usage

Not available

---

### SapBuffer::GetHeight, SapBuffer::SetHeight

```
int GetHeight();  
BOOL SetHeight(int height);
```

#### Remarks

Gets/sets the height of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is 0, and is then set correctly from the transfer node object after calling the Create method. In this case, calling SetHeight has no effect, as the height from the SapXferNode object always takes precedence.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer object with a starting index and count, then this value is 0. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to 480.

You can only call SetHeight before the Create method.

### Demo/Example Usage

Dual Acquisition Demo, Color Split Example, File Load MFC Example, GigE Auto-White Balance Example, Grab MFC Example

---

## SapBuffer::GetHostCounterStamp

```
BOOL GetHostCounterStamp(UINT64* pCounterStamp);  
BOOL GetHostCounterStamp (int index,UINT64* pCounterStamp);
```

### Parameters

*pCounterStamp* Pointer to the returned counter value for the specified buffer resource

*index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Host counter timestamp at which a specific event occurred, such as the end or start of frame. This value is determined by the timebase of the CPU clock. Subtracting counter stamp values for two buffers gives the amount of time elapsed between a common reference point for their respective data transfers.

Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

Note, the CPU clock is common to all applications and devices on the PC. For example, if you have several Teledyne DALSA boards installed, they all refer to the same CPU clock.

### Demo/Example Usage

Not available

---

## SapBuffer::GetNumPages

```
BOOL GetNumPages(int *pNumPages);
```

### Parameters

*pNumPages* Pointer to the returned number of pages

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the number of pages in the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes (pages), instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8) 8-bit HSI planar (SapFormatHSIP8), or multiformat (SapFormatRG888\_MONO8 or SapFormatRGB161616\_MONO16).

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively.

Note that all methods that access an individual buffer resource in the SapBuffer class use the current index when none is specified.

### Demo/Example Usage

Not available

---

## SapBuffer::GetIndex, SapBuffer::SetIndex

```
int GetIndex();
BOOL SetIndex(int index);
```

### Parameters

*index* Buffer resource index

### Remarks

Gets/sets the index of the current buffer. The value of this attribute is set to the last buffer resource after calling the Create method. It is then automatically set by the SapTransfer class to the last acquired buffer through the Next method.

Note that all methods that access an individual buffer resource in the SapBuffer class use the current index when none is specified.

### Demo/Example Usage

Sequential Grab Demo, GigE Sequential Grab Demo

---

## SapBuffer::GetPage

```
BOOL GetPage(int *pPage);
BOOL GetPage(int index, int *pPage);
```

### Parameters

*pPage* Pointer to the returned page number

*index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the active page (or plane) of the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes, instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8) or 8-bit HSI planar (SapFormatHSIP8).

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Not available

---

## SapBuffer::GetPageFormat

```
BOOL GetPageFormat(SapFormat *pageFormat);
```

### Parameters

*pageFormat* Pointer to the returned SapFormat

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the buffer format of the active page (or plane) of the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes, instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8) or 8-bit HSI planar (SapFormatHSIP8).

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Not available

---

## SapBuffer::GetParameter, SapBuffer::SetParameter

```
BOOL GetParameter(int param, void* pValue);
BOOL GetParameter(int index, int param, void* pValue);
BOOL SetParameter(int param, int value);
BOOL SetParameter(int index, int param, int value);
BOOL SetParameter(int param, void* pValue);
BOOL SetParameter(int index, int param, void* pValue);
```

#### Parameters

*param* Low-level Sapera C library parameter to read or write  
*pValue* Pointer to parameter value to read back or to write  
*index* Buffer resource index  
*value* New parameter value to write

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the buffer module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32 bits of storage. If no index is specified, the current buffer index is assumed.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Sequential Grab Demo, GigE Sequential Grab Demo

---

### SapBuffer::GetParameters, SapBuffer::SetParameters

```
void GetParameters(int* count, int* width, int* height, SapFormat* format, SapBuffer::Type* type);
BOOL SetParameters(int count, int width, int height, SapFormat format, SapBuffer::Type type);
BOOL SetParameters(int count, void* virtAddress[], int width, int height, SapFormat format,
SapBuffer::Type type);
BOOL SetParameters(int count, int* physAddress[], int width, int height, SapFormat format,
SapBuffer::Type type);
```

#### Remarks

Gets/sets the count, width, height, format, type of all the buffer resources. You can also set the virtual and physical addresses to use when creating buffer resources.

You can only call SetParameters before the Create method. See the SapBuffer constructor for possible values for more details.

#### Demo/Example Usage

Dual Acquisition Demo, Color Split Example

---

### SapBuffer::GetPitch

```
int GetPitch();
```

#### Remarks

Gets the number of bytes between two consecutive lines of all the buffer resources. This is usually equal to the number of bytes per line, with possible exceptions for buffers located in video memory.

You can only read the value of this attribute after calling the Create method.

#### Demo/Example Usage

Not available

---

### SapBuffer::GetPixelDepth, SapBuffer::SetPixelDepth

```
int GetPixelDepth();
void SetPixelDepth(int pixelDepth);
```

## Remarks

Gets/sets the number of significant bits of all the buffer resources. The range of possible values is given by the SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax method.

The value of this attribute is only relevant after calling the Create method, during which it is set in one of the following ways, depending on which SapBuffer constructor was used.

If using a constructor with a SapXferNode object, the value is set from the pixel depth of this object.

Otherwise, the value is set according to the current buffer data format.

## Demo/Example Usage

Grab LUT Example

---

### SapBuffer::GetSpaceUsed

```
BOOL GetSpaceUsed(int* pSpaceUsed);  
BOOL GetSpaceUsed (int index, int* pSpaceUsed);
```

#### Parameters

*pSpaceUsed* Pointer to the returned space used value for the specified buffer resource  
*index* Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the actual number of data bytes stored in a buffer resource after acquiring an image. This value is usually equal to the buffer size, which indicates that the transfer was successful. If it is equal to 0, it means that no data has been acquired yet.

If this value is non-zero, but less than the buffer size, this can indicate some kind of data transfer error. In this case, monitoring of acquisition and transfer events can give more information about the error.

This value can also be smaller than the buffer size when acquiring variable length data streams.

Also note that this value can also sometimes be equal to the buffer size, even if errors occurred during acquisition. In this case, monitoring of acquisition and transfer events can help identify possible errors.

If no buffer index is specified, the current index is assumed.

## Demo/Example Usage

Not available

---

### SapBuffer::GetState

```
BOOL GetState(SapBuffer::State* pState);  
BOOL GetState(int index, SapBuffer::State* pState);
```

#### Parameters

*pState* Pointer to the returned buffer state, which may be one of the following:  
SapBuffer::StateEmpty The buffer is ready to receive new data  
SapBuffer::StateFull The buffer contains unprocessed data  
SapBuffer::StateOverflow The buffer contains incorrect data due to insufficient hardware bandwidth. This state can only occur when StateFull is active (the two values are combined using a bitwise OR).  
*index* Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the current buffer state that indicates whether the specified buffer is ready to accept a new image, or currently contains unprocessed data.

If no buffer index is specified, the current index is assumed.

Note that Sapera automatically manages the buffer state by default, so that you rarely have to call GetState directly. If you wish to perform this management yourself, you must first call SapTransfer::SetAutoEmpty.

## Demo/Example Usage

Not available

---

## SapBuffer::GetType, SapBuffer::SetType

```
SapBuffer::Type GetType();
BOOL SetType(SapBuffer::Type type);
```

### Remarks

Gets/sets the type of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with physical addresses, then this value is TypeContiguous.

If using the constructor with virtual addresses, then this value is TypeScatterGather.

If using the constructor with a shared buffer object with width/height/format, then this value is also TypeScatterGather.

If using the constructor with a shared buffer object with a starting index and count, then this value is TypeVirtual.

Otherwise, the initial value is equal to TypeDefault. It is then set to a valid value (almost always TypeScatterGather) after calling the Create method

You can only call SetType before the Create method. See the SapBuffer constructor for possible values for type.

### Demo/Example Usage

Color Split Example, File Load MFC Example

---

## SapBuffer::GetWidth, SapBuffer::SetWidth

```
int GetWidth();
BOOL SetWidth(int width);
```

### Remarks

Gets/sets the width of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is 0, and is then set correctly from the transfer node object after calling the Create method. In this case, calling SetWidth has no effect, as the width from the SapXferNode object always takes precedence.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer with a starting index and count, then this value is 0. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to 640.

You can only call SetWidth before the Create method.

### Demo/Example Usage

Color Split Example, File Load MFC Example, GigE Auto-White Balance Example, Grab MFC Example

---

## SapBuffer::IsBufferTypeSupported

```
static BOOL IsBufferTypeSupported(int serverIndex, SapBuffer::Type bufType);
static BOOL IsBufferTypeSupported(const char* serverName, SapBuffer::Type bufType);
static BOOL IsBufferTypeSupported (SapLocation loc, SapBuffer::Type bufType);
```

### Parameters

serverIndex	Index of Sapera server containing the acquisition resource
bufType	Type of buffer to check, see the SapBuffer constructor for a list of possible values
serverName	Name of Sapera server containing the acquisition resource
loc	Valid SapLocation object for the acquisition resource

### Return Value

Returns TRUE if the specified buffer type is supported, FALSE otherwise

### Remarks

Checks if an acquisition resource supports data transfers to a specific buffer type.

For most acquisition hardware, this functionality is not implemented, so it is not possible to determine if the buffer type is supported, and this method returns TRUE. In this case, an error will be returned when calling the SapTransfer::Create or SapTransfer::Connect method when trying to set up a transfer to an unsupported buffer type.

#### Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigESequential Grab Demo, Grab Demo, Sequential Grab Demo

---

### SapBuffer::IsCapabilityValid

**BOOL IsCapabilityValid(int cap);**

#### Parameters

*cap* Low-level Sapera C library capability to check

#### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library capability for the buffer module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapBuffer class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

Not available

---

### SapBuffer::IsMapped

**BOOL IsMapped();**  
**BOOL IsMapped(int index);**

#### Parameters

*index* Buffer resource index

#### Return Value

Returns TRUE if there currently exists a valid virtual memory address for the specified buffer resource, FALSE otherwise

#### Remarks

This method is only relevant for buffers that are TypeUnmapped or TypeScatterGatherUnmapped. In this case, the GetAddress method sets up a valid virtual address mapping, and ReleaseAddress ends it. For all other buffer types, it always returns TRUE.

If no buffer index is specified, the current index is assumed.

#### Demo/Example Usage

Not available

---

### SapBuffer::IsMultiFormat

**BOOL IsMultiFormat();**

#### Return Value

Returns TRUE if the buffer resources are multiformat, FALSE otherwise.

#### Remarks

Multiformat buffers (for example, SapFormatRGB888\_MONO8 or SapFormatRGB161616\_MONO16) contain two formats within the same buffer, such as RGB and monochrome. Typically, depending on the acquisition device output, a multiformat buffer contains two images, one with color data and one with IR data.

Use the GetPage, SetPage and SetAllPage functions to manage the current page of the buffer. This only applies when choosing what format to display when calling the SapView::Show function.

#### Demo/Example Usage

Not available

---

## SapBuffer::IsParameterValid

BOOL **IsParameterValid**(int *param*);

### Parameters

*param* Low-level Sapera C library parameter to check

### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

### Remark

Checks for the availability of a low-level Sapera C library parameter for the buffer module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapBuffer class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapBuffer::Load

BOOL **Load**(const char\* *fileName*, int *bufIndex*, int *numBuffers* = 0, int *frameIndex* = 0, const char\* *options* = "-format auto");

### Parameters

*fileName* Name of the image file to load

*bufIndex* Index of the buffer (or first buffer) in which to load, where -1 is equivalent to the current index.

*numBuffers* Maximum number of buffers to load when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.

*frameIndex* Index of first image frame to load when the file contains a sequence

*options* String containing the loading options. The following are supported:

"-format bmp"	Window bitmap format
"-format tiff"	TIFF format
"-format jpeg"	JPEG format
"-format jpeg_2000-component [value]"	JPEG 2000 format. When loading into a monochrome buffer, specify which color component to load (0 for red, 1 for green, 2 for blue); otherwise this argument is ignored.
"-format crc"	Teledyne DALSA proprietary format
"-format raw -width [value]-height [value] -o [offset]"	Raw data format. You must specify the image width and height, as well as the offset of image data from the beginning of the file.
"-format avi"	AVI image sequence format
"-format auto"	Automatic format detection

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Loads an image file into the current buffer. If no *options* are specified, the format is automatically detected. If the format is AVI, you may use *frameIndex* to specify the first frame to load from the file. If *numBuffers* is 0, the number of frames loaded will not exceed the buffer count.

If the buffer object was constructed using the same *fileName* (see the SapBuffer constructor), no data conversion will be performed since the buffer is compatible with the file.

However, if the buffer was not constructed this way, you must first use the SetParametersFromFile method to make certain that the buffer object is compatible with the file.

### Demo/Example Usage

Color Split Example, File Load Console Example

---

## SapBuffer::MergeComponents

```
BOOL MergeComponents(SapBuffer* pSrc,);
BOOL MergeComponents(SapBuffer* pSrc, int dstIndex);
BOOL MergeComponents(SapBuffer *pFirstSrc, SapBuffer *pSecondSrc, SapBuffer *pThirdSrc);
BOOL MergeComponents(SapBuffer *pFirstSrc, SapBuffer *pSecondSrc, SapBuffer *pThirdSrc, int dstIndex);
```

### Parameters

*pSrc* Source monochrome buffer object  
*pFirstSrc* First source buffer.  
*pSecondSrc* Second source buffer.  
*pThirdSrc* Third source buffer.  
*dstIndex* Destination buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Combines the individual monochrome components from the first three buffer resources of the source object into the color buffer resource at *dstIndex* in the current object. Three monochrome buffer objects can also be merged. If no destination buffer index is specified, the current value is assumed.

The destination and source buffer dimensions must be equal. The output buffer format can be either RGB or YUV. See the SapBuffer constructor for a list of valid RGB and YUV formats.

If the output buffer format is RGB, then the three input buffer resources must contain the red, green, and blue components, respectively. If the output buffer format is YUV, then the three input buffer resources must contain the Y, U, and V components, respectively.

If individual color components have 8-bits or less, then the input format must be SapFormatMono8. If color components have more than 8-bits, then the input format must be SapFormatMono16.

For multiformat buffers (for example, SapFormat.RGB888\_MONO8 or tRGB161616\_MONO16), the function prototype with 3 destination buffers is used to merge 2 source buffers, the RGB and mono components (RGB888/MONO8 or RGB161616/MONO16) respectively into the current buffer object; the 3rd source buffer is ignored.

### Demo/Example Usage

Color Split Example

---

## SapBuffer::Next

```
void Next();
```

### Remarks

Increments the current buffer index. The SapTransfer class calls Next each time an image is acquired to a buffer. The index wraps around to 0 when it reaches the end of the resource array. It always points to the last acquired buffer.

### Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

---

## SapBuffer::Read

```
BOOL Read(UINT64 offset, int numElements, void* pData);
BOOL Read(int index, UINT64 offset, int numElements, void* pData);
```

### Parameters

*offset* Starting position within the buffer (in pixels)  
*numElements* Number of pixels to read  
*pData* Destination memory area for pixel values  
*index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Reads a consecutive series of elements (pixels) from a buffer resource, ignoring line boundaries.

For 1-bit data buffers, the offset must be a multiple of 8, and the memory area must have at least  $((numElements + 7) >> 3)$  bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Note that reading elements from video memory buffers may be very slow.

#### Demo/Example Usage

Not available

---

### SapBuffer::ReadElement

```
BOOL ReadElement(int x, int y, void* pData);
BOOL ReadElement(int index, int x, int y, void* pData);
BOOL ReadElement(int x, int y, SapData* pData);
BOOL ReadElement(int index, int x, int y, SapData* pData);
```

#### Parameters

*x* Horizontal position  
*y* Vertical position  
*pData* Pointer to a destination memory area for the pixel value, or to one of the SapData wrapper classes for Sapera data elements described in this document  
*index* Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reads a single element (pixel) from a buffer resource.

If using one of the first two forms of ReadElement, the memory area must have a number of bytes larger than or equal to the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Reading elements from video memory buffers may be very slow.

Multiformat buffers (for example, SapFormatRGB888\_MONO8 or RGB161616\_MONO16) use a SapDataRGB object. Function prototypes using a 'void\*' data argument use values formatted as B/G/R/Mono. For RGB888\_MONO8 buffers, this is a 32-bit value. For RGB161616\_MONO16 buffers, this is a 64-bit value.

#### Demo/Example Usage

Not available

---

### SapBuffer::ReadLine

```
BOOL ReadLine(int x1, int y1, int x2, int y2, void* pData, int* numRead);
BOOL ReadLine(int index, int x1, int y1, int x2, int y2, void* pData, int* numRead);
```

#### Parameters

*x1* Starting horizontal position  
*y1* Starting vertical position  
*x2* Ending horizontal position  
*y2* Ending vertical position  
*pData* Destination memory area for pixel values  
*numRead* Returns the number of pixels read along the line  
*index* Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reads one line of buffer elements, from position (*x1,y1*) to position (*x2,y2*). Diagonal lines are supported.

The memory area must have a number of bytes larger than or equal to the line length times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

This method does not support 1-bit buffers.

Reading elements from video memory buffers may be very slow.

#### Demo/Example Usage

Not available

---

### SapBuffer::ReadRect

```
BOOL ReadRect(int x, int y, int width, int height, void* pData);  
BOOL ReadRect(int index, int x, int y, int width, int height, void* pData);
```

#### Parameters

<i>x</i>	Left coordinate of rectangle origin
<i>y</i>	Top coordinate of rectangle origin
<i>width</i>	Rectangle width
<i>height</i>	Rectangle height
<i>pData</i>	Destination memory area for pixel values
<i>index</i>	Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reads a rectangular region of elements (pixels) from a buffer resource.

For 1-bit data buffers, *x* and *width* must be multiples of 8, and the memory area must have at least  $((numElements + 7) \gg 3)$  bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Reading elements from video memory buffers may be very slow.

#### Demo/Example Usage

Not available

---

### SapBuffer::ReleaseAddress

```
BOOL ReleaseAddress(void* pData);  
BOOL ReleaseAddress(int index, void* pData = NULL);
```

#### Parameters

<i>pData</i>	Buffer data address to release
<i>index</i>	Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Ends direct buffer data access through a pointer.

When dealing with buffers located in video memory, you must call ReleaseAddress as soon as possible after GetAddress; otherwise, you may encounter image display problems, since getting the address prevents the display hardware from accessing buffer data.

When dealing with buffers that are TypeUnmapped or TypeScatterGatherUnmapped, you should call ReleaseAddress as soon as possible when you are finished with direct data access. Calling the GetAddress method causes the actual physical to virtual memory mapping to occur. Releasing the address ends the memory mapping and may prevent exhaustion of virtual memory resources in the operating system.

For buffer types other than those mentioned above, you do not need to call ReleaseAddress after accessing buffer data.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Not available

---

## SapBuffer::ResetIndex

void **ResetIndex()**;

### Remarks

Initializes the current buffer index to the last buffer resource, so that it will be equal to 0 after the next call to the Next method (from the SapTransfer class). This means that the first buffer resource will then be the current one.

Note that ResetIndex may be called automatically by the SapTransfer::Init method, if you set its optional argument to TRUE.

### Demo/Example Usage

Not available

---

## SapBuffer::Save

BOOL **Save**(const char\* *fileName*, const char\* *options*, int *bufIndex* = -1, int *numBuffers* = 0);

### Parameters

<i>fileName</i>	Name of the image file to save
<i>options</i>	String containing the saving options. The following are supported: "-format bmp" Window bitmap format "-format tiff" TIFF format. Compression may be set to none, run-length encoding, Lempel-Ziv-Welch, or JPEG. For the latter, you may also set a quality level. "-compression [none/rle/lzw/jpeg]" "-quality [value]" "-format jpeg" JPEG format. The quality level may vary between 1 and 100. "-quality [value]" "-format jpeg_2000" JPEG 2000 format. The quality level may vary between 1 and 100, where the latter specifies lossless compression. "-format crc" Teledyne DALSA proprietary format "-format raw" Raw data format "-format avi" AVI image sequence format
<i>bufIndex</i>	Index of the first buffer to save when the file contains a sequence, where a value of -1 is equivalent to the first buffer. If the file contains only one image, then this is the index of the buffer resource to save and -1 is equivalent to the current index.
<i>numBuffers</i>	Number of buffers to save when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Saves one or more buffers to an image file.

If the format is AVI, use the *bufIndex* and *numBuffers* arguments to specify the first buffer and the number of buffers to save. When saving to a file with any other format, the *numBuffers* argument is ignored. The maximum supported size for AVI files is 2 Gbytes.

Note, multiformat buffers, such as SapFormatRGB888\_MONO8 and RGB161616\_MONO16, only support saving in CRC or RAQ format.

### Demo/Example Usage

Not available

---

## SapBuffer::SetAllPage

BOOL **SetAllPage**(int *page*);

### Parameters

*page* New page number for the buffer resources

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Sets the active page (or plane) for all buffer resources in the current object. See also the [SetPage](#) method.

You can only change the value of th property before calling the [Create](#) method.

### Demo/Example Usage

Not available

---

## SapBuffer::SetAllState

BOOL **SetAllState**(SapBuffer::State *state*);

### Parameters

*state* New state for the buffer resources. See SapBuffer::GetState method for possible values.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Sets the current state for all buffer resources in the current object. See also the SapBuffer::SetState method.

### Demo/Example Usage

Not available

---

## SapBuffer::SetPage

BOOL **SetPage**(int *page*);

BOOL **SetPage**(int *index*, int *page*);

### Parameters

*page* New page number for the buffer resource

*index* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Sets the active page (or plane) of the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes, instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8), 8-bit HSI planar (SapFormatHSIP8) or multiformat buffer types such as SapFormat:RGB888\_MONO8 or RGB161616\_MONO16.

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively. For multiformat buffers, 2 pages are used; one for the color part and one for the mono (IR) part.

If no buffer index is specified, the current index is assumed.

### Demo/Example Usage

Not available

---

## SapBuffer::SetParametersFromFile

**BOOL SetParametersFromFile(const char\* fileName, SapBuffer::Type type);**

**Parameters**

*fileName* Name of a Sapera image file from which to extract buffer attributes

*type* Type of buffer resources. See the SapBuffer constructor for details.

**Remarks**

Sets the count, width, height, format, and type of all the buffer resources from an existing Sapera image file to ensure buffer compatibility.

You can only call SetParametersFromFile before the Create method. You can then use the Load method after calling Create.

See the SapBuffer constructor for possible values for *type*.

**Demo/Example Usage**

Color Split Example

---

### SapBuffer::SetPhysicalAddress

**BOOL SetPhysicalAddress(ULONG\_PTR physAddress[]);**

**Parameters**

*physAddress* Array of physical addresses to use when creating buffer resources. See the SapBuffer constructor for more details.

**Remarks**

Sets the physical addresses to use for creating buffer resources.

You can only call SetPhysicalAddress before the Create method.

**Demo/Example Usage**

Color Split Example

---

### SapBuffer::SetState

**BOOL SetState(SapBuffer::State state);**

**BOOL SetState(int index, SapBuffer::State state);**

**Parameters**

*state* New state for the buffer resource. See the SapBuffer::GetState method for possible values.

*index* Buffer resource index

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Sets the current buffer state that indicates whether the specified buffer is ready to accept a new image, or currently contains unprocessed data.

If no buffer index is specified, the current index is assumed.

Note that Sapera automatically manages the buffer state by default, so that you rarely have to call SetState directly. If you wish to perform this management yourself, you must first call SapTransfer::SetAutoEmpty.

**Demo/Example Usage**

Not available

---

### SapBuffer::SetVirtualAddress

**BOOL SetVirtualAddress(void\* virtAddress[]);**

**Parameters**

*virtAddress* Array of virtual addresses to use when creating buffer resources. See the SapBuffer constructor for more details.

**Remarks**

Sets the virtual addresses to use for creating buffer resources.

You can only call SetVirtualAddress before the Create method.

**Demo/Example Usage**

---

## SapBuffer::SplitComponents

```
BOOL SplitComponents(SapBuffer* pSrc);
BOOL SplitComponents(SapBuffer* pSrc, int srcIndex);
BOOL SplitComponents(SapBuffer *pFirstDst, SapBuffer *pSecondDst, SapBuffer *pThirdDst);
BOOL SplitComponents(SapBuffer *pFirstDst, SapBuffer *pSecondDst, SapBuffer *pThirdDst, int srcIndex);
```

**Parameters**

*pSrc* Source color buffer object  
*pFirstDst* First destination buffer object.  
*pSecondDst* Second destination buffer object.  
*pThirdDst* Third destination buffer object.  
*srcIndex* Source buffer resource index

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Splits the color buffer resource at *srcIndex* into its individual monochrome components in the first three buffer resources of the current object. The color buffer can also be extracted into three separate monochrome buffer objects. If no source buffer index is specified, the current value is assumed.

The destination buffer dimensions (in the current object) must be equal to or larger than the source buffer object dimensions. The input buffer format can be either RGB or YUV. See the SapBuffer constructor for a list of valid RGB and YUV formats.

If the input buffer format is RGB, then the three output buffer resources will contain the red, green, and blue components, respectively. If the input buffer format is YUV, then the three output buffer resources will contain the Y, U, and V components, respectively.

If individual color components have 8-bits or less, then the output format (in the current buffer object) must be SapFormatMono8. If color components have more than 8-bits, then the output format must be SapFormatMono16.

For multiformat buffers (for example, SapFormatRGB888\_MONO8 or SapFormatRGB161616\_MONO16) the source buffer is the current object; the function prototype with 3 destination buffers is used to extract the RGB and mono components (RGB888/MONO8 or RGB161616/MONO16) into the first 2 buffer objects; the 3<sup>rd</sup> destination buffer is ignored. If no source buffer index is specified, the current value is assumed.

**Demo/Example Usage**

Color Split Example

---

## SapBuffer::Write

```
BOOL Write(UINT64 offset, int numElements, void* pData);
BOOL Write(int index, UINT64 offset, int numElements, void* pData);
```

**Parameters**

*offset* Starting position within the buffer (in pixels)  
*numElements* Number of pixels to write  
*pData* Source memory area for pixel values  
*index* Buffer resource index

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Writes a consecutive series of elements (pixels) to a buffer resource, ignoring line boundaries.

For 1-bit data buffers, the offset must be a multiple of 8, and the memory area must have at least  $((numElements + 7) >> 3)$  bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes of at least *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

### Demo/Example Usage

Not available

---

### SapBuffer::WriteElement

```
BOOL WriteElement(int x, int y, const void* pData);
BOOL WriteElement(int index, int x, int y, const void* pData);
BOOL WriteElement(int x, int y, SapData data);
BOOL WriteElement(int index, int x, int y, SapData data);
```

#### Parameters

x	Horizontal position
y	Vertical position
pData	Pointer to a source memory area for the pixel value
data	Pixel value represented by one of the SapData wrapper classes for Sapera data elements described in this document
index	Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Writes a single element (pixel) to a buffer resource.

If using one of the first two forms of WriteElement, the memory area must have a number of bytes equal or larger than the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

Multiformat buffers (for example, SapFormatRGB888\_MONO8 or RGB161616\_MONO16) use a SapDataRGB object. Function prototypes using a 'void\*' data argument use values formatted as B/G/R/Mono. For 8-bit buffers, this is a 32-bit value. For 16-bit buffers, this is a 64-bit value.

### Demo/Example Usage

Not available

---

### SapBuffer::WriteLine

```
BOOL WriteLine(int x1, int y1, int x2, int y2, const void* pData, int* numWritten);
BOOL WriteLine(int index, int x1, int y1, int x2, int y2, const void* pData, int* numWritten);
```

#### Parameters

x1	Starting horizontal position
y1	Starting vertical position
x2	Ending horizontal position
y2	Ending vertical position
pData	Source memory area for pixel values
numWritten	Returns the number of pixels written along the line
index	Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Writes one line of buffer elements, from position (x1,y1) to position (x2,y2). Diagonal lines are supported.

The memory area must have a number of bytes larger than or equal to the line length times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

WriteLine does not support 1-bit buffers.

Writing elements to video memory buffers may be very slow.

### Demo/Example Usage

Not available

---

### SapBuffer::WriteRect

```
BOOL WriteRect(int x, int y, int width, int height, const void* pData);  
BOOL WriteRect(int index, int x, int y, int width, int height, const void* pData);
```

#### Parameters

<i>x</i>	Left coordinate of rectangle origin
<i>y</i>	Top coordinate of rectangle origin
<i>width</i>	Rectangle width
<i>height</i>	Rectangle height
<i>pData</i>	Source memory area for pixel values
<i>index</i>	Buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Writes a rectangular region of elements (pixels) to a buffer resource.

For 1-bit data buffers, *x* and *width* must be multiples of 8, and the memory area must have at least  $((numElements + 7) \gg 3)$  bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

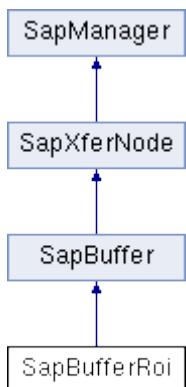
If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

### Demo/Example Usage

Not available

# SapBufferRoi



The purpose of the SapBufferRoi Class is to create a rectangular Region of Interest (ROI) inside an existing SapBuffer object. The ROI has the same origin and dimensions for all buffer resources in the object.

You may create multiple instances of this class using the same SapBuffer as a parent; however, the acquisition hardware dictates the number of maximum simultaneous ROIs when acquiring images.

One typical usage of this class is reducing acquisition bandwidth requirements when only a subset of an image is needed.

```
#include <SapClassBasic.h>
```

## SapBufferRoi Class Members

### Construction

<a href="#"><u>SapBufferRoi</u></a>	Class constructor
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources
<b>Attributes</b>	
<a href="#"><u>GetParent</u></a>	Gets/sets the parent SapBuffer object for the ROI
<a href="#"><u>SetParent</u></a>	
<a href="#"><u>GetRoot</u></a>	Gets the topmost SapBuffer object for the ROI
<a href="#"><u>GetXMin</u></a>	Gets/sets the left origin for the ROI, relative to the parent object
<a href="#"><u>SetXMin</u></a>	
<a href="#"><u>GetYMin</u></a>	Gets/sets the top origin for the ROI, relative to the parent object
<a href="#"><u>SetYMin</u></a>	
<a href="#"><u>GetWidth</u></a>	Gets/sets the width (in pixels) for the ROI
<a href="#"><u>SetWidth</u></a>	
<a href="#"><u>GetHeight</u></a>	Gets/sets the height (in pixels) for the ROI
<a href="#"><u>SetHeight</u></a>	
<a href="#"><u>SetRoi</u></a>	Sets the ROI origin and dimensions in one step
<a href="#"><u>ResetRoi</u></a>	Sets the ROI origin and dimensions to default values
<a href="#"><u>GetTrash</u></a>	Returns the low-level Sapera handle of the trash buffer resource, if any

## SapBufferRoi Member Functions

The following are members of the SapBufferRoi Class.

---

### SapBufferRoi::SapBufferRoi

**SapBufferRoi**(*SapBuffer*\* *pParent*, int *xmin* = 0, int *ymin* = 0, int *width* = -1, int *height* = -1);

#### Parameters

<i>pParent</i>	SapBuffer object that represents the parent for the current SapBufferRoi object
<i>xmin</i>	Left origin for the ROI, relative to the parent object
<i>ymin</i>	Top origin for the ROI, relative to the parent object
<i>width</i>	Width (in pixels) of the ROI
<i>height</i>	Height (in lines) of the ROI

#### Remarks

The SapBufferRoi constructor sets up a rectangular Region of Interest (ROI) inside the SapBuffer object identified by *pParent*. This ROI has the specified origin and dimensions, up to the whole area of the parent object.

A value of -1 for the width/height means that the ROI should have the same width/height as the parent buffer. The constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapBufferRoi::Create

BOOL **Create**();

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Creates all the low-level Sapera resources needed by the current object. Always call this method before SapTransfer::Create.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapBufferRoi::Destroy

BOOL **Destroy**();

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Destroys all low-level Sapera resources used by the current object. Always call this method after SapTransfer::Destroy.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapBufferRoi::GetHeight, SapBufferRoi::SetHeight

int **GetHeight**();

void **SetHeight**(int *height*);

#### Remarks

Gets/sets the height (in lines) for the ROI. If it has not been specified in the constructor, the value of this attribute is set to the parent buffer height when calling the Create method.

You can only call SetHeight before the Create method.

#### Demo/Example Usage

Not available

---

## SapBufferRoi::GetParent, SapBuffer::SetParent

```
SapBuffer* GetParent();  
void SetParent(SapBuffer *pParent);
```

### Remarks

Gets/sets the parent buffer object for the ROI. Note that you can only call SetParent before the Create method.

### Demo/Example Usage

Not available

---

## SapBufferRoi::GetRoot

```
SapBuffer* GetRoot();
```

### Remarks

Gets the topmost SapBuffer object for the ROI.

When there is a one-level ROI hierarchy below the topmost object, then the returned value is the same as for the GetParent method.

When there is a multi-level ROI hierarchy below the topmost object, then the returned value is the equivalent of going up the ROI tree by calling GetParent repeatedly until we reach the top.

### Demo/Example Usage

Not available

---

## SapBufferRoi::GetTrash

```
CORBUFFER GetTrash();
```

### Remarks

If the current object has a SapBufferWithTrash parent object, then this method returns the low-level Sapera handle of the ROI trash buffer resource, which you may then use from the low-level Sapera functionality.

Note that the handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

### Demo/Example Usage

Dual Acquisition Demo

---

## SapBufferRoi::GetWidth, SapBufferRoi::SetWidth

```
int GetWidth();  
void SetWidth(int width);
```

### Remarks

Gets/sets the width (in pixels) for the ROI. If it has not been specified in the constructor, the value of this attribute is set to the parent buffer width when calling the Create method.

You can only call SetWidth before the Create method.

### Demo/Example Usage

Not available

---

## SapBufferRoi::GetXMin, SapBufferRoi::SetXMin

```
int GetXMin();  
void SetXMin(int xmin);
```

### Remarks

Gets/sets the left origin for the ROI, relative to the parent object. The initial value of this attribute is 0 if it was not specified in the constructor.

You can only call SetXMin before the Create method.

### Demo/Example Usage

Not available

---

## SapBufferRoi::GetYMin, SapBufferRoi::SetYMin

```
int GetYMin();
void SetYMin(int ymin);
```

#### Remarks

Gets/sets the top origin for the ROI, relative to the parent object. The initial value of this attribute is 0 if it was not specified in the constructor.

You can only call SetYMin before the Create method.

#### Demo/Example Usage

Not available

---

### SapBufferRoi::ResetRoi

```
BOOL ResetRoi();
```

#### Remarks

Sets the ROI origin and dimensions to default values corresponding to the whole buffer area in the parent object. You can only call ResetRoi before the Create method.

#### Demo/Example Usage

Not available

---

### SapBufferRoi::SetRoi

```
BOOL SetRoi(int xmin, int ymin, int width, int height);
BOOL SetRoi(RECT* pRect);
```

#### Parameters

<i>xmin</i>	Left origin for the ROI, relative to the parent object
<i>ymin</i>	Top origin for the ROI, relative to the parent object
<i>width</i>	Width (in pixels) of the ROI
<i>height</i>	Height (in lines) of the ROI
<i>pRect</i>	Pointer to a Windows RECT structure that specifies the four corners of the ROI

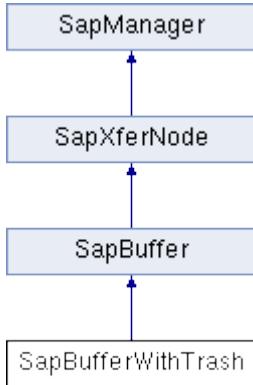
#### Remarks

Sets the ROI origin and dimensions in one step. You can only call SetRoi before the Create method.

#### Demo/Example Usage

Dual Acquisition Demo

# SapBufferWithTrash



The SapBufferWithTrash Class is derived from SapBuffer. It creates an additional resource called the trash buffer used when transferring data in real-time applications.

The trash buffer is an emergency buffer used by the SapTransfer class when the data transfer is faster than a processing task performed on the buffers. When processing is not fast enough to keep up with the incoming data, images are transferred temporarily into the trash buffer until stability is reestablished.

```
#include <SapClassBasic.h>
```

## SapBufferWithTrash Class Members

### Construction

<u><a href="#">SapBufferWithTrash</a></u>	Class constructor
<u><a href="#">Create</a></u>	Allocates the low-level Sapera resources
<u><a href="#">Destroy</a></u>	Releases the low-level Sapera resources
<b>Attributes</b>	
<u><a href="#">GetTrashType</a></u>	Gets/sets the buffer type for the trash buffer resource only
<u><a href="#">SetTrashType</a></u>	
<u><a href="#">GetTrash</a></u>	Returns the low-level Sapera handle of the trash buffer resource

## SapBufferWithTrash Member Functions

The following are members of the SapBufferWithTrash Class.

### SapBufferWithTrash::SapBufferWithTrash

```
SapBufferWithTrash()
    int count = 2,
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeScatterGather,
    SapLocation loc = SapLocation::ServerSystem
);
SapBufferWithTrash(
    int count,
    ULONG_PTR physAddress[]
    int width = 640,
    int height = 480,
    SapFormat format = SapFormatMono8,
    SapBuffer::Type type = SapBuffer::TypeContiguous,
);
SapBufferWithTrash (
```

```

int count,
void* virtAddress[]
int width = 640,
int height = 480,
SapFormat format = SapFormatMono8,
SapBuffer::Type type = SapBuffer::TypeScatterGather,
);
SapBufferWithTrash (
int count,
SapXferNode* pSrcNode,
SapBuffer::Type type = SapBuffer::TypeScatterGather,
SapLocation loc = SapLocation::ServerSystem
);
SapBufferWithTrash(  

const char* fileName,  

SapBuffer::Type type = SapBuffer::TypeScatterGather  

SapLocation loc = SapLocation::ServerSystem  

);
SapBufferWithTrash (  

int count,  

const char* bufName  

int width = 640,  

int height = 480,  

SapFormat format = SapFormatMono8,  

SapBuffer::Type type = SapBuffer::TypeScatterGather,  

SapLocation loc = SapLocation::ServerSystem  

);
SapBufferWithTrash (  

int count,  

const char* bufName  

SapXferNode* pSrcNode,  

SapBuffer::Type type = SapBuffer::TypeScatterGather,  

SapLocation loc = SapLocation::ServerSystem);
SapBufferWithTrash (  

const char* bufName  

int startIndex  

int count,  

SapBuffer::Type type = SapBuffer::TypeVirtual,  

SapLocation loc = SapLocation::ServerSystem
);

```

## Parameters

See the SapBuffer constructor for a description of the parameters

## Remarks

Derived from SapBuffer, the SapBufferWithTrash object contains an additional resource called the trash buffer that has the same attributes (width, height, format, and type) as the other buffer resources.

The *count* argument does not include the trash buffer. Its value cannot be smaller than 2.

The constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

## Demo/Example Usage

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab MFC Example

## **SapBufferWithTrash::Create**

BOOL Create();

## Return Value

Returns TRUE if successful, FALSE otherwise

## **Remarks**

Creates all the low-level Sapera resources needed by the SapBufferWithTrash object. Always call this method before SapTransfer::Create.

## **Demo/Example Usage**

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab MFC Example

---

## **SapBufferWithTrash::Destroy**

**BOOL Destroy();**

## **Return Value**

Returns TRUE if successful destroyed, FALSE otherwise

## **Remarks**

Destroys all low-level Sapera resources needed by the SapBufferWithTrash object. Always call this method after SapTransfer::Destroy.

## **Demo/Example Usage**

Bayer Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab MFC Example

---

## **SapBufferWithTrash::GetTrash**

**CORBUFFER GetTrash();**

## **Remarks**

Returns the low-level Sapera handle of the trash buffer resource, which you may then use from the low-level Sapera functionality. Note that the handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

## **Demo/Example Usage**

Not available

---

## **SapBufferWithTrash::GetTrashType, SapBufferWithTrash::SetTrashType**

**SapBuffer::Type GetTrashType();**  
**void SetTrashType(SapBuffer::Type type);**

## **Remarks**

Gets/sets the buffer type for the trash buffer resource only. This may be useful, for example, if the current transfer device allows the usage of dummy buffers (SapBuffer::TypeDummy) to reduce bandwidth requirements associated with trash buffer transfers.

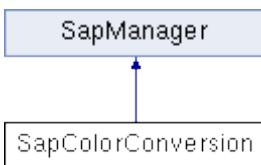
If you do not set a value for this property, then it is set to the same value as the other buffer resources when calling the Create method.

You can only call SetTrashType before the Create method. See the SapBuffer constructor for possible values for *type*.

## **Demo/Example Usage**

Not available

# SapColorConversion



The purpose of the SapColorConversion Class is to support conversion of color images, such as Bayer encoded images or other color formats, to RGB images for output. In the first case, images are acquired from a Bayer, or other supported format, camera. They are then converted to RGB either by the acquisition device (if supported) or through software. In the second case, images are taken from another source (for example, loaded from disk). Only the software implementation is then available.

This class can perform the following operations:

- Apply color conversion on a raw Bayer Mono8/16 input buffer and get a resulting RGB888/8888 output buffer (Methods 1-5)
- Apply color conversion on a raw Bi-Color88/1616 input buffer and get a resulting RGB888/8888 output buffer (Methods 6-7)
- Apply white-balance gain on a RGB/Bayer/Bi-Color input buffer

```
#include <SapClassBasic.h>
```

## SapColorConversion Class Members

### Construction

<u>SapColorConversion</u>	Class constructor
<u>Create</u>	Allocates the internal resources
<u>Destroy</u>	Releases the internal resources
<b>Attributes</b>	
<u>GetAcquisition</u>	Gets/sets the acquisition object for acquiring color images
<u>SetAcquisition</u>	
<u>GetInputBuffer</u>	Gets/sets the buffer object in which images are acquired or loaded
<u>SetInputBuffer</u>	
<u>GetOutputBuffer</u>	Gets the buffer object used as the destination for software conversion
<u>GetOutputBufferCount</u>	Gets/sets the number of buffer resources used for software conversion
<u>SetOutputBufferCount</u>	
<u>.IsEnabled</u>	Checks if color conversion is enabled
<u>IsSoftwareEnabled</u>	Checks if color conversion in software is enabled
<u>IsSoftwareSupported</u>	Checks if the input buffer supports color conversion
<u>IsHardwareEnabled</u>	Checks if color conversion in hardware is enabled
<u>IsHardwareSupported</u>	Checks if the input buffer supports color conversion
<u>GetAlign</u>	Gets/sets the color alignment mode
<u>SetAlign</u>	
<u>GetAvailAlign</u>	Gets the available alignment modes
<u>GetMethod</u>	Gets/sets the pixel value calculation method
<u>SetMethod</u>	
<u>GetAvailMethod</u>	Gets the available pixel value calculation methods
<u>GetWBGain</u>	Gets/sets the white balance gain coefficients
<u>SetWBGain</u>	
<u>GetWBOffset</u>	Gets/sets the white balance offset coefficients
<u>SetWBOffset</u>	

<u>GetGamma</u>	Gets/sets the gamma correction factor for the color lookup table
<u>SetGamma</u>	
<u>GetOutputFormat</u>	Gets/sets the data output format of color conversion
<u>SetOutputFormat</u>	
<u>IsLutEnabled</u>	Gets the current color lookup table enable value
<u>IsAcqLut</u>	Checks if the color lookup table corresponds to the acquisition LUT
<b>Operations</b>	
<u>Enable</u>	Enables/disables color conversion
<u>Convert</u>	Converts a color-encoded image to an RGB image using software
<u>WhiteBalance</u>	Calculates the white balance gain coefficients for color conversion
<u>WhiteBalanceManual</u>	Sets the white balance gain coefficients for color conversion
<u>GetLut</u>	Gets the current color lookup table
<u>EnableLut</u>	Enables/disables the color lookup table

## SapColorConversion Member Functions

The following are members of the SapColorConversion Class.

---

### SapColorConversion::SapColorConversion

**SapColorConversion();**  
**SapColorConversion(SapAcquisition\* pAcq, SapBuffer\* pBuffer);**  
**SapColorConversion(SapBuffer\* pBuffer);**

#### Parameters

*pAcq* SapAcquisition object to use for image acquisition and color conversion (if available in hardware)  
*pBuffer* SapBuffer object in which images will be acquired or loaded

#### Remarks

The SapColorConversion constructor does not actually create the internal resources. To do this, you must call the Create method.

When using hardware conversion, the result will be stored in the buffer object identified by *pBuffer*. When using software conversion, the buffer object for the result of the conversion is automatically created using relevant attributes from *pBuffer*.

In both cases, the resulting SapBuffer object will be available through the GetOutputBuffer method.

#### Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

---

### SapColorConversion::Convert

**BOOL Convert();**  
**BOOL Convert(int srcIndex);**  
**BOOL Convert(int srcIndex, int dstIndex);**

#### Parameters

*srcIndex* Source buffer resource index  
*dstIndex* Destination buffer resource index

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

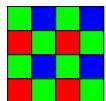
Converts a color-encoded image to an RGB image using software.

The source buffer for the conversion is the current buffer resource in the main buffer object, unless you specify a source index. The GetBuffer method allows you to access this buffer.

The destination buffer for the conversion is the current buffer resource in the internal color buffer object, unless you specify a destination index. The GetOutputBuffer method allows you to access this buffer.

The color format assigns each pixel in a monochrome image the value of one color channel. RGB images are created by using neighboring pixel values to get the two missing color channels at each pixel.

Pixels in one row of a color image alternate between the green channel value and either the red or the blue channel value. The default scheme is shown below.



The missing color channel values are found using neighboring pixel values for the color channel in question by various methods, some of which are more computationally expensive, but give better image quality when the input image contains many strong edges.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::Create

**BOOL Create();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Creates all the internal resources needed by the color conversion object.

If the color conversion object is associated with a SapAcquisition object (using the SapColorConversion constructor or the SetAcquisition method), then you can only call this method after the Create method for the acquisition object.

If there is no acquisition object, then you can only call this method after the Create method for the associated buffer object instead (specified using the SapColorConversion constructor or the SetBuffer method).

#### Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

---

### SapColorConversion::Destroy

**BOOL Destroy();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Destroys all the internal resources needed by the color conversion object

#### Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

---

### SapColorConversion::Enable

**BOOL Enable(BOOL enable = TRUE, BOOL useHardware = TRUE);**

#### Parameters

*enable*      TRUE to enable color conversion, FALSE to disable it

*useHardware*    TRUE to use hardware conversion, FALSE to use the software implementation

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Enables/disables conversion of color images to RGB. If you set *useHardware* to TRUE, and hardware

conversion is not available, then this method returns FALSE. If you set *useHardware* to FALSE, then you must call the Convert method to perform the actual conversion.

Use the [SapAcquisition::IsColorConverionsAvailable](#) method to find out if hardware correction is available in the acquisition device.

### Demo/Example Usage

Color Conversion Demo

---

## SapColorConversion::EnableLut

BOOL **EnableLut**(BOOL *enable* = TRUE);

### Parameters

*enable*      TRUE to enable the color lookup table, FALSE to disable it

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Enables or disables the color lookup table that is applied to image data after color conversion has been performed.

For hardware conversion, this is actually the acquisition lookup table. For software conversion, the lookup table is created automatically inside the SapColorConversion object so that it is compatible with the buffer object on which color conversion is performed.

### Demo/Example Usage

Not available

---

## SapColorConversion::GetAcquisition, SapColorConversion::SetAcquisition

SapAcquisition\* **GetAcquisition**();

BOOL **SetAcquisition**(SapAcquisition\* *pAcq*);

### Remarks

Gets/sets the SapAcquisition object to be used for image acquisition and for color conversion. You can only call SetAcquisition before the Create method.

### Demo/Example Usage

Not available

---

## SapColorConversion::GetAlign, SapColorConversion::SetAlign

SapColorConversion::Align **GetAlign**();

BOOL **SetAlign**(SapColorConversion::Align *align*);

### Parameters

*align*      Color alignment mode may be one of the following values

SapColorConversion::AlignGBRG



SapColorConversion::AlignBGGR



SapColorConversion::AlignRGGB



SapColorConversion::AlignGRBG



SapColorConversion::AlignRGBG



SapColorConversion::AlignBGRG



### Remarks

Gets/sets the color alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of the camera, or 1x4 line for a Bicolor camera.

The initial value for this attribute is SapColorConversion::AlignGRBG. It is then set to the acquisition device

color alignment value when calling the Create method (except when no acquisition device is used).

### Demo/Example Usage

Color Converions Demo, GigE Auto-White Balance Example

---

## SapColorConversion::GetAvailAlign

SapColorConversion::Align **GetAvailAlign()**;

### Remarks

Gets the valid color alignment modes, combined together using bitwise OR.

The initial value for this attribute includes all available modes. It is then set to the valid acquisition device alignment modes when calling the Create method (except when no acquisition device is used).

See the [GetAlign](#) method for a list of possible alignment modes.

### Demo/Example Usage

Color Converions Demo

---

## SapColorConversion::GetAvailMethod

SapColorConversion::Method **GetAvailMethod()**;

### Remarks

Gets the valid color pixel value calculation methods, combined together using bitwise OR.

The initial value for this attribute includes all available methods. It is then set to the valid acquisition device calculation methods when calling the Create method (except when no acquisition device is used).

See the [GetMethod](#) method for a list of possible calculation methods.

### Demo/Example Usage

Color Converions Demo

---

## SapColorConversion::GetGamma, SapColorConversion::SetGamma

float **GetGamma()**;  
BOOL **SetGamma**(float *gamma*);

### Parameters

*gamma*      New gamma correction factor

### Remarks

Gets/sets the color gamma correction factor. If color conversion is enabled, and the color lookup table is also enabled (using the EnableLut method), then Gamma correction with the specified *factor* is applied after color conversion has been performed.

The initial value for this attribute is 1.0, which effectively disables Gamma correction.

### Demo/Example Usage

Color Converions Demo

---

## SapColorConversion::GetInputBuffer, SapColorConversion::SetInputBuffer

SapBuffer \***GetInputBuffer()**;  
BOOL **SetInputBuffer**(SapBuffer \**pBuffer*);

### Remarks

Gets/sets the SapBuffer object in which images will be acquired or loaded.

For software conversion, the buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer object with the result of the conversion is then available by calling the GetOutputBuffer method.

For hardware conversion, the buffer format may be SapFormatRGB888, SapFormatRGB8888, or SapFormatRGB101010 (16-bit input image only). In this case, the buffer object returned by this method is the same as the one returned by calling the [GetInputBuffer](#) method.

You can only call SetBuffer before the Create method.

### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::GetLut

SapLut\* **GetLut()**;

#### Remarks

Gets the current color lookup table that is applied to image data after color conversion has been performed, if the lookup table has been enabled using the EnableLut method.

For hardware conversion, this is actually the acquisition lookup table, which you may also obtain through the SapAcquisition::GetLut method. If the acquisition hardware has no lookup table, then the return value is NULL.

For software conversion, the lookup table is created automatically inside the SapColorConversion object so that it is compatible with the buffer object on which color conversion is performed.

### Demo/Example Usage

Not available

---

### SapColorConversion::GetMethod, SapColorConversion::SetMethod

SapColorConversion::Method **GetMethod()**;

BOOL **SetMethod**(SapColorConversion::Method *method*);

#### Parameters

*method* Color pixel value calculation method may be one of the following values

SapColorConversion::Method1      Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image. Based on a 3x3 neighborhood operation.

See the Remarks section for more information.

SapColorConversion::Method2      Proprietary adaptive technique, better for preserving the edges of the image. However, it works well only when the image has a strong content in green. Otherwise, little amounts of noise may be visible within objects.

SapColorConversion::Method3      Proprietary adaptive technique, almost as good as Method2 for preserving the edges, but independent of the image content in green. Small colour artefacts of 1 pixel may be visible at the edges.

SapColorConversion::Method4      Technique based on 2x2 interpolation. This is the simplest and fastest algorithm. Compared to 3x3 it is better at preserving edge sharpness but introduces a slight jitter in pixel position. In practice it is a good choice for image display but less recommended than 3x3 for accurate image processing.

SapColorConversion::Method5      Technique based on a set of linear filters. It assumes that edges have a much stronger luminance than chrominance component.

SapColorConversion::Method7      Support for the Teledyne DALSA Piranha 4 line scan camera color output. If the appropriate camera firmware is loaded, the driver will return this value as one of the available methods.

#### Remarks

Gets/sets the color pixel value calculation method.

The initial value for this attribute is SapColorConversion::Method1. It is then set to the acquisition device color conversion method when calling the Create method (except when no acquisition device is used).

For SapColorConversion::Method1, four cases are possible according to window position:

G	R	G
B	G	B
G	R	G

$$R = (R[\text{up}] + R[\text{down}]) / 2;$$

$$G = G$$

$$B = (B[\text{left}] + B[\text{right}]) / 2$$

R	G	R
G	B	G
R	G	R

$R = (R[\text{left,up}] + R[\text{right,up}] + R[\text{left,down}] + R[\text{right,down}]) / 4$   
 $G = (G[\text{left}] + G[\text{right}] + G[\text{up}] + G[\text{down}]) / 4$   
 $B = B$

B	G	B
G	R	G
B	G	B

$R = R$   
 $G = (G[\text{left}] + G[\text{right}] + G[\text{up}] + G[\text{down}]) / 4$   
 $B = (B[\text{left,up}] + B[\text{right,up}] + B[\text{left,down}] + B[\text{right,down}]) / 4$

G	B	G
R	G	R
G	B	G

$R = (R[\text{left}] + R[\text{right}]) / 2;$   
 $G = G$   
 $B = (B[\text{up}] + B[\text{down}]) / 2$

### Demo/Example Usage

Color Converions Demo

## SapColorConversion::GetOutputBuffer

**SapBuffer \*GetOutputBuffer();**

### Remarks

Gets the buffer object used as the destination for software conversion. When using software conversion, this object is automatically created using relevant attributes from the main buffer object (the one in which images are acquired or loaded).

When color conversion is performed in hardware, this method returns the same buffer object as the GetBuffer method.

You cannot call GetOutputBuffer before the Create method.

### Demo/Example Usage

Color Conversion Demo

## SapColorConversion::GetOutputBufferCount, SapColorConversion::SetOutputBufferCount

**int GetOutputBufferCount();**  
**BOOL SetOutputBufferCount(int *bufferCount*);**

### Parameters

*bufferCount* Number of buffer resources

### Remarks

Gets/sets the number of buffer resources used for software conversion. The initial value for this attribute is 2.

You can only call SetOutputBufferCount before the Create method.

### Demo/Example Usage

Not available

## SapColorConversion::GetOutputFormat, SapColorConversion::SetOutputFormat

**SapFormat GetOutputFormat();**  
**BOOL SetOutputFormat (SapFormat *format*);**

### Parameters

*format* New color conversion output format

### Remarks

Gets/sets the data output format of color conversion. The only two possible values for this attribute are SapFormatRGB8888 and SapFormatRGB101010.

The initial value for this attribute is SapFormatUnknown. It is then set to the appropriate value when calling

the Create method.

You can only call SetOutputFormat before the Create method.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::GetWBGain, SapColorConversion::SetWBGain

SapDataFRGB **GetWBGain()**;  
BOOL **SetWBGain**(SapDataFRGB *wbGain*);

#### Parameters

*wbGain*      New white balance gain coefficients

#### Remarks

Gets/sets the white balance gain coefficients. These may also be calculated automatically using the WhiteBalance method.

The white balance gain coefficients are the red, green, and blue gains applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all gains to 1.0 if no white balance gain is required.

The initial value for this attribute is 1.0 for each color component.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::GetWBOffset, SapColorConversion::SetWBOffset

SapDataFRGB **GetWBOffset()**;  
BOOL **SetWBOffset**(SapDataFRGB *wbOffset*);

#### Parameters

*wbOffset*      New white balance offset coefficients

#### Remarks

Gets/sets the white balance offset coefficients. These apply only for hardware conversion, that is, when the IsSoftware method returns FALSE.

The white balance offset coefficients are the red, green, and blue offsets applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all offsets to 0.0 if no white balance offset is required.

The initial value for this attribute is 0.0 for each color component.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::IsAcqLut

BOOL **IsAcqLut**();

#### Remarks

Checks if the color lookup table corresponds to the acquisition LUT. If the return value is FALSE, then a software lookup table is used instead.

The initial value for this attribute is FALSE. It is then set according to the current acquisition device lookup table availability when calling the Create method.

#### Demo/Example Usage

Not available

---

### SapColorConversion::IsEnabled

BOOL **IsEnabled**();

#### Remarks

Checks if color conversion is enabled. The initial value for this attribute depends on the acquisition device.

Use the Enable method if you need to enable or disable color conversion.

#### Demo/Example Usage

---

### SapColorConversion::IsHardwareEnabled

BOOL **IsHardwareEnabled()**;

#### Remarks

Returns TRUE if hardware conversion is enabled.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::IsHardwareSupported

BOOL **IsHardwareSupported()**;

#### Remarks

Returns TRUE if the input buffer is compatible with hardware conversion. Supported input buffer formats for color conversion (Bayer and Bicolor) are SapFormatTypeMono.

#### Demo/Example Usage

Not available

---

### SapColorConversion::IsLutEnabled

BOOL **IsLutEnabled()**;

#### Remarks

Gets the current color lookup table enable value. When enabled, this LUT is applied to image data after color conversion has been performed.

The initial value for this attribute is FALSE. Use the EnableLut method to enable or disable the lookup table.

#### Demo/Example Usage

Color Conversion Demo

---

### SapColorConversion::IsSoftwareEnabled

BOOL **IsSoftwareEnabled()**;

#### Remarks

Returns TRUE if software conversion is enabled.

#### Demo/Example Usage

Not available

---

### SapColorConversion::IsSoftwareSupported

BOOL **IsSoftwareSupported()**;

#### Remarks

Returns TRUE if the input buffer is compatible with software conversion. Supported input buffer formats for color conversion (Bayer and Bicolor) are SapFormatTypeMono.

#### Demo/Example Usage

Not available

---

### SapColorConversion::WhiteBalance

BOOL **WhiteBalance(int x, int y, int width, int height);**

BOOL **WhiteBalance(SapBuffer\* pBuffer, int x, int y, int width, int height);**

#### Parameters

*x* Left coordinate of white balance region of interest

*y* Top coordinate of white balance region of interest

*Width* Width of white balance region of interest

*Height* Height of white balance region of interest

*pBuffer* Buffer object with the white balance region of interest

## **Return Value**

Returns TRUE if successful, FALSE otherwise

## **Remarks**

Calculates the white balance gain coefficients needed for color conversion. The region of interest of a color-encoded image containing a uniformly illuminated white region. The intensity of the pixels should be as high as possible but not saturated. The coefficients are calculated as follows:

$$G_R = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{R}$$

$$G_G = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{G}$$

$$G_B = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{B}$$

where  $\overline{R}$ ,  $\overline{G}$  and  $\overline{B}$  are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer resource at the current index in the main buffer object (the one in which images are acquired or loaded) is used, unless you explicitly specify another buffer object using the *pBuffer* argument.

## **Demo/Example Usage**

Color Conversion Demo, GigE Auto-White Balance Example

---

## **SapColorConversion::WhiteBalanceManual**

**BOOL WhiteBalanceManual(const SapDataFRGB& *wbGain*);**

## **Parameters**

*wbGain* Left coordinate of white balance region of interest

## **Return Value**

Returns TRUE if successful, FALSE otherwise

## **Remarks**

Calculates the white balance gain coefficients needed for color conversion. The region of interest of a color-encoded image containing a uniformly illuminated white region. The intensity of the pixels should be as high as possible but not saturated. The coefficients are calculated as follows:

$$G_R = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{R}$$

$$G_G = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{G}$$

$$G_B = \text{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{B}$$

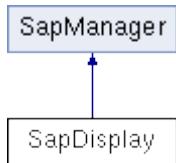
where  $\overline{R}$ ,  $\overline{G}$  and  $\overline{B}$  are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer resource at the current index in the main buffer object (the one in which images are acquired or loaded) is used, unless you explicitly specify another buffer object using the *pBuffer* argument.

## **Demo/Example Usage**

Color Conversion Demo, GigE Auto-White Balance Example

# SapDisplay



The SapDisplay Class includes functionality to manipulate a display resource. There is at least one such resource for each display adapter (VGA board) in the system.

Note that SapView objects automatically manage an internal SapDisplay object for the default display resource. However, you must explicitly manage the object yourself if you need a display resource other than the default one.

```
#include <SapClassBasic.h>
```

## SapDisplay Class Members

### Construction

<a href="#"><u>SapDisplay</u></a>	Class constructor
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources
<b>Attributes</b>	
<a href="#"><u>GetLocation</u></a>	Gets/sets the location where the display resource is located
<a href="#"><u>SetLocation</u></a>	
<a href="#"><u>GetWidth</u></a>	Gets the width (in pixels) for the current display mode
<a href="#"><u>GetHeight</u></a>	Gets the height (in lines) for the current display mode
<a href="#"><u>GetPixelDepth</u></a>	Gets the number of significant bits per pixel for the current display mode
<a href="#"><u>GetRefreshRate</u></a>	Gets the refresh rate for the current display mode
<a href="#"><u>IsInterlaced</u></a>	Checks if the current display mode is interlaced or progressive
<a href="#"><u>GetType</u></a>	Gets the type of the display (primary or secondary)
<a href="#"><u>GetFormatDetection</u></a>	Gets/sets automatic detection of available offscreen and overlay buffer formats
<a href="#"><u>SetFormatDetection</u></a>	
<a href="#"><u>IsPrimaryVGABoard</u></a>	Checks if the current display belongs to the primary VGA board in the system
<a href="#"><u>IsOffscreenAvailable</u></a>	Checks if offscreen display support of a specific buffer format is available
<a href="#"><u>IsOverlayAvailable</u></a>	Checks if overlay display support of a specific buffer format is available
<a href="#"><u>GetHandle</u></a>	Gets the low-level Sapera handle of the display resource
<b>Operations</b>	
<a href="#"><u>GetDC</u></a>	Gets the Windows Device Context corresponding to the entire screen
<a href="#"><u>ReleaseDC</u></a>	Releases the Windows Device Context corresponding to the entire screen
<a href="#"><u>IsCapabilityValid</u></a>	Checks for the availability of a low-level Sapera C library capability
<a href="#"><u>IsParameterValid</u></a>	Checks for the availability of a low-level Sapera C library parameter
<a href="#"><u>GetCapability</u></a>	Gets the value of a low-level Sapera C library capability
<a href="#"><u>GetParameter</u></a>	Gets/sets the value of a low-level Sapera C library parameter
<a href="#"><u>SetParameter</u></a>	

## SapDisplay Member Functions

The following are members of the SapDisplay Class.

### SapDisplay::SapDisplay

## **SapDisplay()**

### **Remarks**

The SapDisplay constructor does not actually create the low-level Saperia resources. To do this, you must call the Create method. The SapDisplay object's display resource is always the host server system.

Note that SapView objects automatically manages an internal SapDisplay object for the default display resource; however, you must explicitly manage the object if you need a display resource other than the default one.

### **Demo/Example Usage**

Not available

---

## **SapDisplay::Create**

**BOOL Create();**

### **Return Value**

Returns TRUE if successful, FALSE otherwise

### **Remarks**

Creates all the low-level Saperia resources needed by the display object.

If you allow a SapView object to automatically manage a SapDisplay object, then you do not need to call this method; otherwise, you must always call it before the SapView::Create method.

### **Demo/Example Usage**

Not available

---

## **SapDisplay::Destroy**

**BOOL Destroy();**

### **Return Value**

Returns TRUE if successful, FALSE otherwise

### **Remarks**

Destroys all the low-level Saperia resources needed by the display object.

If you allow a SapView object to automatically manage a SapDisplay object, then you do not need to call this method; otherwise, you must always call it after the SapView::Destroy method.

### **Demo/Example Usage**

Not available

---

## **SapDisplay::GetCapability**

**BOOL GetCapability(int cap, void\* pValue);**

### **Parameters**

*cap* Low-level Saperia C library capability to read

*Pvalue* Pointer to capability value to read back

### **Return Value**

Returns TRUE if successful, FALSE otherwise

### **Remarks**

This method allows direct read access to low-level Saperia C library capabilities for the display module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapDisplay Class already uses important capabilities internally for self-configuration and validation.

See the *Saperia LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

### **Demo/Example Usage**

Not available

---

## **SapDisplay::GetDC**

```
BOOL GetDC(HDC* pDC);
```

#### Parameters

*pDC* Pointer to display context value

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the Windows Device Context corresponding to the entire screen for the current display object.

#### Demo/Example Usage

Not available

---

### SapDisplay::GetFormatDetection, SapDisplay::SetFormatDetection

```
BOOL GetFormatDetection(); BOOL SetFormatDetection(BOOL formatDetection);
```

#### Remarks

Gets/sets automatic detection of available offscreen and overlay buffer formats. If the value of this attribute is TRUE, then all offscreen and overlay formats available for creating buffers are automatically detected when calling the Create method. It is then possible to call the IsOffscreenAvailable and IsOverlayAvailable methods to quickly find out if creating such buffers should succeed. The drawback to this detection is that creating a SapDisplay object takes much longer, and can produce a noticeable flicker effect whenever a SapDisplay object is created explicitly by the application, or implicitly through a SapView object. While turning off auto detection solves these issues, the IsOffscreenAvailable and IsOverlayAvailable methods then become useless, and always return TRUE. In this case, trying to create a buffer of an invalid format generates an error without any possibility of prior checking. You can only call SetFormatDetection before the Create method. The initial value for this attribute is TRUE.

#### Demo/Example Usage

Not available

---

### SapDisplay::GetHandle

```
CORHANDLE GetHandle();
```

#### Remarks

Gets the low-level Sapera handle of the display resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method. See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

#### Demo/Example Usage

Not available

---

### SapDisplay::GetHeight

```
int GetHeight();
```

#### Remarks

Gets the height (in lines) for the current display mode. This attribute has a value of value of 0 until the Create method is called.

#### Demo/Example Usage

Not available

---

### SapDisplay::GetLocation, SapDisplay::SetLocation

```
SapLocation GetLocation();
```

```
BOOL SetLocation(SapLocation location);
```

#### Remarks

Gets/sets the location where the display resource is located. This usually corresponds to the system server. A specific server can also be specified through the SapDisplay constructor.

You can only call SetLocation before the Create method.

#### Demo/Example Usage

Not available

---

## SapDisplay::GetParameter, SapDisplay::SetParameter

```
BOOL GetParameter(int param, void* pValue);
BOOL SetParameter(int param, int value);
BOOL SetParameter(int param, void* pValue);
```

### Parameters

*param* Low-level Sapera C library parameter to read or write  
*pValue* Pointer to parameter value to read back or to write  
*value* New parameter value to write

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the display module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapDisplay Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapDisplay::GetPixelDepth

```
int GetPixelDepth();
```

### Remarks

Gets the number of significant bits per pixel for the current display mode.

The initial value for this attribute is 0. It is then set according to the current display value when calling the Create method.

### Demo/Example Usage

Not available

---

## SapDisplay::GetRefreshRate

```
int GetRefreshRate();
```

### Remarks

Gets the refresh rate (in Hz) for the current display mode

The initial value for this attribute is 0. It is then set according to the current display value when calling the Create method.

### Demo/Example Usage

Not available

---

## SapDisplay::GetType

```
SapDisplay::Type GetType();
```

### Return Value

Display type, which can be one of the following values:

SapDisplay::TypeUnknown	Undetermined display type
SapDisplay::TypeSystem	A display under the control of the primary Windows display driver. It normally displays the Windows Desktop.
SapDisplay::TypeDuplicate	A secondary display that shows the same contents as the primary Windows VGA display
SapDisplay::TypeExtended	A secondary display that extends the desktop from the primary

SapDisplay::TypeIndependent	Windows VGA display A secondary display that is completely independent from the primary Windows VGA display
-----------------------------	--

### Remarks

Gets the type of the display (primary or secondary) .

The initial value for this attribute is TypeUnknown. It is then set according to the current display value when calling the Create method.

### Demo/Example Usage

Not available

## SapDisplay::GetWidth

**int GetWidth();**

### Remarks

Gets the width (in pixels) for the current display mode.

The initial value for this attribute is 0. It is then set according to the current display value when calling the Create method.

### Demo/Example Usage

Not available

## SapDisplay::IsCapabilityValid

**BOOL IsCapabilityValid(int cap);**

### Parameters

*cap* Low-level Sapera C library capability to check

### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

### Remarks

Checks for the availability of a low-level Sapera C library capability for the display module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapDisplay class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

### Demo/Example Usage

Not available

## SapDisplay::IsInterlaced

**BOOL IsInterlaced();**

### Remarks

Checks if the current display mode is interlaced or progressive (non-interlaced).

The initial value for this property is FALSE. It is then set according to the current display value when calling the Create method.

### Demo/Example Usage

Not available

## SapDisplay::IsOffscreenAvailable

**BOOL IsOffscreenAvailable(SapFormat format);**

### Remarks

Checks if offscreen display support is available for a given buffer format. See the SapBuffer constructor for a list of possible values for *format*.

You can only call IsOffscreenAvailable after the Create method.

### Demo/Example Usage

Not available

---

### SapDisplay::IsOverlayAvailable

BOOL **IsOverlayAvailable**(SapFormat *format*);

#### Remarks

Checks if overlay display support is available for a given buffer format. See the SapBuffer constructor for a list of possible values for *format*.

You can only call IsOverlayAvailable after the Create method.

#### Demo/Example Usage

Not available

---

### SapDisplay::IsPrimaryVGABoard

BOOL **IsPrimaryVGABoard**();

#### Remarks

Checks if the current display belongs to the primary VGA board in the system.

You can only call IsPrimaryVGABoard after the Create method.

#### Demo/Example Usage

Not available

---

### SapDisplay::IsParameterValid

BOOL **IsParameterValid**(int *param*);

#### Parameters

*param* Low-level Sapera C library parameter to check

#### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library parameter for the display module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapDisplay class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Not available

---

### SapDisplay::ReleaseDC

BOOL **ReleaseDC**();

#### Return Value

Returns TRUE if successful, FALSE otherwise

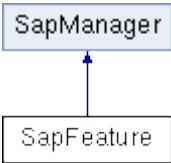
#### Remarks

Releases the Windows Device Context corresponding to the entire screen for the current display object.

#### Demo/Example Usage

Not available

# SapFeature



The purpose of the SapFeature class is to retrieve individual feature information from the SapAcqDevice class. Each feature supported by SapAcqDevice provides a set of capabilities such as name, type, access mode, and so forth, which can be obtained through SapFeature. The GetFeatureInfo method of SapAcqDevice gives access to this information.

```
#include <SapClassBasic.h>
```

## SapFeature Class Members

### Construction

[SapFeature](#)

Class constructor

[Create](#)

Allocates the low-level Sapera resources

[Destroy](#)

Releases the low-level Sapera resources

### Attributes

[GetLocation](#)

Gets/sets the location where the feature resource is located

[SetLocation](#)

Gets the low-level Sapera handle of the feature resource

### General Parameters

[GetName](#)

Gets the short name of the feature

[GetType](#)

Gets the data type of the feature

[IsStandard](#)

Checks if a feature is standard or custom

[GetAccessMode](#)

Gets the current data access mode for a feature

[GetPollingTime](#)

Gets the interval of time between two consecutive feature updates

[GetToolTip](#)

Gets the text which represents the explanation of the feature

[GetDescription](#)

Gets the text which represents the full description of the feature

[GetDisplayName](#)

Gets the descriptive name of the feature

[GetFloatNotation](#)

Gets the notation type to use to display a float type feature

[GetFloatPrecision](#)

Gets the number of decimal places to display for a float type feature

[GetRepresentation](#)

Gets the mathematical representation of a integer or float feature

[GetSign](#)

Checks if an integer/float feature is signed or not

[GetSiUnit](#)

Gets the physical units representing the feature in the international system (SI)

[GetCategory](#)

Gets the category to which the current feature belongs

[GetWriteMode](#)

Checks if a feature can be modified when the transfer object is connected and/or acquiring

[IsSavedToConfigFile](#)

Checks if a feature is saved to a CCF configuration file

[SetSavedToConfigFile](#)

Gets the feature conversion factor from international system (SI) units to native units

[GetSiToNativeExp10](#)

Gets the level of visibility assigned to a feature

[GetArrayLength](#)

Gets the number of bytes required for an array type feature

[GetIncrementType](#)

Gets the type of increment for an integer or floating-point feature

[GetValidValueCount](#)

Get the number of valid values for an integer or floating-point feature which defines them as a list

## **Integer/float-Parameters**

<u>GetMin</u>	Gets the minimum acceptable value for a feature
<u>GetMax</u>	Gets the maximum acceptable value for a feature
<u>GetInc</u>	Gets the minimum acceptable increment for an integer or a float feature
<u>GetValidValue</u>	Gets one of a predefined set of valid values for a feature

## **Enumeration-Parameters**

<u>GetEnumCount</u>	Get the number of possible values for a feature which belongs to an enumerated type
<u>GetEnumString</u>	Gets the string value at a specified index for the enumerated type corresponding to the current feature
<u>GetEnumValue</u>	Gets the integer value at a specified index for the enumerated type corresponding to the current feature
<u>IsEnumEnabled</u>	Checks if the enumeration value corresponding to a specified index is enabled
<u>GetEnumStringFromValue</u>	Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature
<u>GetEnumValueFromString</u>	Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature

## **Selector-Parameters**

<u>IsSelector</u>	Determines if the value of a feature directly affects other features
<u>GetSelectedFeatureCount</u>	Gets the number of features associated with a selector
<u>GetSelectedFeatureIndex</u>	Gets the index of a feature associated with a selector
<u>GetSelectedFeatureName</u>	Gets the name of a feature associated with a selector
<u>GetSelectingFeatureCount</u>	Gets the number of selectors associated with a feature
<u>GetSelectingFeatureIndex</u>	Gets the index of a selector associated with a feature
<u>GetSelectingFeatureName</u>	Gets the name of a selector associated with a feature

## **SapFeature Member Functions**

The following are members of the SapFeature Class.

---

### **SapFeature::SapFeature**

**SapFeature**(SapLocation *location* = SapLocation::ServerSystem);

#### **Parameters**

*location* SapLocation object specifying where the feature is located. This location must be the same as that of the SapAcqDevice object from which the feature is retrieved.

#### **Remarks**

The SapFeature constructor does not actually create the low-level Sapera resources. To do this, you must call the SapFeature::Create method. Upon creation the feature object contents are meaningless. To fill-in a feature object, call the SapAcqDevice::GetFeatureInfo method.

#### **Demo/Example Usage**

Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, Grab CameraLink Example

---

### **SapFeature::Create**

BOOL **Create()**;

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

#### **Remarks**

Creates all the low-level Sapera resources needed by the feature object. Call this method before using the object as a parameter to the SapAcqDevice::GetFeatureInfo method.

## Demo/Example Usage

Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, Grab CameraLink Example

---

## SapFeature::Destroy

**BOOL Destroy();**

### Return Value

Returns TRUE if successful, FALSE otherwise.

### Remarks

Destroys all the low-level Sapera resources needed by the feature object.

## Demo/Example Usage

Camera Events Example, Camera Features Example, Camera Files Example, GigE Auto-White Balance Example, Grab CameraLink Example

---

## SapFeature::GetAccessMode

**BOOL GetAccessMode(SapFeature::AccessMode\* accessMode);**

### Parameters

<i>accessMode</i>	Returned data access mode can be one of the following values:
SapFeature::AccessUndefined	Undefined access mode
SapFeature::AccessRW	The feature may be read and written. Most features are of this type.
SapFeature::AccessRO	The feature can only be read.
SapFeature::AccessWO	The feature can only be written. This is the case for some features which represent commands (or actions) such as 'TimestampReset'.
SapFeature::AccessNP	The feature is not present. The feature is visible in the interface but is not implemented for this device.
SapFeature::AccessNE	The feature is present but currently not enabled. Often used when a feature depends on another feature's value.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the current data access mode for a feature.

## Demo/Example Usage

Camera Features Example, Camera Files Example

---

## SapFeature::GetArrayLength

**BOOL GetArrayLength(int\* arrayLength);**

### Parameters

<i>arrayLength</i>	Returned array length (in bytes).
--------------------	-----------------------------------

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the number of bytes required to store the value of a feature of array type, that is, when the value returned by the GetType method is SapFeature::TypeArray. You can then create a SapBuffer object with a height of one line, and a width corresponding to this number of bytes, and then use this buffer when calling the GetFeatureValue and SetFeatureValue methods in the SapAcqDevice class.

## Demo/Example Usage

Not available

---

## SapFeature::GetCategory

BOOL **GetCategory**(char\* *category*, int *categorySize*);

### Parameters

*category* Buffer for the returned text string. Must be large enough for 64 characters.

*categorySize* Size of the buffer pointed to by *category* (in bytes)

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the category to which the current feature belongs. To simplify the classification of a large set of features from the same SapAcqDevice object, the features are divided into categories. These categories are useful for presenting a list of features in a graphical user interface.

### Demo/Example Usage

Not available

---

## SapFeature::GetDescription

BOOL **GetDescription**(char\* *description*, int *descriptionSize*);

### Parameters

*description* Buffer for the returned text string. Must be large enough for 512 characters.

*descriptionSize* Size of the buffer pointed to by *description* (in bytes)

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the text which represents the full description of the feature. This information can be used to display detailed textual information in a graphical user interface.

### Demo/Example Usage

Not available

---

## SapFeature::GetDisplayName

BOOL **GetDisplayName**(char\* *displayName*, int *displayNameSize*);

### Parameters

*displayName* Buffer for the returned text string. Must be large enough for 64 characters.

*displayNameSize* Size of the buffer pointed to by *displayName* (in bytes)

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the descriptive name of the feature. This name can be used for listing features in a graphical user interface.

### Demo/Example Usage

Camera Features Example

---

## SapFeature::GetEnumCount

BOOL **GetEnumCount**(int\* *enumCount*);

### Parameters

*enumCount* Returned number of enumeration items

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the number of possible values for a feature which belongs to an enumerated type. Use this method along with the GetEnumString and GetEnumValue methods to enumerate all the items contained within an enumeration feature.

## Demo/Example Usage

Camera Features Example

---

### SapFeature::GetEnumString

BOOL **GetEnumString**(int *enumIndex*, char\* *enumString*, int *enumStringSize*);

#### Parameters

<i>enumIndex</i>	Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)
<i>enumString</i>	Buffer for the returned text string.
<i>enumStringSize</i>	Size of the buffer pointed to by <i>enumString</i> (in bytes)

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the string value at a specified index for the enumerated type corresponding to the current feature. Use this method along with the GetEnumCount and GetEnumValue methods to enumerate all the items contained within an enumeration feature.

## Demo/Example Usage

Camera Features Example

---

### SapFeature::GetEnumStringFromValue

BOOL **GetEnumStringFromValue**(int *enumValue*, char\* *enumString*, int *enumStringSize*);

#### Parameters

<i>enumValue</i>	Value to look for in the enumeration items
<i>enumString</i>	Buffer for the returned text string
<i>enumStringSize</i>	Size of the buffer pointed to by <i>enumString</i> (in bytes)

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature. For example you may use this method to retrieve the string corresponding to an enumeration value returned by the SapAcqDevice::GetFeatureValue method.

## Demo/Example Usage

Camera Features Example

---

### SapFeature::GetEnumValue

BOOL **GetEnumValue**(int *enumIndex*, int\* *enumValue*);

#### Parameters

<i>enumIndex</i>	Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)
<i>enumValue</i>	Returns enumeration value

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the integer value at a specified index for the enumerated type corresponding to the current feature. Use this method along with the GetEnumCount and GetEnumString methods to enumerate all the items contained within an enumeration feature.

## Demo/Example Usage

Not available

---

### SapFeature::GetEnumValueFromString

BOOL **GetEnumValueFromString**(const char\* *enumString*, int\* *enumValue*);

## Parameters

- enumString* Text string to look for in the enumeration  
*enumValue* Returned integer value

## Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature. For example you may use this method to retrieve the value corresponding to a known enumeration string before calling the SapAcqDevice::SetFeatureValue method.

## Demo/Example Usage

Not available

---

## SapFeature::GetFloatNotation

BOOL **GetFloatNotation**(FloatNotation\* *notation*);

## Parameters

- notation* Specifies how the float type feature is displayed. Possible values are:
- |                                     |   |
|-------------------------------------|---|
| SapFeature::FloatNotationFixed      | Display variable using fixed notation. For example,<br>123.4          |
| SapFeature::FloatNotationScientific | Display variable using scientific notation. For<br>example, 1.234e-2. |
| SapFeature::FloatNotationUndefined  | Undefined.  |

## Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the type of notation to use to display a float type feature.

## Demo/Example Usage

Not available

---

## SapFeature::GetFloatPrecision

BOOL **GetFloatPrecision**(int64 \* *precision*);

## Parameters

- precision* Number of decimal places of a float type feature to display.

## Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the number of decimal places to display for a float type feature.

## Demo/Example Usage

Not available

---

## SapFeature::GetHandle

CORHANDLE **GetHandle**();

## Remarks

Gets the low-level Sapera handle of the feature resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method. See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

## Demo/Example Usage

Camera Features Example

---

## SapFeature::GetInc

BOOL **GetInc**(INT32\* *incValue*);  
BOOL **GetInc**(UINT32\* *incValue*);

```
BOOL GetInc(INT64* incValue);
BOOL GetInc(UINT64* incValue);
BOOL GetInc(float* incValue);
BOOL GetInc(double* incValue);
```

#### Parameters

*incValue*      Returned increment value

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the minimum acceptable increment for an integer or a float feature. Some features cannot vary by increments of 1. Their value must be a multiple of a certain increment. For example the buffer cropping dimensions might require to be a multiple of 4 in order to optimize the data transfer.

#### Demo/Example Usage

Not available

---

### SapFeature::GetIncrementType

```
BOOL GetIncrementType(SapFeature::IncrementType* incrementType);
```

#### Parameters

<i>incrementType</i>	Returned increment type can be one of the following values:
SapFeature::IncrementUndefined	Undefined increment type. This normally means that the acquisition device to which the feature is associated does not support reading the value of the increment type.
SapFeature::IncrementNone	The feature has no increment. Use the GetMin and GetMax functions to find out the feature value limits.
SapFeature::IncrementLinear	The feature has a fixed increment. Use the GetMin and GetMax functions to find the feature value limits, and GetInc to find the increment.
SapFeature::IncrementList	The feature has a fixed set of valid values. Use the GetValidValueCount function to find the number of values, and the GetValidValue function to enumerate them.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the type of increment for an integer or floating-point feature. This is useful for finding out which values are valid for this feature.

#### Demo/Example Usage

Camera Features Example

---

### SapFeature::GetLocation, SapFeature::SetLocation

```
SapLocation GetLocation();
BOOL SetLocation(SapLocation location);
```

#### Remarks

Gets/sets the location where the feature resource is located. This location must be the same as that of the corresponding SapAcqDevice object. A specific location can also be specified through the SapFeature constructor.

You can only call SetLocation before the Create method.

#### Demo/Example Usage

Not available

---

### SapFeature::GetMax

```
BOOL GetMax(INT32* maxValue);
BOOL GetMax(UINT32* maxValue);
BOOL GetMax(INT64* maxValue);
BOOL GetMax(UINT64* maxValue);
```

```
BOOL GetMax(float* maxValue);  
BOOL GetMax(double* maxValue);
```

#### Parameters

*maxValue*      Returned maximum value

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the maximum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the UINT32 version to get the maximum length of the string (excluding the trailing null character).

#### Demo/Example Usage

Camera Events Example, Camera Features Example, GigE Auto-White Balance Example, Grab CameraLink Example

---

### SapFeature::GetMin

```
BOOL GetMin(INT32* minValue);  
BOOL GetMin(UINT32* minValue);  
BOOL GetMin(INT64* minValue);  
BOOL GetMin(UINT64* minValue);  
BOOL GetMin(float* minValue);  
BOOL GetMin(double* minValue);
```

#### Parameters

*minValue*      Returned minimum value

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the minimum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the UINT32 version to get the minimum length of the string (excluding the trailing null character).

#### Demo/Example Usage

Camera Events Example, Camera Features Example, GigE Auto-White Balance Example, Grab CameraLink Example

---

### SapFeature::GetName

```
BOOL GetName(char* name, int nameSize);
```

#### Parameters

*name*      Buffer for the returned text string. Must be large enough for 64 characters.

*nameSize*      Size of the buffer pointed to by *name* (in bytes)

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the short name of the feature. This name can be used with SapAcqDevice method which expect a feature name. This string should not be used for display in a graphical user interface. Use the GetDisplayName method instead to provide a more descriptive name.

#### Demo/Example Usage

Not available

---

### SapFeature::GetPollingTime

```
BOOL GetPollingTime(int* pollingTime);
```

#### Parameters

*pollingTime*      Returned polling time (in milliseconds).

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the interval of time between two consecutive feature updates. Some read-only features (such as 'InternalTemperature') are read internally from the acquisition device at a certain frequency in order to always stay up to date.

Note that this method is only relevant for acquisition devices which are supported through the Genie Framework. Other devices do not return a polling time, but instead use internal polling that generates "Feature Info Changed" events whenever required.

#### Demo/Example Usage

Not available

---

### SapFeature::GetRepresentation

**BOOL GetRepresentation(SapFeature::Representation\* representation);**

#### Parameters

<i>representation</i>	Returned representation can be one of the following values:
SapFeature::RepresentationUndefined	Undefined representation
SapFeature::RepresentationLinear	The feature follows a linear scale
SapFeature::RepresentationLogarithmic	The feature follows a logarithmic scale
SapFeature::RepresentationBoolean	The feature can have two values: zero or non-zero

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the mathematical representation of a integer or float feature.

#### Demo/Example Usage

Not available

---

### SapFeature::GetSelectedFeatureCount

**BOOL GetSelectedFeatureCount(int \* selectedCount);**

#### Parameters

<i>selectedCount</i>	Returned number of features associated with the selector, 0 if the current feature is not a selector.
----------------------	---

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Returns the number of features associated with a selector (value returned by IsSelector method is TRUE). These selected features can be considered as children of the current SapFeature object.

#### Demo/Example Usage

Not available

---

### SapFeature::GetSelectedFeatureIndex

**BOOL GetSelectedFeatureIndex(int selectedIndex, int \* featureIndex);**

#### Parameters

<i>selectedIndex</i>	Index of the selected feature, relative to the selector, from 0 to the value returned by the GetSelectedFeatureCount method, minus 1.
<i>featureIndex</i>	Returned index of the selected feature, relative to the acquisition device, from 0 to the value returned by the SapAcqDevice::GetFeatureCount method, minus 1.

#### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Returns the acquisition device index of a feature associated with a selector (value returned by the IsSelector method is TRUE). This feature can be considered as a child of the current SapFeature object.

The number of features associated with the selector is returned by the GetSelectedFeatureCount method.

The returned index can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the SapAcqDevice::GetFeatureCount method.

## Demo/Example Usage

Not available

---

### SapFeature::GetSelectedFeatureName

**BOOL GetSelectedFeatureName(int selectedIndex, char \* featureName, int featureNameSize);**

#### Parameters

<i>selectedIndex</i>	Index of the selected feature, relative to the selector, from 0 to the value returned by the GetSelectedFeatureCount method, minus 1.
<i>featureName</i>	Acquisition device feature name.
<i>featureNameSize</i>	Size (in bytes) of the buffer pointed to by <i>featureName</i>

#### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Returns the acquisition device name of a feature associated with a selector (value returned by the IsSelector method is TRUE). This feature can be considered as a child of the current SapFeature object.

The number of features associated with the selector is returned by the GetSelectedFeatureCount method.

The returned name can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the SapAcqDevice::GetFeatureCount method.

## Demo/Example Usage

Not available

---

### SapFeature::GetSelectingFeatureCount

**BOOL GetSelectingFeatureCount(int \* selectingCount);**

#### Parameters

<i>selectingCount</i>	Returned number of selectors associated with the current feature, 0 if there are no associated selectors.
-----------------------	---

#### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Returns the number of selectors (value returned by IsSelector method is TRUE) associated with a feature. These selectors can be considered as parents of the current SapFeature object.

## Demo/Example Usage

Not available

---

### SapFeature::GetSelectingFeatureIndex

**BOOL GetSelectingFeatureIndex(int selectingIndex, int \* featureIndex);**

#### Parameters

<i>selectingIndex</i>	Index of the selector, relative to the current feature, from 0 to the value returned by the GetSelectingFeatureCount method, minus 1.
<i>featureIndex</i>	Returned index of the selector, relative to the acquisition device, from 0 to the value returned by the SapAcqDevice::GetFeatureCount method, minus 1.

#### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Returns the acquisition device index of a selector (value returned by the IsSelector method is TRUE) associated with a feature. This selector can be considered as a parent of the current SapFeature object.

The number of selectors associated with a feature is returned by the GetSelectingFeatureCount method.

The returned index can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the SapAcqDevice::GetFeatureCount method.

## Demo/Example Usage

Not available

---

## SapFeature::GetSelectingFeatureName

BOOL **GetSelectingFeatureName**(int *selectingIndex*, char \* *featureName*, int *featureNameSize*);

### Parameters

<i>selectingIndex</i>	Index of the selector, relative to the current feature, from 0 to the value returned by the GetSelectingFeatureCount method, minus 1.
<i>featureName</i>	Acquisition device feature name.
<i>featureNameSize</i>	Size (in bytes) of the buffer pointed to by <i>featureName</i>

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Returns the acquisition device name of a selector (value returned by the IsSelector method is TRUE) associated with a feature. This selector can be considered as a parent of the current SapFeature object.

The number of selectors associated with a feature is returned by the GetSelectingFeatureCount method.

The returned name can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the SapAcqDevice::GetFeatureCount method.

## Demo/Example Usage

Not available

---

## SapFeature::GetSign

BOOL **GetSign**(SapFeature::Sign\* *sign*);

### Parameters

<i>sign</i>	Returned sign can be one of the following values:
SapFeature::SignUndefined	Sign is undefined
SapFeature::Signed	The feature is a signed integer or float
SapFeature::Unsigned	The feature is an unsigned integer or float

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the sign of an integer or float feature. This information is useful when reading and writing feature values. By knowing the sign of the feature value you can cast it to the corresponding C/C++ type.

## Demo/Example Usage

Not available

---

## SapFeature::GetSiToNativeExp10

BOOL **GetSiToNativeExp10**(int\* *exponent*);

### Parameters

<i>exponent</i>	Returned exponent value (base 10). It can be negative or positive.
-----------------	--

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the exponent for converting the value of a feature from international system (SI) units to native units (the units used to read/write the feature through the API).

The following equation describes the relation between the two unit systems:

$$V_{\text{NATIVE}} = V_{\text{SI}} * 10^E$$

Where  $V$  is the value of a feature and  $E$  is the current parameter.

## Example 1

You want to set the camera exposure time to a known value in seconds. The 'ExposureTime' feature is represented in microseconds. Therefore the current exponent value is 6. If the desired integration time is 0.5 second, then you can compute the actual value for the SapAcqDevice::SetFeatureValue method as follows:

$$V_{\text{NATIVE}} = 0.5 * 10^6 = 500000$$

## Example 2

You want to monitor the temperature of the camera sensor. The 'InternalTemperature' feature is reported in degrees Celcius. Therefore the current exponent value is 0. If the feature value returned by the SapAcqDevice::GetFeatureValue method is 50 then the temperature in Celcius is also equal to 50.

Use the GetSiUnit method to retrieve the international system (SI) units corresponding to the feature to monitor.

## Demo/Example Usage

Camera Events Example, GigE Auto-White Balance Example

---

## SapFeature::GetSiUnit

BOOL **GetSiUnit**(char\* *unit*, int *unitSize*);

### Parameters

*unit* Buffer for the returned text string. Must be large enough for 32 characters.

*unitSize* Size of the buffer pointed to by *unit* (in bytes)

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the physical units representing the feature in the international system (SI). Examples of units are Volts, Pixels, Celsius, Degrees, etc. This information is useful to present in a graphical user interface.

Most of the time the units used by the feature (the native units) are NOT the same as SI units, but rather a multiple of them. For example, the exposure time may be represented in microseconds instead of seconds. To convert the feature value to the SI units you must use the exponent value provided by the GetSiToNativeExp10 method.

## Demo/Example Usage

Not available

---

## SapFeature::GetToolTip

BOOL **GetToolTip**(char\* *tooltip*, int *tooltipSize*);

### Parameters

*tooltip* Buffer for the returned text string. Must be large enough for 256 characters.

*tooltipSize* Size of the buffer pointed to by *tooltip* (in bytes)

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the text which represents the explanation of the feature. This information can be used to implement tool tips in a graphical user interface.

## Demo/Example Usage

Not available

---

## SapFeature::GetType

**BOOL GetType(SapFeature::Type \*type);**

**Parameters**

type	Returned type can be one of the following values:
SapFeature::TypeUndefined	Undefined type
SapFeature::TypeInt32	32-bit integer
SapFeature::TypeInt64	64-bit integer
SapFeature::TypeFloat	32-bit floating-point
SapFeature::TypeDouble	64-bit floating-point
SapFeature::TypeBool	Boolean
SapFeature::TypeEnum	Enumeration
SapFeature::TypeString	ASCII character string
SapFeature::TypeBuffer	Saper LT buffer object (SapBuffer)
SapFeature::TypeLut	Saper LT look-up table object (SapLut)
SapFeature::TypeArray	Saper LT buffer object (SapBuffer)

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Gets the data type of the feature.

If the feature is of array type, then the SapBuffer object should have a height of one line, and a width corresponding to the number of bytes given by the value returned by the GetArrayLength method.

**Demo/Example Usage**

Camera Features Example

---

### SapFeature::GetValidValue

```
BOOL GetValidValue(int validValueIndex, INT32* validValue);
BOOL GetValidValue(int validValueIndex, UINT32* validValue);
BOOL GetValidValue(int validValueIndex, INT64* validValue);
BOOL GetValidValue(int validValueIndex, UINT64* validValue);
BOOL GetValidValue(int validValueIndex, float* validValue);
BOOL GetValidValue(int validValueIndex, double* validValue);
```

**Parameters**

validValueIndex	Index of the valid value, can be any value from 0 to the value returned by the GetValidValueCount function, minus 1
validValue	Returned valid value, must point to a variable of the same type as the feature

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Gets one of a predefined set of valid values for an integer or floating-point feature which defines them as a list, that is, the GetIncrementType function returns SapFeature::IncrementList.

**Demo/Example Usage**

Camera Features Example

---

### SapFeature::GetValidValueCount

```
BOOL GetValidValueCount(int* validValueCount);
```

**Parameters**

validValueCount	Returned count of valid values.
-----------------	---------------------------------

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Get the number of valid values for an integer or floating-point feature which defines them as a list, that is, the GetIncrementType function returns SapFeature::IncrementList. In this case, use the GetValidValue function to

enumerate these values.

### Demo/Example Usage

Camera Features Example

---

## SapFeature::GetVisibility

**BOOL GetVisibility(SapFeature::Visibility\* visibility);**

### Parameters

<i>visibility</i>	Returned visibility can be one of the following values:
SapFeature::VisibilityUndefined	Undefined visibility level
SapFeature::VisibilityBeginner	The feature should be made visible to any user
SapFeature::VisibilityExpert	The feature should be made visible to users with a certain level of expertise
SapFeature::VisibilityGuru	Specifies that the feature should be made visible to users with a high level of expertise
SapFeature::VisibilityInvisible	The feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets the level of visibility assigned to a feature. This information is useful to classify the features in a graphical user interface in terms of user expertise.

### Demo/Example Usage

Not available

---

## SapFeature::GetWriteMode

**BOOL GetWriteMode(SapFeature::WriteMode\* writeMode);**

### Parameters

<i>writeMode</i>	Returned write mode can be one of the following values:
SapFeature::WriteUndefined	Undefined write mode
SapFeature::WriteAlways	The feature can always be written
SapFeature::WriteNotAcquiring	The feature can only be written when the transfer object is not acquiring. If the transfer is currently acquiring you must stop the acquisition using the SapTransfer.Freeze or SapTransfer.Wait methods before modifying the feature value.
SapFeature::WriteNotConnected	The feature can only be written when the transfer object is not connected. If the transfer is currently connected you must disconnect it using the SapTransfer.Disconnect or SapTransfer.Destroy method before modifying the feature value. After modifying the value reconnect the transfer object using the SapTransfer.Connect or SapTransfer.Create method.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Checks if a feature can be modified when the corresponding transfer object (SapTransfer) is connected and/or acquiring. This transfer object is the one which uses the SapAcqDevice object from which the feature object was read.

Some features like buffer dimensions cannot be changed while data is being transferred to the buffer. Use this information to prevent an application from changing certain features when the transfer object is connected and/or acquiring.

### Demo/Example Usage

Not available

---

### SapFeature::IsEnumEnabled

BOOL **IsEnumEnabled**(int *enumIndex*, BOOL\* *enabled*);

#### Parameters

*enumIndex* Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)

*enabled* Returned item enabled value (TRUE or FALSE)

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Checks if the enumeration value corresponding to a specified index is enabled.

Each item in an enumeration is present for all the application duration. However an enumeration item may be dynamically enabled/disabled according to the value of another feature. Use this function to find out the enable state of an item at a given time.

#### Demo/Example Usage

Not available

---

### SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile

BOOL **IsSavedToConfigFile**(BOOL\* *savedToConfigFile*);

BOOL **SetSavedToConfigFile**(BOOL *savedToConfigFile*);

#### Parameters

*savedToConfigFile* TRUE for allowing the feature to be saved, FALSE otherwise.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Checks if a feature is saved to a CCF configuration file when calling the SapAcqDevice::SaveFeatures method.

All features are assigned a default behavior. For example, the read-only features are not saved while the read/write features are. You can, however, change the default behavior. For example a read-only feature such as 'InternalTemperature' is not saved by default. You can set *savedToConfigFile* to TRUE to force the feature to be written to the configuration file.

If you force read-only features to be saved those features will not be restored when loading back the CCF file. The reason is that the features are not writable to the device.

For acquisition devices which are not supported through the Genie Framework, the features saved to the configuration file are hardcoded and cannot be changed. Therefore these functions have no effect and always return FALSE.

#### Demo/Example Usage

Not available

---

### SapFeature::IsSelector

BOOL **IsSelector**(BOOL\* *isSelector*);

#### Parameters

*isSelector* Returns TRUE if the current feature is a selector, FALSE otherwise

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Determines if the value of the current feature directly affects the values of other features, using a one to many parent-child relationship. For example, if the current feature represents a look-up table index, then the affected features could represent values associated with one specific look-up table.

In this case, the current feature is called the selector.

Use the following methods to find out which features are associated: GetSelectedFeatureCount, GetSelectedFeatureIndex, and GetSelectedFeatureName.

You can only call IsSelector after the Create method

#### Demo/Example Usage

Not available

---

### SapFeature::IsStandard

BOOL **IsStandard**(BOOL\* *isStandard*);

#### Parameters

*isStandard* Returns whether the feature is standard (TRUE) or not (FALSE).

#### Return Value

Returns TRUE if successful, FALSE otherwise

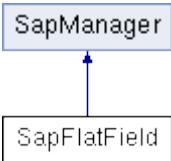
#### Remarks

Checks if a feature is standard or custom. Most of the features are standard. However, sometimes custom features might be provided as part of a special version of an acquisition device driver.

#### Demo/Example Usage

Not available

# SapFlatField



The purpose of the SapFlatField Class is to support flat field correction on monochrome images. The first scenario is where images are acquired from a camera. Flat field correction is then performed either by the acquisition device (if supported) or through software. The second scenario is where images are taken from another source (for example, loaded from disk). Only the software implementation is then available.

Flat field correction is the process of eliminating small gain differences between pixels in a sensor, eliminate sensor hotspots by automatically doing pixel replacement, and also to compensate for light distortion caused by a lens. Flat field correction data is composed of gain and offset coefficients for each pixel. A sensor exposed to a uniformly lit field will have no graylevel differences between pixels when calibrated flat field correction is applied to the image.

```
#include <SapClassBasic.h>
```

## SapFlatField Class Members

### Construction

<u>SapFlatField</u>	Class constructor
<u>Create</u>	Allocates the internal resources
<u>Destroy</u>	Releases the internal resources

### Attributes

<u>GetAcquisition</u> ,	Gets/sets the acquisition object for acquiring images and flat-field correction
<u>SetAcquisition</u>	
<u>GetAcqDevice</u> ,	Gets/sets the acquisition device object for acquiring images and flat-field correction
<u>SetAcqDevice</u>	
<u>GetBuffer</u> ,	Gets/sets the buffer object for operating the flat-field correction without an acquisition object
<u>SetBuffer</u>	
<u>GetBufferOffset</u> ,	Gets the buffer objects for the flat-field correction gain and offset coefficients
<u>GetBufferGain</u>	
<u>IsClippedGainOffsetDefects</u>	Checks if pixels with gain or offset coefficients that reach hardware limitations are considered to be defective
<u>.IsEnabled</u>	Checks if flat-field correction is enabled
<u>IsPixelReplacement</u>	Checks if replacement of defective pixels is enabled
<u>IsSoftware</u>	Checks if flat-field correction is performed in software or using the hardware
<u>GetCorrectionType</u> ,	Gets/sets line scan vs area scan correction type
<u>SetCorrectionType</u>	
<u>GetVideoType</u> ,	Gets/sets the acquisition video type (monochrome or color)
<u>SetVideoType</u>	
<u>GetBlackPixelPercentage</u> ,	Gets/sets allowed percentage of black pixels (value 0) in an image when flat field calibration is done
<u>SetBlackPixelPercentage</u>	
<u>GetDeviationMaxBlack</u> ,	Gets/sets the maximum deviation of the calculated coefficients towards black
<u>SetDeviationMaxBlack</u>	
<u>GetDeviationMaxWhite</u> ,	Gets/sets the maximum deviation of the calculated coefficients towards white
<u>SetDeviationMaxWhite</u>	
<u>GetGainDivisor</u> ,	Gets/sets the factor by which a gain coefficient has to be divided for getting a unitary scale factor.
<u>SetGainDivisor</u>	
<u>GetGainBase</u> ,	Gets/sets the gain base used when calculating the gain coefficients

<u><a href="#">SetGainBase</a></u>	
<u><a href="#">GetOffsetFactor</a></u>	Gets/sets the multiplication factor applied to the offset coefficients
<u><a href="#">SetOffsetFactor</a></u>	
<u><a href="#">GetOffsetMinMax</a></u>	Gets / sets the minimum and maximum values for computed offset values
<u><a href="#">SetOffsetMinMax</a></u>	
<u><a href="#">GetGainMinMax</a></u>	Gets / sets the minimum and maximum values for computed gain values
<u><a href="#">SetGainMinMax</a></u>	
<u><a href="#">GetNumLinesAverage</a></u>	Gets/sets the number of lines to be averaged in the image used for doing the calibration before computing the gain and offset coefficients for linescan video source.
<u><a href="#">SetNumLinesAverage</a></u>	
<u><a href="#">GetNumFramesAverage</a></u>	Gets/sets the number of frames to average for the calibration before computing the gain and offset coefficients for linescan video source.
<u><a href="#">SetNumFramesAverage</a></u>	
<u><a href="#">SetRegionOfInterest</a></u>	Specifies the ROI of coefficients to use for software flat-field correction.
<u><a href="#">ResetRegionOfInterest</a></u>	Resets the ROI to the full image size.
<u><a href="#">GetVerticalOffset</a></u>	Gets/sets the vertical line scan averaging offset in a full frame
<u><a href="#">SetVerticalOffset</a></u>	
<b>Operations</b>	
<u><a href="#">Load</a></u>	Loads gain and offset buffer data from disk files or from existing buffer objects
<u><a href="#">Save</a></u>	Saves gain and offset buffer data to disk files
<u><a href="#">Clear</a></u>	Clears the gain and offset buffers
<u><a href="#">ReadGainOffsetFromDevice</a></u>	Gets the current flat-field correction coefficients from the acquisition hardware
<u><a href="#">ComputeOffset</a></u>	Calculates the flat-field correction offset coefficients
<u><a href="#">ComputeGain</a></u>	Calculates the flat-field correction gain coefficients
<u><a href="#">Enable</a></u>	Enables/disables flat-field correction
<u><a href="#">EnableClippedGainOffsetDefects</a></u>	Enables/disables whether to consider pixels as defective when calculated gain or offset coefficients reach the hardware limitations.
<u><a href="#">EnablePixelReplacement</a></u>	Enables/disables replacement of defective pixels
<u><a href="#">Execute</a></u>	Performs the software implementation of flat-field correction
<u><a href="#">GetAverage</a></u>	Gets average pixel value and standard deviation for a buffer
<u><a href="#">GetStats</a></u>	Gets statistics for a buffer subtracted from the offset buffer

## SapFlatField Member Functions

The following are members of the SapFlatField Class.

---

### SapFlatField::SapFlatField

```
SapFlatField();
SapFlatField(SapAcquisition* pAcq);
SapFlatField(SapAcqDevice* pAcqDevice);
SapFlatField(SapBuffer* pBuffer);
```

#### Parameters

<i>pAcq</i>	SapAcquisition object to be used for image acquisition and for flat-field correction (if available in hardware). This object typically corresponds to a frame grabber.
<i>pAcqDevice</i>	SapAcqDevice object to be used for image acquisition and for flat-field correction (if available in hardware). This object typically corresponds to a Teledyne DALSA camera, for example, Genie.
<i>pBuffer</i>	SapBuffer object to be used to find out the width, height and format for the flat-field correction gain and offset buffer objects

#### Remarks

The SapFlatField constructor does not actually create the internal resources. To do this, you must call the Create method.

By default, there is only one buffer pair for gain and offset coefficients. However, multi-flat field capability is available with the Sapera PowerPack package (contact sales for more information).

The constructor with a SapBuffer object is used only for offline operation (no acquisition device), so that only software correction will be available.

#### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

### SapFlatField::Clear

**BOOL Clear();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Clears the flat-field correction gain and offset coefficients buffers. The gain coefficients are initialized for getting a unitary scale factor while the offset coefficients are initialized to 0.

#### Demo/Example Usage

Not available

---

### SapFlatField::ComputeGain

**BOOL ComputeGain(SapBuffer\* pBuffer, SapFlatFieldDefects\* pDefects,  
                  BOOL bUseImageMaxValue = TRUE, int numImages = 0);**

**BOOL ComputeGain(SapBuffer\* pBuffer, SapFlatFieldDefects\* pDefects, SapData target);**

#### Parameters

<i>pBuffer</i>	Pointer to a buffer object containing one or more calibration image(s)
<i>pDefects</i>	Pointer to a SapFlatFieldDefects object
<i>bUseImageMaxValue</i>	Indicates how to calculate the range of the calibrated output images
<i>numImages</i>	Number of images contained in <i>pBuffer</i> to be used for calibration (obsolete)
<i>target</i>	Maximum pixel target value for gain coefficient calculation

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Calculates the flat-field correction gain coefficients from one or more calibration image(s).

If *bUseImageMaxValue* is TRUE, then this method uses the highest actual pixel value of the input buffer as the maximum output value. Otherwise, it uses the highest possible pixel value, according to the pixel depth (see the SapBuffer::GetPixelDepth, SapBuffer::SetPixelDepth method).

The *target* parameter allows application code to specify the maximum output pixel value target for the gain. For flat-field correction on monochrome images, specify a SapDataMono object for this parameter. For color images, use a SapDataRGB object with target values for each color channel.

When this method returns, the SapFlatFieldDefects object pointed to by *pDefects* contains statistics about the defects found in the gain image. It has the following attributes:

<code>int GetNumDefects()</code>	Number of defective pixels
<code>int GetNumClusters()</code>	Number of defective pixels that are adjacent
<code>float GetDefectRatio()</code>	Ratio between defective pixels and good pixels in percent

#### Note

The *numImages* argument is now obsolete, it has been replaced by the SapFlatField::GetNumFramesAverage, SapFlatField::SetNumFramesAverage functions. However, for backwards compatibility, if this argument is set to any other value than the default (0), it will override the value set by the SetNumFramesAverage function.

#### Demo/Example Usage

Not available

---

### SapFlatField::ComputeOffset

**BOOL ComputeOffset(SapBuffer\* pBuffer, int numImages = 0);**

**Parameters**

*pBuffer* Pointer to buffer object containing a calibration image  
*numImages* Indicates the number of images contained in *pBuffer* to be used for calibration (obsolete)

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Calculates the flat-field correction offset coefficients from a calibration image.

You must call this method before the ComputeGain method.

**Note**

The *numImages* argument is now obsolete, it has been replaced by the SapFlatField::GetNumFramesAverage, SapFlatField::SetNumFramesAverage functions. However, for backwards compatibility, if this argument is set to any other value than the default (0), it will override the value set by the SetNumFramesAverage function.

**Demo/Example Usage**

Not available

---

### SapFlatField::Create

**BOOL Create();**

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Creates all the internal resources needed by the flat-field correction object

**Demo/Example Usage**

FlatField Demo, GigE FlatField Demo

---

### SapFlatField::Destroy

**BOOL Destroy();**

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Destroys all the internal resources needed by the flat-field correction object

**Demo/Example Usage**

FlatField Demo, GigE FlatField Demo

---

### SapFlatField::Enable

**BOOL Enable(BOOL enable = TRUE, BOOL useHardware = TRUE);**

**Parameters**

*Enable* TRUE to enable flat-field correction, FALSE to disable it  
*useHardware* TRUE to use hardware correction, FALSE to use the software implementation

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

Enables/disables flat-field correction. If you set *useHardware* to TRUE and hardware correction is not available, then this method returns FALSE. If you set *useHardware* to FALSE, then you must call the Execute method to perform the actual correction.

**Demo/Example Usage**

FlatField Demo, GigE FlatField Demo

---

### SapFlatField::EnableClippedGainOffsetDefects

**BOOL EnableClippedGainOffsetDefects(BOOL enable = TRUE);**

### Parameters

*Enable* TRUE to consider these pixels as defects, FALSE to disable it

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Enables/disables assignment of a defective flag to pixels that have gain or offset coefficients that reach or go beyond the supported limits of the hardware or software used to perform the flat-field correction.

If the value of this attribute is TRUE (its initial value), the chosen method to handle defective pixels will be performed. If FALSE, the gain and offset coefficients for those pixels will be used as-is.

### Demo/Example Usage

Not available

---

## SapFlatField::EnablePixelReplacement

BOOL **EnablePixelReplacement**(BOOL *enable* = TRUE);

### Parameters

*Enable* TRUE to enable pixel replacement, FALSE to disable it

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Enables/disables replacement of defective pixels.

Pixel replacement is used when calling the Execute method to perform the software implementation of flat-field correction. If TRUE, then defective pixel values are replaced by the value of a neighboring pixel. This is usually the one to the left of the current pixel, except for the first column, where the value of the pixel to the right is used instead.

### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapFlatField::Execute

BOOL **Execute**(SapBuffer \**pBuffer*);

BOOL **Execute**(SapBuffer \**pBuffer*, int *bufIndex*);

### Parameters

*pBuffer* Pointer to a buffer object for performing flat-field correction

*bufIndex* Buffer resource index

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Performs the software implementation of flat-field correction. If no buffer index is specified, the current index is assumed.

For each pixel, flat-field correction is performed according to the following formula:

$$\text{correctedValue} = (\text{originalValue} - \text{offset}) * (\text{gain} / \text{gainDivisor} + \text{gain base})$$

For 8-bit gain coefficients, the gain divisor is typically equal to 128, so that a gain value between 0 and 255 becomes a value between 0 and 2. Use the SetGainDivisor method to change its value.

### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapFlatField::GetAcquisition, SapFlatField::SetAcquisition

SapAcquisition\* **GetAcquisition**();

BOOL **SetAcquisition**(SapAcquisition\* *pAcq*);

### Remarks

Gets/sets the SapAcquisition object to be used for image acquisition and for flat-field correction. This object typically corresponds to a frame grabber.

You can only call SetAcquisition before the Create method.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetAcqDevice, SapFlatField::SetAcqDevice

```
SapAcqDevice* GetAcqDevice();
BOOL SetAcqDevice(SapAcqDevice* pAcqDevice);
```

#### Remarks

Gets/sets the SapAcqDevice object to be used for image acquisition and for flat-field correction. This object typically corresponds to a Teledyne DALSA camera, for example, Genie.

You can only call SetAcquisition before the Create method.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetAverage

```
BOOL GetAverage(SapBuffer* pBuffer, SapFlatFieldStats* pStats);
```

#### Parameters

*pBuffer* Pointer to buffer object from which to compute the average  
*pStats* Pointer to a SapFlatFieldStats object for returned statistics

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets average pixel value and standard deviation for a buffer. See the GetStats method for details about the SapFlatFieldStats class.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetBlackPixelPercentage, SapFlatField::SetBlackPixelPercentage

```
float GetBlackPixelPercentage();
BOOL SetBlackPixelPercentage(float percentage);
```

#### Parameters

*percentage* Percentage of black pixels tolerated

#### Remarks

Gets/sets the allowed percentage of black pixels (with value 0) in an image when flat field calibration is done. You must set the value of this attribute before calling the ComputeOffset and ComputeGain methods. The actual result may be better than the requested percentage but never worse.

The initial value for this attribute is 2.0.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetBuffer, SapFlatField::SetBuffer

```
SapBuffer* GetBuffer();
BOOL SetBuffer(SapBuffer* pBuffer);
```

#### Remarks

Gets/sets the buffer object for operating the flat-field correction without an acquisition object. It is used to find out the width, height and format for the flat-field correction gain and offset buffer objects.

You can only call SetBuffer before the Create method.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetBufferOffset, SapFlatField::GetBufferGain

```
SapBuffer* GetBufferOffset();
SapBuffer* GetBufferGain();
BOOL GetBufferOffset(SapBuffer* pBuffer, int buflIndex = 0, int offsetIndex = 0);
BOOL GetBufferGain(SapBuffer* pBuffer, int buflIndex = 0, int gainIndex = 0);
```

#### Parameters

<i>pBuffer</i>	Existing buffer object for retrieving a copy of the offset or gain buffer data
<i>buflIndex</i>	Buffer resource index for <i>pBuffer</i> argument
<i>offsetIndex</i>	Offset buffer resource index
<i>gainIndex</i>	Gain buffer resource index

#### Return Value

The methods with a *pBuffer* and *offsetIndex/gainIndex* arguments return TRUE if successful, FALSE otherwise

#### Remarks

The methods with no arguments respectively retrieve a pointer to the SapBuffer objects that contains the flat-field correction gain and offset coefficients.

The methods with the *pBuffer* argument can be used to copy the contents of the gain or offset buffer to an application supplied buffer with a different data format. For example, it may be required to retrieve the 8-bit version of a 10-bit gain buffer. In this case, if the supplied buffer objects have different data formats, automatic data conversion takes place whenever possible, with clipping to maximum destination pixel values in case of overflow.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetCorrectionType, SapFlatField::SetCorrectionType

```
SapFlatField::CorrectionType GetCorrectionType();
BOOL SetCorrectionType(SapFlatField::CorrectionType correctionType);
```

#### Parameters

<i>correctionType</i>	Flat-field correction type may be one of the following values
SapFlatField::CorrectionTypeField	Correction is performed on full frames
SapFlatField::CorrectionTypeLine	Correction is performed on individual lines
SapFlatField::CorrectionTypeInvalid	Invalid correction type

#### Remarks

Gets/sets the flat-field correction type.

The initial value for this attribute is SapFlatField::CorrectionTypeInvalid. It is then set according to the acquisition device scan type when calling the Create method. This means that calling SetCorrectionType is only relevant when no acquisition device is available, that is, when the SapFlatField constructor with a SapBuffer argument has been used for the current object.

#### Demo/Example Usage

Not available

---

### SapFlatField::GetDeviationMaxBlack, SapFlatField::SetDeviationMaxBlack

```
int GetDeviationMaxBlack();
BOOL SetDeviationMaxBlack(int deviationMax);
```

#### Remarks

Gets/sets the maximum deviation of the calculated coefficients from the average value towards the black pixel value so a pixel is not considered as being defective

The initial value for this attribute is 0. It is then set to 25% of the highest possible pixel value when calling the Create method. This pixel value is calculated either from the acquisition device pixel depth, or from the input buffer pixel depth, depending on which version of the SapFlatField constructor was used.

The maximum deviation value is used when calculating flat-field correction gain coefficients with the ComputeGain method.

#### Demo/Example Usage

Not available

---

## **SapFlatField::GetDeviationMaxWhite, SapFlatField::SetDeviationMaxWhite**

```
int GetDeviationMaxWhite();
BOOL SetDeviationMaxWhite(int deviationMax);
```

### **Remarks**

Gets/sets the maximum deviation of the calculated coefficients from the average value towards the white pixel value so a pixel is not considered as being defective.

The initial value for this attribute is 0. It is then set to 25% of the highest possible pixel value when calling the Create method. This pixel value is calculated either from the acquisition device pixel depth, or from the input buffer pixel depth, depending on which version of the SapFlatField constructor was used.

The maximum deviation value is used when calculating flat-field correction gain coefficients with the ComputeGain method.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetGainBase, SapFlatField::SetGainBase**

```
int GetGainBase();
BOOL SetGainBase(int gainBase);
```

### **Remarks**

Gets/sets the gain base used when calculating the gain coefficients.

When using a Teledyne DALSA acquisition device which support hardware-based gain base (for example, Genie TS), then the initial value for this attribute is only meaningful after calling the Create method, since it is retrieved from the acquisition hardware itself. In this case, application code should not call SetGainBase at all.

For all other acquisition devices, and also for software based flat-field correction, the initial value for this attribute is 0, and application code can call SetGainBase if required.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetGainDivisor, SapFlatField::SetGainDivisor**

```
int GetGainDivisor();
BOOL SetGainDivisor(int gainDivisor);
```

### **Remarks**

Gets/sets the factor by which the gain coefficients have to be divided for getting a unitary scale factor.

The initial value for this attribute is 128. It is then set to the acquisition device gain divisor value when calling the Create method.

The SetGainDivisor method should only be used when operating without hardware support.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetGainMinMax, SapFlatField::SetGainMinMax**

```
void GetGainMinMax(int* pGainMin, int* pGainMax);
BOOL SetGainMinMax(int gainMin, int gainMax);
```

### **Parameters**

*pGainMin* Pointer to returned minimum gain value

*pGainMax* Pointer to returned maximum gain value

### **Remarks**

Gets/sets the minimum and maximum resulting values when computing gain values using the ComputeGain method.

This is useful when computing the gain values for an acquisition device that has known limitations on these values.

The initial value for these attributes are 0 and 255.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetNumFramesAverage, SapFlatField::SetNumFramesAverage**

```
int GetNumFramesAverage();
BOOL SetNumFramesAverage(int numFramesAverage);
```

### **Remarks**

Gets/sets the number of frames to be averaged before computing the flat-field correction gain and offset coefficients for an areascan video source. The initial value for this attribute is 10. You must call SetNumFramesAverage before the ComputeOffset and ComputeGain methods.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetNumLinesAverage, SapFlatField::SetNumLinesAverage**

```
int GetNumLinesAverage();
BOOL SetNumLinesAverage(int numLinesAverage);
```

### **Remarks**

Gets/sets the number of lines to be averaged in the image used for doing the calibration before computing the flat-field correction gain and offset coefficients for linescan video source. The initial value for this attribute is 128. You must call SetNumFramesAverage before the ComputeOffset and ComputeGain methods.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetOffsetFactor , SapFlatField::SetOffsetFactor**

```
double GetOffsetFactor();
BOOL SetOffsetFactor(double offseFactor);
```

### **Parameters**

*offsetFactor* Sets the offset factor for the flat field offset coefficient. Possible values are hardware dependent; refer to the acquisition device documentation for more information.

### **Remarks**

Gets/sets the multiplication factor used when calculating flat field offset coefficients.

When using a Teledyne DALSA acquisition device which support hardware-based offset factor (e.g., Genie TS), then the initial value for this attribute is only meaningful after calling the Create method, since it is retrieved from the acquisition hardware itself. In this case, application code should not call SetOffsetFactor at all.

For all other acquisition devices, and also for software based flat-field correction, the initial value for this attribute is 1, and application code can call SetOffsetFactor if required.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetOffsetMinMax , SapFlatField::SetOffsetMinMax**

```
void GetOffsetMinMax(int* pOffsetMin, int* pOffsetMax);
BOOL SetOffsetMinMax(int offsetMin, int offsetMax);
```

### **Parameters**

*pOffsetMin* Pointer to returned minimum offset value

*pOffsetMax* Pointer to returned maximum offset value

### **Remarks**

Gets/sets the minimum and maximum resulting values when computing offset values using the ComputeOffset method.

This is useful when computing the offset values for an acquisition device that has known limitations on these values.

The initial value for these attributes are 0 and 255.

### **Demo/Example Usage**

Not available

---

## **SapFlatField::GetStats**

```
BOOL GetStats(SapBuffer* pBuffer, SapFlatFieldStats* pStats);
```

#### Parameters

<i>pBuffer</i>	Pointer to a buffer object from which to compute the statistics
<i>pStats</i>	Pointer to a SapFlatFieldStats object for returned statistics

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Calculates statistics about the image that are used internally to compute the flat-field correction gain and offset coefficients.

When the method returns, the SapFlatFieldStats object pointed to by *pStats* contains statistics about the image. It has the following attributes:

int GetNumComponents()	Returns the number of color components for which statistics were computed. For a monochrome image, it is 1. For a color image, it is 4, corresponding to the components of the color scheme of the camera (see the <a href="#">SapColorConversion::GetAlign</a> and <a href="#">SapColorConversion::SetAlign</a> methods).
int GetAverage()	Returns buffer average
int GetStdDeviation()	Returns buffer standard deviation
int GetPeakPosition()	Returns the peak value position in the histogram used to calculate the gain coefficients
int GetLow()	Returns the lower bound of the histogram. Pixels below the lower bound will be assigned a gain of 2.
int GetHigh()	Returns the higher bound of the histogram. Pixels above the higher bound will be assigned a gain of 1.
int GetNumPixels()	Returns the number of pixels in the histogram between the lower and the higher bounds
float GetPixelRatio()	Returns the ratio between the number of pixels inside the lower and the higher bound of the histogram and the number of pixels in the buffer in percent

All methods except GetNumComponents accept an optional iComponent argument that specifies the component index for which statistics are retrieved. If not specified, the value of this argument is 0, corresponding to the first component.

Note that only the GetNumComponents, GetAverage, and GetStdDeviation methods are relevant when the SapFlatFieldStats object is used in a call to the GetAverage method.

#### Demo/Example Usage

Not available

---

## SapFlatField::GetVerticalOffset, SapFlatField::SetVerticalOffset

```
int GetVerticalOffset();
BOOL SetVerticalOffset(int verticalOffset);
```

#### Parameters

*verticalOffset*      Vertical offset in lines

#### Remarks

Gets/sets the vertical line scan averaging offset in a full frame.

The initial value for this attribute is 0. This means that, for line scan acquisition, correction is performed on all lines. Specify a nonzero value if you need to skip a fixed number of lines at the beginning of each frame.

#### Demo/Example Usage

Not available

---

## SapFlatField::GetVideoType, SapFlatField::SetVideoType

```
SapAcquisition::VideoType GetVideoType();
BOOL SetVideoType(SapAcquisition::VideoType videoType, SapColorConversion::Align alignment);
```

#### Parameters

<code>videoType</code>	New acquisition video type (SapAcquisition::VideoMono or SapAcquisition::VideoColor)
<code>alignment</code>	Color alignment. Only used when <code>videoType</code> is set to SapAcquisition::VideoColor.
	Possible values are:
	SapColorConversion::AlignGBRG
	SapColorConversion::AlignBGGR
	SapColorConversion::AlignRGGB
	SapColorConversion::AlignGRBG
	SapColorConversion::AlignRGBG
	SapColorConversion::AlignBGRG

#### Remarks

Gets/sets the acquisition video type. The initial value for this attribute is monochrome. If the current flat-field object is associated with a SapAcquisition or SapAcqDevice object (see the SapFlatField constructor), then the value is set according to the acquisition video type when calling the Create method.

If the current flat-field object is not associated with an acquisition object, then the object will be used only for offline operation (no acquisition), so that only software correction will be available. In this case, you should call SetVideoType before the Create method.

#### Demo/Example Usage

Not available

### SapFlatField::IsClippedGainOffsetDefects

**BOOL IsClippedGainOffsetDefects();**

#### Remarks

Checks if pixels with gain or offset coefficient that reach or go beyond the maximum limit are considered to be defective. The initial value for this attribute is TRUE.

Use the EnableClippedGainOffsetDefects method if you need to explicitly enable or disable this behavior.

#### Demo/Example Usage

Not available

### SapFlatField::IsEnabled

**BOOL IsEnabled();**

#### Remarks

Checks if flat-field correction is enabled. The initial value for this attribute depends on the current acquisition device.

Use the Enable method if you need to explicitly enable or disable flat-field correction.

#### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

### SapFlatField::IsPixelReplacement

**BOOL IsPixelReplacement();**

#### Remarks

Checks if replacement of defective pixels is enabled.

Pixel replacement is used when calling the Execute method to perform the software implementation of flat-field correction. If it is TRUE, then defective pixel values are replaced by the value of a neighboring pixel. This is usually the one to the left of the current pixel, except for the first column, where the value of the pixel to the right is used instead.

The initial value for this attribute is TRUE.

Use the EnablePixelReplacement method if you need to explicitly enable or disable this feature.

#### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

### SapFlatField::IsSoftware

**BOOL IsSoftware();**

## Remarks

Checks if flat-field correction is performed in software or using the acquisition hardware. To check if your hardware supports on-board flat field correction, see SapAcquisition::IsFlatFieldAvailable or SapAcqDevice::IsFlatFieldAvailable. The SapFlatField::Enable *useHardware* parameter determines if hardware correction is used.

## Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapFlatField::Load

```
BOOL Load(const char* fileName);  
BOOL Load(SapBuffer* pBufferGain, SapBuffer* pBufferOffset);
```

### Parameters

*fileName* Name of the image file with the gain and offset parameters

*pBufferGain* Pointer to buffer object containing the gain values

*pBufferOffset* Pointer to buffer object containing the offset values

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Loads flat-field correction gain and offset coefficients buffers from disk files or from existing buffer objects.

The specified file must be in TIFF format, and contains the data for both buffers.

## Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapFlatField::ReadGainOffsetFromDevice

```
BOOL ReadGainOffsetFromDevice();
```

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Gets the current flat-field correction coefficients from the acquisition hardware (frame grabber or camera). These coefficients can then be accessed using the SapFlatField::GetBufferOffset, SapFlatField::GetBufferGain methods.

## Demo/Example Usage

Not available

---

## SapFlatField::ResetRegionOfInterest

```
BOOL ResetRegionOfInterest();
```

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Resets the ROI used for flat field calibration and correction to the full image size. The ROI is set using the SetRegionOfInterest method.

## Demo/Example Usage

Not available

---

## SapFlatField::Save

```
BOOL Save(const char* fileName);
```

### Parameters

*fileName* Name of the image file with the gain and offset parameters

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Saves flat-field correction gain and offset coefficients buffers to disk files. The specified file is always written in TIFF format, no matter which file extension you specify.

## Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapFlatField::SetRegionOfInterest

BOOL **SetRegionOfInterest**(int *leftOffset*, int *topOffset*, int *width*, int *height*);

### Parameters

*leftOffset* Left offset, in pixels, of the top left corner of the ROI.

*topOffset* Top offset, in pixels, of the top left corner of the ROI

*width* Width in pixels of the ROI.

*height* Height in pixels of the ROI

### Return Value

Returns TRUE if successful, FALSE otherwise

### Description

This method is relevant only for software flat field correction, that is, when the `IsSoftware` method returns TRUE. It specifies the area to process in the image buffer when you do not need to apply flat-field correction for the full camera sensor. This method must be called before `SapFlatField::Execute`. The ROI is also applied during the calibration phase when calling the `ComputeGain` method and `ComputeOffset` method, such that coefficients are only calculated for those pixels within the ROI. The ROI can be reset to the full image size using the `ResetRegionOfInterest` method.

Note, if the ROI is modified, coefficients must be recalculated.

## Demo/Example Usage

Not available

# SapGio



The purpose of the SapGio Class is to control a block of general inputs and outputs, that is, a group of I/Os that may be read and/or written all at once. For a TTL level type I/Os, its state is considered ON or active if the measured voltage on the I/O is 5V (typical).

This class may be used together with SapCounter to associate event counting with the state of specific I/O pins.

Note that acquisition devices do not all support general I/Os.

```
#include <SapClassBasic.h>
```

## SapGio Class Members

### Construction

<u>SapGio</u>	Class constructor
<u>Create</u>	Allocates the low-level Sapera resources
<u>Destroy</u>	Releases the low-level Sapera resources
<b>Attributes</b>	
<u>GetLocation</u>	Gets/sets the location where the I/O resource is located
<u>SetLocation</u>	
<u>SetCallbackInfo</u>	Sets the application callback method for I/O events and the associated context
<u>GetCallback</u>	Gets the current application callback method for I/O events
<u>GetContext</u>	Gets the application context associated with I/O events
<u>GetNumPins</u>	Gets the number of pins present on the I/O resource
<u>GetAvailPinConfig</u>	Gets the set of possible configurations for a specific I/O pin or all pins
<u>GetPinConfig</u>	Gets/sets the current configuration for a specific I/O pin or all pins
<u>SetPinConfig</u>	
<u>GetPinState</u>	Gets/sets the low/high state of a specific I/O pin or all pins
<u>SetPinState</u>	
<u>GetHandle</u>	Gets the low-level Sapera handle of the I/O resource
<b>Operations</b>	
<u>EnableCallback</u>	Allows an application callback function to be called at specific I/O events
<u>DisableCallback</u>	Disables calls to the application callback function
<u>AutoTrigger</u>	Automatically changes the state of an I/O pin for a specified duration
<u>IsCapabilityValid</u>	Checks for the availability of a low-level Sapera C library capability
<u>IsParameterValid</u>	Checks for the availability of a low-level Sapera C library parameter
<u>GetCapability</u>	Gets the value of a low-level Sapera C library capability
<u>GetParameter</u>	Gets/sets the value of a low-level Sapera C library parameter
<u>SetParameter</u>	

## SapGio Member Functions

The following are members of the SapGio Class.

---

### SapGio::SapGio

**SapGio(**  
    SapLocation *loc* = SapLocation::ServerSystem,  
    SapGioCallback *pCallback* = NULL,  
    void\* *pContext* = NULL  
);

#### Parameters

<i>loc</i>	SapLocation object specifying the server where the I/O resource is located and the index of the resource on this server
<i>pCallback</i>	Application callback function to be called each time an I/O event happens. The callback function must be declared as: void MyCallback(SapGioCallbackInfo * <i>pInfo</i> );
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL, this parameter is ignored.

#### Remarks

The SapGio constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

Specifying a callback function in the constructor does not automatically activate it after the call to the Create method. You must subsequently call the EnableCallback method in order to be notified of I/O events.

#### Demo/Example Usage

IO Demo

---

### SapGio::AutoTrigger

BOOL **AutoTrigger**(SapCounter\* *pCounter*, int *startCount*, int *stopCount*, int *pinMask*, int *pinState*);

#### Parameters

<i>pCounter</i>	Counter object that causes I/O state transitions when reaching <i>startCount</i> and <i>stopCount</i>
<i>startCount</i>	Count at which the I/O pins identified by <i>pinMask</i> will change state
<i>stopCount</i>	Count at which the I/O pins identified by <i>pinMask</i> will go back to their original state
<i>pinMask</i>	Bit field specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.
<i>pinState</i>	Bit field representing the state of I/O pins identified by <i>pinMask</i> when the counter resource reaches <i>startCount</i> . The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Bits that are set to 1 represent high, while 0 represents low.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Automatically triggers the state of one or more I/O pins at a specific time.

The I/O pins identified by *pinMask* are initially set to the opposite of the values represented by *pinState*. When the counter device reaches *startCount*, their state changes to the values represented by *pinState*. When the counter device reaches *stopCount*, their state goes back to the original values. This method is not available in Sapera LT for 64-bit Windows.

#### Demo/Example Usage

Not available

---

### SapGio::Create

BOOL **Create()**;

#### Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

## Remarks

Creates all the low-level Sapera resources needed by the I/O object

## Demo/Example Usage

IO Demo

---

## SapGio::Destroy

**BOOL Destroy();**

### Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

## Remarks

Destroys all the low-level Sapera resources needed by the I/O object

## Demo/Example Usage

IO Demo

---

## SapGio::DisableCallback

**BOOL DisableCallback(int pinNumber);**

**BOOL DisableCallback();**

### Parameters

*pinNumber* Pin number on the current I/O resource.

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Disables calls to the application callback function.

The first form of Disable callback only affects the specified I/O pin. The second form affects all pins.

See the SapGio constructor and the EnableCallback method for more details.

## Demo/Example Usage

Not available

---

## SapGio::EnableCallback

**BOOL EnableCallback(int pinNumber, SapGio::EventType eventType);**

**BOOL EnableCallback(int pinMask, SapGio::EventType\* pEventType);**

**BOOL EnableCallback(SapGio::EventType eventType);**

### Parameters

*pinNumber* Pin number on the current input I/O resource

*eventType* Type of I/O event that initiates calls to the application callback function, can be one of the following values:

SapGio::EventRisingEdge      Rising edge of I/O pin state transition (low to high)

SapGio::EventFallingEdge      Falling edge of I/O pin state transition (high to low)

*pinMask* Bit field specifying which input I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.

*pEventType* Pointer to event types array. This argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.

### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Allows an application callback function to be called at specific input I/O events. See the SapGio constructor for details about the application callback function.

The first form of EnableCallback may be used using a single input pin number and a corresponding event type. Use this method together with the version of DisableCallback which takes a pin number argument.

The second form takes a bit mask of affected input I/O pins, and an array *pEventType* to specify the event

associated with each pin. Entries in *pEventType* corresponding to bits set to 1 in the *pinMask* argument enable callbacks for the corresponding pins. Bits set to 0 in *pinMask* disable callbacks for the corresponding pins.

The third form enables callbacks for all input pins using the same event type. The drawback of using this form is that it will not be possible to uniquely identify the pin causing the I/O event when the callback function is called. Use this method together with the version of DisableCallback with no arguments.

#### Demo/Example Usage

Not available

---

### SapGio::GetAvailPinConfig

```
BOOL GetAvailPinConfig(int pinNumber, SapGio::PinConfig* pAvailPinConfig);  
BOOL GetAvailPinConfig(SapGio::PinConfig* pAvailPinConfig);
```

#### Parameters

<i>pinNumber</i>	Pin number on the current I/O resource
<i>pAvailPinConfig</i>	Pointer to available pin configurations, including one or more of the following (combined using bitwise OR)
SapGio::PinInput	I/O pin may be configured as an input
SapGio::PinOutput	I/O pin may be configured as an output
SapGio::PinTristate	I/O pin may be tri-stated
If no <i>pinNumber</i> is specified, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.	

#### Remarks

Gets the set of possible configurations for a specific I/O pin or all pins. The first form of this method takes a single pin number, and returns a single value through the *pAvailPinConfig* argument. The second form returns the configuration for all pins in the *pAvailPinConfig* array.

You can only call GetAvailPinConfig after the Create method.

#### Demo/Example Usage

Not available

---

### SapGio::GetCallback

```
SapGioCallback GetCallback();
```

#### Remarks

Gets the current application callback method for I/O events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapGio constructor for more details.

#### Demo/Example Usage

Not available

---

### SapGio::GetCapability

```
BOOL GetCapability(int cap, void* pValue);
```

#### Parameters

<i>param</i>	Low-level Sapera C library capability to read
<i>pValue</i>	Pointer to capability value to read back

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

This method allows direct read access to low-level Sapera C library capabilities for the I/O module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapGio Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

**SapGio::GetContext**

```
void* GetContext();
```

**Remarks**

Gets the application context associated with I/O events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapGio constructor for more details.

**Demo/Example Usage**

Not available

**SapGio::GetHandle**

```
CORHANDLE GetHandle();
```

**Remarks**

Gets the low-level Sapera handle of the I/O resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

**Demo/Example Usage**

Not available

**SapGio::GetLocation,SapGio::SetLocation**

```
SapLocation GetLocation();
```

```
BOOL SetLocation(SapLocation location);
```

**Remarks**

Gets/sets the location where the I/O resource is located. A specific server can also be specified through the SapGio constructor.

**Demo/Example Usage**

Not available

**SapGio::GetNumPins**

```
int GetNumPins();
```

**Remarks**

Gets the number of pins present on the I/O resource.

The initial value for this attribute is 0. It is then set to the I/O device pin count value when calling the Create method.

**Demo/Example Usage**

Not available

**SapGio::GetParameter, SapGio::SetParameter**

```
BOOL GetParameter(int param, void* pValue);
```

```
BOOL SetParameter(int param, int value);
```

```
BOOL SetParameter(int param, void* pValue);
```

**Parameters**

*param* Low-level Sapera C library parameter to read or write

*pValue* Pointer to parameter value to read back or to write

*value* New parameter value to write

**Return Value**

Returns TRUE if successful, FALSE otherwise

**Remarks**

These methods allow direct read/write access to low-level Sapera C library parameters for the I/O module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which

is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapGio class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapGio::GetPinConfig, SapGio::SetPinConfig

```
BOOL GetPinConfig(int pinNumber, SapGio::PinConfig* pPinConfig);
BOOL GetPinConfig(SapGio::PinConfig* pPinConfig);
BOOL SetPinConfig(int pinNumber, SapGio::PinConfig pinConfig);
BOOL SetPinConfig(SapGio::PinConfig* pPinConfig);
```

### Parameters

<i>pinNumber</i>	Pin number on the current I/O resource (from 0 to the value returned by the GetNumPins method, minus 1)
<i>pPinConfig</i>	Pointer to pin configuration. See SapGio::GetAvailPinConfig method for possible values. If no <i>pinNumber</i> is specified, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.
<i>pinConfig</i>	New pin configuration. See the SapGio::GetAvailPinConfig method for possible values.

### Remarks

Gets/sets the current configuration for a specific I/O pin or all pins.

The first form of GetPinConfig takes a single pin number and returns a single value through the *pPinConfig* argument. The second form returns the configuration for all pins in the *pPinConfig* array.

The first form of SetPinConfig may be used using a single pin number and a corresponding pin configuration. You may also set *pinNumber* to the special constant SapGio::AllPins to apply the specified *pinConfig* to all I/O pins. The second form of SetPinConfig allows all pins to be set to a different value through the *pPinConfig* array argument.

You can only call GetPinConfig and SetPinConfig after the Create method.

### Demo/Example Usage

Not available

---

## SapGio::GetPinState, SapGio::SetPinState

```
BOOL GetPinState(int pinNumber, SapGio::PinState* pPinState);
BOOL GetPinState(SapGio::PinState* pPinState);
BOOL SetPinState (int pinNumber, SapGio::PinState pinState);
BOOL SetPinState (int pinMask, SapGio::PinState* pPinState);
```

### Parameters

<i>pinNumber</i>	Pin number on the current I/O resource
<i>pPinState</i>	Pointer to pin state, can be one of the following values: SapGio::PinLow      The I/O pin is low SapGio::PinHigh     The I/O pin is high
	If no <i>pinNumber</i> is specified in GetPinState, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.
<i>pinState</i>	New pin state. See above for possible values.
<i>pinMask</i>	Bit mask specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.

### Remarks

Gets/sets the current state of a specific I/O pin or all pins.

The first form of GetPinState takes a single pin number and returns a single value through the *pPinState* argument. The second form returns the configuration for all pins in the *pPinState* array.

The first form of SetPinState may be used using a single pin number and a corresponding pin state. The second form takes a bit mask of affected I/O pins, and an array *pPinState* to specify the state of each pin. Only entries in *pPinState* corresponding to bits set to 1 in the *pinMask* argument are used.

You can only call GetPinState and SetPinState after the Create method.

#### Demo/Example Usage

Not available

---

### SapGio::IsCapabilityValid

**BOOL IsCapabilityValid(int cap);**

#### Parameters

*cap* Low-level Sapera C library capability to be checked

#### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library capability for the general I/O module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapGio class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

IO Demo

---

### SapGio::IsParameterValid

**BOOL IsParameterValid(int param);**

#### Parameters

*param* Low-level Sapera C library parameter to be checked

#### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library parameter for the general I/O module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapGio class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Not available

---

### SapGio::SetCallbackInfo

**BOOL SetCallbackInfo(SapGioCallback *pCallback*, void\* *pContext* = NULL);**

#### Remarks

Sets the application callback method for I/O events and the associated context. See the SapGio constructor for more details.

You can only call SetCallbackInfo before the EnableCallback method.

#### Demo/Example Usage

Not available

# SapGioCallbackInfo

The SapGioCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapGio Class.

```
#include <SapClassBasic.h>
```

## SapGioCallbackInfo Class Members

### Construction

[SapGioCallbackInfo](#) Class constructor

### Attributes

<u><a href="#">GetGio</a></u>	Gets the SapGio object associated with I/O events
<u><a href="#">GetCustomData</a></u>	Gets the data associated with a custom I/O event
<u><a href="#">GetCustomSize</a></u>	Gets the size of the custom data returned by GetCustomData
<u><a href="#">GetEventType</a></u>	Gets the I/O events that triggered the call to the application callback
<u><a href="#">GetEventCount</a></u>	Gets the current count of I/O events
<u><a href="#">GetEventInfo</a></u>	Gets the low-level Sapera handle of the event info resource
<u><a href="#">GetContext</a></u>	Gets the application context associated with I/O events
<u><a href="#">GetGenericParam0</a></u>	Gets generic parameters supported by some events
<u><a href="#">GetGenericParam1</a></u>	
<u><a href="#">GetGenericParam2</a></u>	
<u><a href="#">GetGenericParam3</a></u>	
<u><a href="#">GetPinNumber</a></u>	Get the I/O pin number that generated an I/O event
<u><a href="#">GetAuxiliaryTimestamp</a></u>	Gets the auxiliary timestamp associated with I/O events.
<u><a href="#">GetHostTimestamp</a></u>	Gets the host timestamp associated with I/O events.

## SapGioCallbackInfo Member Functions

The following are members of the SapGioCallbackInfo Class.

### SapGioCallbackInfo::SapGioCallbackInfo

```
SapGioCallbackInfo(  
    SapGio* pGio,  
    void* pContext,  
    SapGio::EventType eventType,  
    int eventCount  
    ...int pinNumber  
)
```

```
SapGioCallbackInfo(  
    SapGio *pGio,  
    void *pContext,  
    COREVENTINFO eventInfo,  
    int pinNumber  
)
```

### Parameters

<i>pGio</i>	SapGio object that calls the callback function
<i>pContext</i>	Pointer to the application context
<i>eventType</i>	Combination of I/O events. See SapGio::EnableCallback for a list a possible values
<i>eventCount</i>	Current I/O event count
<i>pinNumber</i>	Current I/O pin number
<i>eventInfo</i>	Low-level Sapera handle of the event info resource

## Remarks

SapGio objects create an instance of this class before each call to the I/O callback method, in order to combine all function arguments into one container.

SapGio uses this class for reporting of I/O events. The *pContext* parameter takes the value specified in the SapGio Class constructor, *eventType* identifies the combination of events that triggered the call to the callback function, *eventCount* increments by one at each call (starting at 1), and *pinNumber* identifies the I/O pin that had a state change.

## Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetAuxiliaryTimestamp

**BOOL GetAuxiliaryTimestamp(UINT64 \*auxTimestamp);**

#### Parameters

*auxTimestamp* Address of a pointer to receive the auxiliary timestamp

#### Remarks

Gets the auxiliary timestamp associated with I/O events. Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

## Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetContext

**void \*GetContext();**

#### Remarks

Gets the application context associated with I/O events. See the SapGio constructor for more details.

## Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetCustomData

**BOOL GetCustomData(void\*\* customData);**

#### Parameters

*customData* Address of a pointer to receive the address to the data buffer

#### Remarks

Gets the address of a buffer containing the data associated with a custom I/O event. You must not free the buffer after you are finished using it.

This functionality is usually not supported, except for special versions of certain acquisition devices. See the device User's Manual for more information on availability.

#### Example

```
void MyCallback(SapGioCallbackInfo* pInfo)
{
    // Retrieve the data buffer
    void* pCustomData;
    pInfo->GetCustomData(&pCustomData);

    // Use the data buffer
    //...
}
```

## Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetCustomSize

**BOOL GetCustomSize(int\* customSize);**

#### Parameters

*customSize* Address of an integer to return the value

#### Remarks

Gets the size of the custom data returned by the GetCustomData method.

#### Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetEventCount

```
int GetEventCount();
BOOL GetEventCount(int *eventCount);
```

#### Parameters

*eventCount* Address of an integer where the count is written

#### Remarks

Gets the current count of I/O events. The initial value is 1, and increments after every call to the I/O callback function.

#### Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetEventInfo

```
COREVENTINFO GetEventInfo();
```

#### Remarks

Gets the low-level Sapera handle of the event info resource. You should not use this method unless you need a handle to the low-level C API to access some functionality not exposed in the C++ API.

#### Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetEventType

```
SapGio::EventType GetEventType();
BOOL GetEventType(SapGio::EventType *eventType);
```

#### Parameters

*eventType* Pointer to the integer variable to hold the event type

#### Remarks

Gets the combination of I/O events that triggered the call to the application callback. See the SapGio constructor for the list of possible values.

#### Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetGio

```
SapGio *GetGio();
```

#### Remarks

Gets the SapGio object associated with I/O events. See the SapGio constructor for more details.

#### Demo/Example Usage

Not available

---

### SapGioCallbackInfo::GetGenericParam0

### SapGioCallbackInfo::GetGenericParam1

### SapGioCallbackInfo::GetGenericParam2

### SapGioCallbackInfo::GetGenericParam3

```
BOOL GetGenericParam0(int* paramValue);
BOOL GetGenericParam1(int* paramValue);
BOOL GetGenericParam2(int* paramValue);
BOOL GetGenericParam3(int* paramValue);
```

#### Parameters

*paramValue* Address of an integer where the parameter value is written

## Remarks

Gets any of the four generic parameters supported by some I/Or events. You should use aliases instead when they are available. See the acquisition device User's Manual for a list of transfer events using generic parameters.

## Demo/Example Usage

Not available

---

## SapGioCallbackInfo::GetHostTimestamp

BOOL **GetHostTimestamp**(UINT64 \**hostTimestamp*);

### Parameters

*hostTimestamp* Address of a pointer to receive the host timestamp

## Remarks

Gets the host timestamp associated with I/O events. When a registered event is raised, the host timestamp is retrieved from the host CPU at the kernel level before the callback function executes at the application level.

Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

## Demo/Example Usage

Not available

---

## SapGioCallbackInfo::GetPinNumber

int **GetPinNumber**();

### Remarks

Get the I/O pin number that generated an I/O event.

If this number is equal to the special constant SapGio::AllPins, the pin then cannot be uniquely identified. In this case, use the SapGio::GetState method to get the required pin information.

## Demo/Example Usage

Not available

# SapLocation

The SapLocation Class identifies a Sapera server/resource pair.

A Sapera server is an abstract representation of a physical device like a frame grabber, a processing board, a GigE camera, or the host computer. In general, a Teledyne DALSA board or GigE camera camera is a server. Resources are attached to these physical devices. For example, a frame grabber can have one or more acquisition resources.

Sapera Class methods do not always need the server information from SapLocation. In these cases, the resource index is simply ignored.

```
#include <SapClassBasic.h>
```

## SapLocation Class Members

### Construction

[SapLocation](#) Class constructor

### Attributes

[GetServerIndex](#) Gets the server index

[GetServerName](#) Gets the server name

[GetResourceIndex](#) Gets the resource index

[IsUnknown](#) Checks if neither the server index nor the server name is valid

## SapLocation Member Functions

The following are members of the SapLocation Class.

---

### SapLocation::SapLocation

```
SapLocation();
SapLocation(int serverIndex, int resourceIndex = 0);
SapLocation(const char *serverName, int resourceIndex = 0);
SapLocation(const SapLocation &loc);
SapLocation(const SapLocation &loc, int resourceIndex);
```

### Parameters

*serverIndex* Sapera server index. There is always one server associated with the host computer at SapLocation::ServerSystem (index 0).

*serverName* Sapera server name. The 'System' server is associated with the host computer.

*resourceIndex* Sapera resource index

*loc* Existing SapLocation object from which server and resource information are to be extracted.

### Remarks

Use the Sapera Configuration utility to find the names and indices of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

The constructor with only the *loc* argument allows you to reuse the same server and resource information. The constructor with both *loc* and *resourceIndex* allows use to reuse the same server information, while specifying a different resource index.

### Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, IO Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

---

### SapLocation::GetResourceIndex

```
int GetResourceIndex();
```

### Remarks

Returns the resource index.

## **Demo/Example Usage**

Not available

---

### **SapLocation::GetServerIndex**

`int GetServerIndex();`

#### **Remarks**

Returns the server index. If the returned value is equal to SapLocation::ServerUnknown, it does not necessarily mean that the object is invalid. In this case, use the GetServerName method instead.

## **Demo/Example Usage**

Not available

---

### **SapLocation::GetServerName**

`char* GetServerName();`

#### **Remarks**

Returns the server name. If the returned value is an empty string, it does not necessarily mean that the object is invalid. In this case, use the GetServerIndex method instead.

## **Demo/Example Usage**

Not available

---

### **SapLocation::IsUnknown**

`BOOL IsUnknown();`

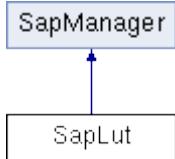
#### **Remarks**

Returns TRUE if neither the server index nor the server name is valid, FALSE otherwise. Although an unknown SapLocation is usually invalid, it may be useful in some particular cases.

## **Demo/Example Usage**

Not available

# SapLut



The SapLut Class implements lookup table management. Although you may create and destroy SapLut objects explicitly in application code, you usually do not have to do this.

If you need to manipulate acquisition lookup tables on a frame grabber, first call the SapAcquisition::GetLut method [\\_SapAcquisition.Luts\\_Property](#) to get a valid SapLut object. You may then manipulate the LUT through the methods in this class, and reprogram it using SapAcquisition::ApplyLutmethod.

If you need to manipulate lookup tables on an acquisition device controlled through the SapAcqDevice class (for example, a Genie camera), use the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue methods, both of which provide versions with a SapLut argument.

If you need to manipulate display lookup tables, you may use the same technique, but using the SapView::GetLut [\\_SapView.Lut\\_Property](#) and SapView::ApplyLut [\\_SapView.ApplyLut\\_Method](#) methods instead.

```
#include <SapClassBasic.h>
```

## SapLut Class Members

### Construction

[SapLut](#) Class constructor

[Create](#) Allocates the low-level Saper resources

[Destroy](#) Releases the low-level Saper resources

### Attributes

[GetLocation](#), [SetLocation](#) Gets/sets the location where the LUT resource is located

[GetNumEntries](#), [SetNumEntries](#) Gets/sets the number of LUT entries

[GetFormat](#), [SetFormat](#) Gets/sets the LUT data format

[GetElementSize](#) Gets the number of bytes required to store a single LUT element

[GetNumPages](#) Gets the number of color planes in the LUT

[IsSigned](#) Checks if the LUT contains signed or unsigned data

[GetTotalSize](#) Gets the total number of bytes required to store LUT data

[GetHandle](#) Gets the low-level Saper handle of the LUT resource

### Operations

[Read](#) Reads one or more elements from LUT storage to user-allocated memory

[Write](#) Writes one or more elements from user-allocated memory to LUT storage

[Arithmetic](#) Modifies all LUT entries using an arithmetic operation

[BinaryPattern](#) Modifies some LUT entries based on a binary pattern

[Boolean](#) Modifies all LUT entries using a Boolean operation

[Gamma](#) Modifies all LUT entries using Gamma correction

[Normal](#) Modifies all LUT entries using a linear mapping with a positive slope

[Reverse](#) Modifies all LUT entries using a linear mapping with a negative slope

[Roll](#) Relocates LUT entries upwards or downwards as one block

[Shift](#) Modifies all LUT entries using a logical shift

[Slope](#) Modifies part of a LUT with a linear mapping

[Threshold](#) Modifies all LUT entries using a threshold operation

<u>Copy</u>	Copies all LUT entries to another LUT resource
<u>Load</u>	Loads LUT entries from a file
<u>Save</u>	Saves LUT entries to a file
<u>GetParameter</u>	Gets/sets the value of a low-level Sapera C library parameter
<u>SetParameter</u>	

## SapLut Member Functions

The following are members of the SapLut Class.

---

### SapLut::SapLut

```
SapLut(
    int numEntries = 256,
    SapFormat format = SapFormatUInt8,
    SapLocation loc = SapLocation::ServerSystem
);
SapLut(
    const char *filename,
    SapLocation loc = SapLocation::ServerSystem
);
```

#### Parameters

<i>numEntries</i>	Number of LUT entries
<i>format</i>	Data format for the LUT resource, can be one of the following values.
	<b>Monochrome (unsigned)</b>
SapFormatMono8	8-bit
SapFormatMono9	9-bit
...	...
SapFormatMono15	15-bit
SapFormatMono16	16-bit
	<b>Monochrome (unsigned)</b>
SapFormatInt8	8-bit
SapFormatInt9	9-bit
...	...
SapFormatInt15	15-bit
SapFormatInt16	16-bit
	<b>Color (non-interlaced)</b>
SapFormatColorNI8	8-bit
SapFormatColorNI9	9-bit
...	...
SapFormatColorNI15	15-bit
SapFormatColorNI16	16-bit
	<b>Color (interlaced)</b>
SapFormatColorI8	8-bit
SapFormatColorI9	9-bit
...	...
SapFormatColorI15	15-bit
SapFormatColorI16	16-bit
<i>loc</i>	SapLocation object specifying the server where the LUT resource is located and the index of the resource on this server
<i>filename</i>	String containing the name of a Sapera LUT file from which to extract the <i>numEntries</i> and <i>format</i> parameters.

#### Remarks

The SapLut constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

For non-interlaced color formats, the red/green/blue components for one LUT element are stored separately:

RRR ... RRR	Red components of all elements
-------------	--------------------------------

GGG ... GGG	Green components of all elements
BBB ... BBB	Blue components of all elements
For interlaced color formats, the red/green/blue components for one LUT element are stored together:	
RGBRGBRGB	First three elements
...	...
RGBRGBRGB	Last three elements

The constructor is automatically called from the SapAcquisition Class so that acquisition lookup tables may be managed automatically. You just need to call the SapAcquisition::GetLut and SapAcquisition::ApplyLut methods, together with the functionality in this class, for all required LUT manipulations.

If you need to manage the LUTs for acquisition hardware which uses the SapAcqDevice class (for example, a Genie camera), use the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue methods, both of which provide versions with a SapLut argument.

The SapView Class also manages display LUTs automatically in a similar way to SapAcquisition.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Arithmetic

BOOL **Arithmetic**(SapLut::ArithmeticOp *operation*, SapData *value*);

### Parameters

<i>operation</i>	Specifies how to modify LUT data elements. The following operations are available:
SapLut::Add	Addition with saturation: $\text{data}[\text{index}] = \min(\text{maxValue}, \text{data}[\text{index}] + \text{value})$
SapLut::Asub	Absolute subtraction: $\text{data}[\text{index}] = \text{abs}(\text{data}[\text{index}] - \text{value})$
SapLut::Max	Maximum value: $\text{data}[\text{index}] = \max(\text{data}[\text{index}], \text{value})$
SapLut::Min	Minimum value: $\text{data}[\text{index}] = \min(\text{data}[\text{index}], \text{value})$
SapLut::Scale	Scale to smaller maximum value: $\text{data}[\text{index}] = (\text{data}[\text{index}] * \text{value}) / \text{maxValue}$
SapLut::Sub	Subtraction with saturation: $\text{data}[\text{index}] = \max(\text{minValue}, \text{data}[\text{index}] - \text{value})$
<i>value</i>	Source value object

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies all LUT entries using an arithmetic operation. The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::BinaryPattern

BOOL **BinaryPattern**(int *bitNumber*, SapData *newValue*);

### Parameters

<i>bitNumber</i>	Bit number that identifies the indices of the LUT data elements to modify
<i>newValue</i>	Source value object

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies some LUT entries based on a binary pattern. Only the entries with indices that have the *bitNumber* bit set are modified using *newValue*. Each entry is calculated as follows:

$$\text{data}[\text{index}] = (\text{index} \& (1 << \text{bitNumber})) ? \text{newValue} : \text{data}[\text{index}]$$

The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

---

## SapLut::Boolean

BOOL Boolean(SapLut::BooleanOp *operation*, SapData *value*);

### Parameters

*operation* Specifies how to modify LUT data elements. The following operations are available:

SapLut::And      Boolean AND:  $\text{data}[\text{index}] = \text{data}[\text{index}] \& \text{value}$

SapLut::Or      Boolean OR:  $\text{data}[\text{index}] = \text{data}[\text{index}] | \text{value}$

SapLut::Xor      Boolean XOR:  $\text{data}[\text{index}] = \text{data}[\text{index}] ^ \text{value}$

*value* Source value object

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies all LUT entries using a Boolean operation. The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Copy

BOOL Copy(SapLut\* *pSrc*);

### Parameters

*pSrc* LUT object to copy from

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Copies all source LUT object entries to the current object. The two LUTs must be exactly the same size, as returned by the GetTotalSize method.

### Demo/Example Usage

Not available

---

## SapLut::Create

BOOL Create();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Creates all the low-level Sapera resources needed by the LUT object.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Destroy

BOOL Destroy();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Destroys all the low-level Sapera resources needed by the LUT object.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Gamma

BOOL Gamma(float *factor*);

### Parameters

*factor*      Gamma correction factor to apply

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Modifies all LUT entries using inverse gamma correction with the specified *factor*. This is used to correct the light response of the camera, which is often a power function (referred to as the gamma function). A *factor* of 1 means no correction is applied, and a normal LUT is computed instead.

#### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

### SapLut::GetElementSize

**int GetElementSize();**

#### Remarks

Gets the number of bytes required to store a single LUT element.

The initial value for this attribute is 1. It is then set to the LUT element size value when calling the Create method.

#### Demo/Example Usage

Not available

---

### SapLut::GetFormat, SapLut::SetFormat

**SapFormat GetFormat();**

**void SetFormat(SapFormat *format*);**

#### Remarks

Gets/sets the LUT format. The initial value for this attribute is SapFormatMono8, unless a specific value was specified in the constructor.

You can only call SetFormat before the Create method. See the SapLut::SapLut constructor for possible values for *format*.

#### Demo/Example Usage

Not available

---

### SapLut::GetHandle

**CORHANDLE GetHandle();**

#### Remarks

Gets the low-level Sapera handle of the LUT resource that you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

#### Demo/Example Usage

Not available

---

### SapLut::GetLocation, SapLut::SetLocation

**SapLocation GetLocation();**

**BOOL SetLocation(SapLocation *location*);**

#### Remarks

Gets/sets the location where the LUT resource is located. This usually corresponds to the system server. A specific server can also be specified through the SapLut constructor.

#### Demo/Example Usage

Not available

---

### SapLut::GetNumEntries, SapLut::SetNumEntries

**int GetNumEntries();**

**BOOL SetNumEntries (int *numEntries*);**

#### Remarks

Gets/sets the number of LUT entries. The initial value for this attribute is 256, unless a specific value was specified in the constructor.

You can only call SetNumEntries before the Create method.

#### Demo/Example Usage

Not available

---

#### SapLut::GetNumPages

```
int GetNumPages();
```

#### Remarks

Gets the number of color planes in the LUT. The initial value for this attribute is 1. It is then set to the LUT number of pages value when calling the Create method.

This value is usually 1 if the LUT format is monochrome and 3 if it is color.

#### Demo/Example Usage

Not available

---

#### SapLut::GetParameter, SapLut::SetParameter

```
BOOL GetParameter(int param, void* pValue);  
BOOL SetParameter(int param, int value);  
BOOL SetParameter(int param, void* pValue);
```

#### Parameters

*param* Low-level Sapera C library parameter to read or write

*pValue* Pointer to parameter value to read back or to write

*value* New parameter value to write

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the LUT module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapLut Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Not available

---

#### SapLut::GetTotalSize

```
int GetTotalSize();
```

#### Remarks

Gets the number total of bytes required to store the LUT data.

The initial value for this attribute is 256. It is then set to the LUT size value when calling the Create method.

#### Demo/Example Usage

Not available

---

#### SapLut::IsSigned

```
BOOL IsSigned();
```

#### Remarks

Checks if the LUT contains signed or unsigned data.

The initial value for this attribute is FALSE. It is then set to the LUT signed value when calling the Create

method.

### Demo/Example Usage

Not available

---

## SapLut::Load

BOOL **Load**(const char\* *filename*);

### Parameters

*filename* Name of source file

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Loads LUT entries from a file. The number of entries and formats of the LUT are updated to reflect the file contents. After calling Load, use the GetNumEntries and GetFormat methods to get their updated values.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Normal

BOOL **Normal**();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies all LUT entries using a linear mapping with a positive slope, as follows:

```
data[0] = minValue  
(Linear mapping from data[0] to data[maxIndex])  
data[maxIndex] = maxValue
```

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Read

BOOL **Read**(int *lutIndex*, SapData\* *pValue*);

BOOL **Read**(int *offset*, void\* *pData*, int *size*);

### Parameters

*lutIndex* Index of LUT element to read, starting at 0

*pValue* Pointer to a destination value object

*offset* Byte offset to start reading from in the LUT.

*pData* Destination memory area for LUT data

*size* Number of bytes to read

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Use the first form of Read to read a single LUT element to either a SapDataMono or SapDataRGB object. You do not have to know the exact LUT data format and how it is stored in memory.

Use the second form of Read if you want to read raw LUT data directly to a memory area allocated in the application program. In this case, you also need to use the GetFormat, GetElementSize, and GetNumPages methods to find out the LUT data organization.

### Demo/Example Usage

Not available

---

## SapLut::Reverse

BOOL **Reverse**();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies all LUT entries using a linear mapping with a negative slope, as follows:

```
data[0] = maxValue  
(Linear mapping from data[0] to data[maxIndex])  
data[maxIndex] = minValue
```

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Roll

BOOL **Roll**(int *numEntries*);

### Parameters

*numEntries*      Specifies by how many entries LUT data should be shifted

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Relocates LUT entries upwards or downwards as one block. The actual data elements are not modified, and their position relative to one another remains the same. If *numEntries* is positive, then a downward shift occurs. If it is negative, an upward shift occurs. This behavior is expressed as follows:

```
If numEntries > 0: data[(index + numEntries) % maxIndex] = data[index]  
If numEntries < 0: data[index] = data[(index - numEntries) % maxIndex]
```

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Save

BOOL **Save**(const char\* *filename*);

### Parameters

*filename*      Name of destination file

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Saves LUT entries to a file.

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Shift

BOOL **Shift**(int *numBits*);

### Parameters

*numBits*      Specifies by how many bits LUT entries should be shifted

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Modifies all LUT entries using a logical shift. If *numBits* is positive, a left shift occurs, and the least significant bits are filled with 0's. If *numBits* is negative, a right shift occurs, and the most significant bits are filled with 0's. This behavior is expressed as follows:

```
If numBits > 0: data[index] <<= numBits  
If numBits < 0: data[index] >>= (-numBits)
```

### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Slope

```
BOOL Slope(int startIndex, int endIndex, SapData minValue, SapData maxValue,  
BOOL clipOutside = FALSE);
```

#### Parameters

<i>startIndex</i>	Starting LUT index for linear mapping
<i>endIndex</i>	Ending LUT index for linear mapping
<i>minValue</i>	LUT element value at starting index
<i>maxValue</i>	LUT element value at ending index
<i>clipOutside</i>	Specifies whether LUT elements outside the mapping range should also be modified

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Modifies part of a LUT with a linear mapping. LUT elements from *startIndex* to *endIndex* are remapped from *minValue* to *maxValue*. If *clipOutside* is FALSE, then elements outside the range are unaffected. If TRUE, then elements below *startIndex* are set to *minValue* and elements above *endIndex* are set to *maxValue*. This behavior is expressed as follows:

```
If clipOutside is TRUE: data[0] ... data[startIndex - 1] = minValue  
data[startIndex] = minValue  
(Linear mapping from data[startIndex] to data[endIndex])  
data[endIndex] = maxValue  
If clipOutside is TRUE: data[endIndex + 1] ... data[maxIndex - 1] = maxValue
```

The value arguments must be either SapDataMono or SapDataRGB objects, depending on the LUT format.

#### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Threshold

```
BOOL Threshold(SapData threshValue);  
BOOL Threshold(SapData lowValue, SapData highValue);
```

#### Parameters

<i>threshValue</i>	Reference value for single threshold
<i>lowValue</i>	Lower reference value for double threshold
<i>highValue</i>	Upper reference value for double threshold

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Modifies all LUT elements using a threshold operation.

The first form of Threshold implements single threshold. Elements with a value lower than *threshValue* are set to the lowest possible value. Elements with a value higher than or equal to *threshValue* are set to the highest possible value. This behavior is expressed as follows:

```
data[index] = (index >= threshValue) ? maxValue : minValue
```

The second form implements double threshold. Elements with a value higher than or equal to *lowValue*, but lower than *highValue*, are set to the highest possible value. Elements outside that range are set to the lowest possible value. This behavior is expressed as follows:

```
data[index] = (index >= lowValue && index < highValue) ? maxValue : minValue
```

The value arguments must be either SapDataMono or SapDataRGB objects, depending on the LUT format.

#### Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

---

## SapLut::Write

```
BOOL Write(int lutIndex, SapData value);  
BOOL Write(int offset, void* pData, int size);
```

#### Parameters

<i>lutIndex</i>	Index of LUT element to write, starting at 0
-----------------	--

<i>value</i>	Source value object
<i>offset</i>	Byte offset to start writing to in the LUT.
<i>pData</i>	Source memory area for LUT data
<i>size</i>	Number of bytes to write

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

#### **Remarks**

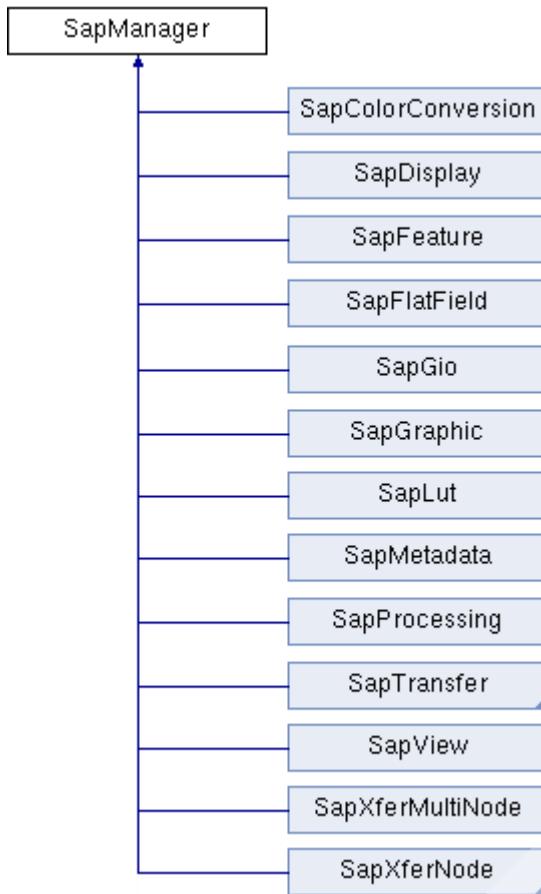
Use the first form of Write to write a single LUT element from either a SapDataMono or SapDataRGB object. You do not have to know how the LUT is stored in memory.

Use the second form of Write if you want to write raw LUT data directly from a memory area allocated in the application program. In this case, you also need to use the GetFormat, GetElementSize, and GetNumPages methods to find out the LUT data organization.

#### **Demo/Example Usage**

Not available

# SapManager



The SapManager Class includes methods for describing the Saperia resources present on the system. It also includes error management capabilities.

You will never need to explicitly create a SapManager object. First, almost all methods are declared as static, which means you may use them at any time. Second, most Saperia LT ++ classes are derived from SapManager, so they inherit its methods and protected data members.

```
#include <SapClassBasic.h>
```

## SapManager Class Members

### Attributes

<u>operator BOOL</u>	Checks whether the Create method has succeeded for a derived object
<u>GetDisplayStatusMode</u>	Gets/sets the global reporting mode for messages and errors
<u>SetDisplayStatusMode</u>	
<u>GetCommandTimeout</u>	Gets/sets the timeout value used when waiting for completion of Saperia LT commands
<u>SetCommandTimeout</u>	
<u>GetResetTimeout</u>	Gets/sets the timeout value used when resetting a hardware device
<u>SetResetTimeout</u>	
<u>GetPixelDepthMin</u>	Gets the minimum and maximum number of significant bits for a given data format
<u>GetPixelDepthMax</u>	
<u>GetServerEventType</u>	Gets the currently registered event type for server related events

### Operations

<u>Open</u>	Initializes access to the Saperia LT libraries
<u>Close</u>	Terminates access to the Saperia LT libraries

<a href="#"><u>DetectAllServers</u></a>	Detects cameras (all types or GenCP only) after a Sapera application has been started
<a href="#"><u>GetVersionInfo</u></a>	Gets Sapera LT version and licensing information
<a href="#"><u>GetServerCount</u></a>	Gets the number of available Sapera servers
<a href="#"><u>GetServerIndex</u></a>	Gets the index of a Sapera server
<a href="#"><u>GetServerName</u></a>	Gets the name of a Sapera server
<a href="#"><u>GetServerType</u></a>	Gets the type of a Sapera server
<a href="#"><u>IsServerAccessible</u></a>	Checks if the resources for a server are accessible
<a href="#"><u>GetServerHandle</u></a>	Returns the low-level Sapera handle of a server resource
<a href="#"><u>GetServerSerialNumber</u></a>	Gets the serial number corresponding to a Sapera server
<a href="#"><u>GetResourceCount</u></a>	Gets the number of Sapera resources of a specific type on a server
<a href="#"><u>GetResourceIndex</u></a>	Gets the index of a Sapera resource
<a href="#"><u>GetResourceName</u></a>	Gets the name of a Sapera resource
<a href="#"><u>IsResourceAvailable</u></a>	Checks whether a resource is available for use
<a href="#"><u>IsSystemLocation</u></a>	Check whether a SapLocation object is located on the system server
<a href="#"><u>IsSameServer</u></a>	Checks whether two SapLocation objects are located on the same server
<a href="#"><u>IsSameLocation</u></a>	Checks whether two SapLocation objects are the same
<a href="#"><u>GetFormatType</u></a>	Gets the data type corresponding to a Sapera data format
<a href="#"><u>GetStringFromFormat</u></a>	Gets a text description of a Sapera data format
<a href="#"><u>IsStatusOk</u></a>	Checks the return value of a Sapera low-level C Library function call, and reports an error if appropriate
<a href="#"><u>GetLastStatus</u></a>	Gets a description of the latest Sapera low-level C library error
<a href="#"><u>DisplayMessage</u></a>	Reports a custom message using the current reporting mode
<a href="#"><u>ResetServer</u></a>	Resets the hardware device associated with a specific server
<a href="#"><u>GetInstallDirectory</u></a>	Gets the directory where a Sapera product is installed
<a href="#"><u>RegisterServerCallback</u></a>	Registers a callback function for server related events
<a href="#"><u>UnregisterServerCallback</u></a>	Unregister the callback function for server related events
<a href="#"><u>WriteFile</u></a>	Writes a file to non-volatile memory on the device

## SapManager Member Functions

The following are members of the SapManager Class.

---

### SapManager::operator BOOL

**operator BOOL();**

#### Remarks

Checks whether the Create method has succeeded for an object derived from SapManager. This allows the variable representing the object to be used in a Boolean expression.

Calling the Destroy method resets this attribute to FALSE.

#### Demo/Example Usage

All demos and examples

---

### SapManager::Close

**static BOOL Close();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Terminates access to the Sapera LT libraries. See the SapManager::Open method for more details.

#### Demo/Example Usage

**SapManager::DetectAllServers**

```
static BOOL DetectAllServers(DetectServerType type = DetectServerAll);
```

**Parameters**

<i>type</i>	Specifies the type of server to detect. Possible values are:
DetectServerType::DetectServerGenCP	Detect GenCP servers only.
DetectServerType::DetectServerAll	Detect all server types.

**Remarks**

Use this function to detect GenCP cameras, or any other type of acquisition device (i.e., server), after a Sapera application has been started. In a typical application device detection (discovery) is initiated during application startup. If a camera is connected after an application has been launched, the device may not be detected. Use this function to trigger the device discovery process.

Note, you must register the following events to enable this method's functionality:

- EventServerConnected
- EventServerDisconnected

See SapManager::RegisterServerCallback.

**Demo/Example Usage**

Find Camera example

**SapManager::DisplayMessage**

```
static void DisplayMessage(const char* message, const char* fileName = NULL, int lineNumber = 0, ...);
```

**Parameters**

<i>message</i>	Custom message to report
<i>fileName</i>	Name of source file from which DisplayMessage is called
<i>lineNumber</i>	Line number from which DisplayMessage is called
...	Variable arguments if <i>message</i> includes printf-style format specifications

**Remarks**

Reports a custom message using the current reporting mode. File and line information is automatically appended to *message*, unless you set *fileName* to NULL.

See the SetDisplayStatusMode method for a description of the available reporting modes.

**Demo/Example Usage**

Not available

**SapManager::GetCommandTimeout, SapManager::SetCommandTimeout**

```
static int GetCommandTimeout();
static void SetCommandTimeout (int commandTimeout);
```

**Remarks**

Gets/sets the timeout value (in milliseconds) used when waiting for completion of Sapera LT commands. The initial value for this attribute is 20000 (20 seconds).

If you need to control the timeout value used by the ResetServer method, use the GetResetTimeout and SetResetTimeout methods instead.

**Demo/Example Usage**

GigE Sequential Grab Demo, Sequential Grab Demo, Camera Files Example

**SapManager::GetDisplayStatusMode, SapManager::SetDisplayStatusMode**

```
static SapManager::StatusMode GetDisplayStatusMode();
static BOOL SetDisplayStatusMode(SapManager::StatusMode mode, SapManCallback pCallback = NULL,
void* pContext = NULL);
```

**Parameters**

<i>mode</i>	New reporting mode. The following values are available:
-------------	---

SapManager::StatusNotify	Sends messages to a popup window
SapManager::StatusLog	Sends messages to the Sapera Log Server (can be displayed using the Sapera Log Viewer)
SapManager::StatusDebug	Sends messages to the active debugger, if any
SapManager::StatusCustom	Error information is not reported, it is just stored internally
SapManager::StatusCallback	Notifies application code through a callback function
<i>pCallback</i>	Application callback function to be called when reporting a message. This function must be declared as: <code>void MyCallback(SapManCallbackInfo* pInfo);</code>
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL, this parameter is ignored.

### Remarks

Gets/sets the global reporting mode for messages and errors. This mode is used by the IsStatusOk and DisplayMessage methods, and also internally by the Sapera LT ++ library.

The initial value for this attribute is StatusNotify.

For StatusCallback reporting mode, you can call one of the following SapManCallbackInfo methods from the application callback function to retrieve the relevant information: GetErrorValue, GetErrorMessage, and GetContext.

For StatusCustom reporting mode, the only way to retrieve messages is by calling the GetLastStatus method.

Note that, for all reporting modes, any Sapera LT ++ method returns FALSE to indicate an error.

### Demo/Example Usage

Camera Features Example, Find Camera Example

## SapManager::GetFormatType

static SapFormatType **GetFormatType**(SapFormat *format*);

### Parameters

*format* Sapera data format

### Remarks

Gets the data type corresponding to the specified Sapera data format as one of the following values:

SapFormatTypeUnknown	Unable to determine data type
SapFormatTypeMono	Monochrome
SapFormatTypeRGB	RGB color
SapFormatTypeYUV	YUV color
SapFormatTypeHSI	HSI color
SapFormatTypeHSV	HSV color
SapFormatTypeColor	Lookup table color data
SapFormatTypeRGBA	RGB color with an additional component (alpha channel, infrared component, etc.)

### Demo/Example Usage

Not available

## SapManager::GetInstallDirectory

static BOOL **GetInstallDirectory**(int *serverIndex*, char\* *installDir*);  
static BOOL **GetInstallDirectory** (const char\* *serverName*, char\* *installDir*);  
static BOOL **GetInstallDirectory** (SapLocation *loc*, char\* *installDir*);

### Parameters

*serverIndex* Sapera server index

*installDir* Memory area large enough to receive the installation directory (at least 257 bytes)

*serverName* Sapera server name

*loc* Valid SapLocation object

## **Return Value**

Returns TRUE if successful, FALSE otherwise

## **Remarks**

Gets the directory where a Sapera product is installed.

For the system server, this corresponds to the Sapera installation directory, for example, **c:\ Program Files\Teledyne DALSA\Sapera**.

For a server corresponding to a hardware device, this corresponds to the directory where the driver for the device is installed, for example, **c:\ Program Files\Teledyne DALSA\X64 Xcelera-CL PX4**.

## **Demo/Example Usage**

Not available

---

## **SapManager::GetLastStatus**

```
static const char* GetLastStatus();
static void GetLastStatus(SAPSTATUS* pLastStatus);
```

## **Parameters**

*pLastStatus* Pointer to the most recent Sapera low-level status code to retrieve

## **Remarks**

Gets a description of the latest Sapera LT ++ and/or low-level C library error.

The first form of GetLastStatus returns the latest text description, similar to what is generated by the IsStatusOk method. If the actual error occurred inside a call to the low-level C library, then you may also use the second form to retrieve the actual error code.

Note that each thread in a Sapera LT application has its own latest error code and description. This means that you cannot call GetLastStatus to retrieve error information for a Sapera LT ++ function which has been called in another thread.

See the SetDisplayStatusMode method for a description of the available reporting modes.

See the Sapera LT Basic Modules Reference Manual for details on low-level Sapera functionality.

## **Demo/Example Usage**

Not available

---

## **SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax**

```
static int GetPixelDepthMin(SapFormat format);
static int GetPixelDepthMax(SapFormat format);
```

## **Remarks**

Gets the minimum and maximum number of significant bits for a given buffer *format*. This corresponds to the minimum and maximum pixel depth values for a corresponding SapBuffer object.

See the SapBuffer constructor for a list of possible values for *format*.

## **Demo/Example Usage**

Dual Acquisition Demo, Grab LUT Example

---

## **SapManager.GetPixelDepthMin, SapManager.GetPixelDepthMax Method**

### **SapManager::GetResetTimeout, SapManager::SetResetTimeout**

```
static int GetResetTimeout();
static void SetResetTimeout (int timeOut);
```

## **Remarks**

Gets/sets the timeout value (in milliseconds) used when resetting a hardware device. This value is used by the ResetServer method.

If you need to get/set the timeout value used when waiting for completion of Sapera LT commands, use the GetCommandTimeout and SetCommandTimeout methods instead.

The initial value for this attribute is 20000 (20 seconds).

## **Demo/Example Usage**

Not available

---

## SapManager::GetResourceCount

```
static int GetResourceCount(int serverIndex, SapManager::ResType resourceType);
static int GetResourceCount(const char* serverName, SapManager::ResType resourceType);
static int GetResourceCount(SapLocation loc, SapManager::ResType resourceType);
```

### Parameters

<i>serverIndex</i>	Sapera server index
<i>resourceType</i>	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.
<i>serverName</i>	Sapera server name
<i>loc</i>	Valid SapLocation object

### Remarks

Gets the number of resources of a specified type on a Sapera server. This only applies to static resources, that is, those attached to physical devices. Dynamic resources, like buffers, do not have a fixed count.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

### Demo/Example Usage

IO Demo, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

---

## SapManager::GetResourceIndex

```
static int GetResourceIndex(int serverIndex, SapManager::ResType resourceType, const char*
resourceName);
static int GetResourceIndex(const char *serverName, SapManager::ResType resourceType,
const char* resourceName);
```

### Parameters

<i>serverIndex</i>	Sapera server index
<i>resourceType</i>	Resource type to inquire. See the GetServerCount method for the list of possible values.
<i>resourceName</i>	Sapera resource name
<i>serverName</i>	Sapera server name

### Remarks

Gets the index of a Sapera resource. Returns SapLocation::ResourceUnknown if the specified resource cannot be found.

The first form of GetResourceIndex looks for the resource of the specified name and type on the server specified by *index*. The second form uses the server name instead of the index.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server and resource names for that product.

### Demo/Example Usage

Not available

---

## SapManager::GetResourceName

```
static BOOL GetResourceName(int serverIndex, SapManager::ResType resourceType, int resourceId,
char* resourceName);
Static BOOL GetResourceName(const char *serverName, SapManager::ResType resourceType,
int resourceId, char* resourceName, int nameSize = MaxLabelSize);
static BOOL GetResourceName(SapLocation loc, SapManager::ResType resourceType, char* resourceName);
```

### Parameters

<i>serverIndex</i>	Index of Sapera server containing the resource.
<i>resourceType</i>	Resource type to inquire. See the GetServerCount method for the list of possible values.

<i>resourceIndex</i>	Index of requested resource of the specified type.
<i>resourceName</i>	Memory area large enough to receive the resource name (at least 128 bytes).
<i>serverName</i>	Name of Sapera server containing the resource.
<i>loc</i>	Valid SapLocation object.
<i>nameSize</i>	Size of memory allocated by <i>resourceName</i> , in bytes.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the name of a Sapera resource of a specified type.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

#### Demo/Example Usage

Not available

### SapManager::GetServerCount

```
static int GetServerCount();
static int GetServerCount(SapManager::ResType resourceType);
```

#### Parameters

<i>resourceType</i>	Resource type to inquire, can be one of the following:
SapManager::ResourceAcq	Frame grabber acquisition hardware
SapManager::ResourceAcqDevice	Camera acquisition hardware (for example, Genie)
SapManager::ResourceCounter	Event counters
SapManager::ResourceDisplay	Physical displays
SapManager::ResourceDsp	Digital Signal Processors
SapManager::ResourceGio	General inputs and outputs
SapManager::ResourceGraphic	Graphics engine
SapManager::ResourceRtPro	Realtime Processor hardware

#### Remarks

Gets the number of available Sapera servers.

The first form of this method considers all servers, regardless of their resource type. In this case, the return value is at least 1, since the system server is always present. The second form returns the number of servers for the specified resource type only, so the return value may be equal to 0.

Note that the following resource types apply only to older products: ResourceCab, ResourceCounterResourceDsp, and ResourcePixPro. See the *Sapera LT ++ Legacy Classes Reference Manual* for related classes.

#### Demo/Example Usage

IO Demo, Find Camera Example

### SapManager::GetServerEventType

```
static SapManager::EventType GetServerEventType();
```

#### Remarks

Gets the currently registered event type for server related events. See the RegisterServerCallback method for the list of possible values.

If this method returns the special value SapManager::EventNone, this means that no server related events are currently registered. This is the default value, which is also reset when calling the UnregisterServerCallback method.

## Demo/Example Usage

Not available

---

### SapManager::GetServerHandle

```
static BOOL GetServerHandle(int serverIndex, PCORSERVER pServer);
static BOOL GetServerHandle(const char* serverName, PCORSERVER pServer);
static BOOL GetServerHandle(SapLocation loc, PCORSERVER pServer);
```

#### Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
pServer	Pointer to returned low-level Sapera server handle

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the low-level handle of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

## Demo/Example Usage

Not available

---

### SapManager::GetServerIndex

```
static int GetServerIndex(const char* serverName);
static int GetServerIndex(SapLocation loc);
```

#### Parameters

serverName	Sapera server name
loc	Valid SapLocation object

#### Remarks

Gets the index of a Sapera server. Returns SapLocation::ServerUnknown if the specified server cannot be found.

The first form of this method uses the server name. The second form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

## Demo/Example Usage

IO Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

---

### SapManager::GetServerName

```
static BOOL GetServerName(int serverIndex, char* serverName);
static BOOL GetServerName(SapLocation loc, char* serverName);
static BOOL GetServerName(int serverIndex, SapManager::ResType resourceType, char* serverName);
```

#### Parameters

serverIndex	Sapera server index
serverName	Memory area large enough to receive the server name (at least 32 bytes)
loc	Valid SapLocation object

**resourceType** Resource type to inquire. See the GetServerCount method for the list of possible values.

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets the name of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses an existing SapLocation object with valid server information. The third form only considers servers with at least one resource of the specified type. For example, index 1 corresponds to the second server with at least one acquisition device.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

#### Demo/Example Usage

Dual Acquisition Demo, IO Demo, Find Camera Example

---

### SapManager::GetServerSerialNumber

```
static BOOL GetServerSerialNumber(int serverIndex, char* serialNumber);
static BOOL GetServerSerialNumber(const char* serverName, char* serialNumber);
static BOOL GetServerSerialNumber(SapLocation loc, char* serialNumber);
```

#### Parameters

*serverIndex* Sapera server index

*serialNumber* Memory area large enough to receive the text for the serial number (at least 16 bytes)

*serverName* Sapera server name

*loc* Valid SapLocation object

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Gets a text representation of the serial number corresponding to the hardware device for the specified Sapera server. It consists of either the letter 'S' or 'H' followed by seven digits, for example, "S1234567".

Note that there is no serial number associated with the System server. Also, this function is only supported for frame grabbers and older Genie cameras (not Genie-TS). When using other camera servers (GigE-Vision or GenCP), you need a valid SapAcqDevice object from which the serial number can be retrieved through a named feature.

#### Demo/Example Usage

Not available

---

### SapManager::GetServerType

```
static SapManager::ServerType GetServerType(int serverIndex);
static SapManager::ServerType GetServerType(const char* serverName);
static SapManager::ServerType GetServerType(SapLocation loc);
```

#### Parameters

*serverIndex* Sapera server index

*serverName* Sapera server name

*loc* Valid SapLocation object

#### Return Value

Can be one of the following:

SapManager::ServerNone	Server type cannot be determined
------------------------	----------------------------------

SapManager::ServerSystem	System server
--------------------------	---------------

SapManager::ServerBandit3CV	Bandit-3 CV Express VGA frame grabber
-----------------------------	---------------------------------------

SapManager::ServerX64CL	X64-CL acquisition board
-------------------------	--------------------------

SapManager::ServerX64CLiPRO	X64-CL iPro acquisition board
-----------------------------	-------------------------------

SapManager::ServerX64CLExpress	X64-CL-Express acquisition board
--------------------------------	----------------------------------

SapManager::ServerX64CLLX4	X64 Xcelera-CL LX4 acquisition board
----------------------------	--------------------------------------

SapManager::ServerX64CLPX4	X64 Xcelera-CL PX4 acquisition board
SapManager::ServerX64LVDS	X64-LVDS acquisition board
SapManager::ServerX64LVDSPX4	X64-LVDSPX4 acquisition board
SapManager::ServerX64LVDSVX4	X64-LVDSVX4 acquisition board
SapManager::ServerX64ANQuad	X64-AN Quad acquisition board
SapManager::ServerX64ANLX1	X64-ANLX1 acquisition board
SapManager::ServerPC2Vision	PC2-Vision acquisition board
SapManager::ServerPC2Comp	PC2-Comp Express acquisition board
SapManager::ServerPC2CamLink	PC2-CamLink acquisition board
SapManager::ServerGenie	Genie camera
SapManager::ServerAnacondaCL	Anaconda-CL vision processor
SapManager::ServerAnacondaLVDS	Anaconda-LVDS vision processor

### Remarks

Gets the type of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

### Demo/Example Usage

Not available

## SapManager::GetStringFromFormat

```
static BOOL GetStringFromFormat(SapFormat format, char* txtFormat);
```

### Parameters

*format* Sapera data format

*txtFormat* Memory area large enough to receive the description (at least 16 bytes)

### Return Value

Returns TRUE if the low-level function call succeeded, FALSE otherwise

### Remarks

Gets a text description of the specified Sapera data format, for example, 'MONO8' for SapFormatMono8.

### Demo/Example Usage

Not available

## SapManager::GetVersionInfo

```
BOOL GetVersionInfo(SapManVersionInfo* pVersionInfo);
```

### Parameters

*pVersionInfo* Pointer to a SapManVersionInfo object

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Gets Sapera LT version and licensing information. When the method call returns, the SapManVersionInfo object pointed to by *pVersionInfo* contains the information. It has the following attributes:

int GetMajor()	Gets the major version number. For example, if the version number is 6.30.01.0806, this method returns 6.
int GetMinor()	Gets the minor version number. For example, if the version number is 6.30.01.0806, this method returns 30.
int GetRevision()	Gets the revision number. For example, if the version number is 6.30.01.0806, this method returns 1.

<code>int GetBuild()</code>	Gets the build number. For example, if the version number is 6.30.01.0806, this method returns 806.
<code>SapManVersionInfo::LicenseType GetLicenseType()</code>	Gets the Sapera LT license type for the current installation, which can be one of SapManVersionInfo::Runtime, SapManVersionInfo::Evaluation, or SapManVersionInfo::FullSDK
<code>int GetEvalDaysRemaining()</code>	Gets the number of days remaining in the evaluation period when the license type is Runtime, where a value equal to 0 means that the evaluation period has expired.

#### Demo/Example Usage

Not available

### SapManager::IsResourceAvailable

```
static BOOL IsResourceAvailable(int serverIndex, SapManager::ResType resourceType, int resourceIndex);
static BOOL IsResourceAvailable(const char* serverName, SapManager::ResType resourceType,
int resourceIndex);
static BOOL IsResourceAvailable (SapLocation loc, SapManager::ResType resourceType);
```

#### Parameters

<code>serverIndex</code>	Index of Sapera server containing the resource
<code>resourceType</code>	Resource type to inquire. See the GetServerCount method for the list of possible values.
<code>resourceIndex</code>	Index of requested resource of the specified type
<code>serverName</code>	Name of Sapera server containing the resource
<code>loc</code>	Valid SapLocation object

#### Return Value

Returns TRUE if the specified resource is not already used, FALSE otherwise

#### Remarks

Determines if a specific Sapera resource on a server is available. You may use this method, for example, before calling SapAcquisition::Create to avoid getting an error when the acquisition resource is already in use.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

#### Demo/Example Usage

Not available

### SapManager::IsSameLocation Method

```
static BOOL IsSameLocation(SapLocation loc1, SapLocation loc2);
```

#### Parameters

<code>loc1</code>	First valid SapLocation object
<code>loc2</code>	Second valid SapLocation object

#### Remarks

Checks if the two specified SapLocation objects have the same server and resource information.

#### Demo/Example Usage

Not available

### SapManager::IsSameServer

```
static BOOL IsSameServer(SapLocation loc1, SapLocation loc2);
```

#### Parameters

<code>loc1</code>	First valid SapLocation object
<code>loc2</code>	Second valid SapLocation object

## Remarks

Checks if the two specified SapLocation objects have the same server information.

## Demo/Example Usage

IsSameServer

---

## SapManager::IsServerAccessible

```
static BOOL IsServerAccessible(int serverIndex);  
static BOOL IsServerAccessible(const char* serverName);  
static BOOL IsServerAccessible(SapLocation loc);
```

### Parameters

*serverIndex* Index of Sapera server containing the resource  
*serverName* Name of Sapera server containing the resource  
*Loc* Valid SapLocation object

### Return Value

Returns TRUE if the resources for the server are accessible, FALSE otherwise.

## Remarks

Checks if the resources belonging to a server are currently accessible. Although existing objects for these resources are still valid when their server becomes unaccessible, they must be left alone or destroyed (for example, SapAcqDevice::Destroy).

When a Sapera application starts, all detected servers are automatically accessible. However, Sapera camera devices (GigE-Vision and GenCP) can be connected and disconnected while a Sapera application is running. When such a device is connected for the first time, its server is automatically accessible. When the device is later disconnected, the server becomes unaccessible. If it is reconnected again, the server is once again accessible.

The first form of this method uses a server index. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object.

Accessibility of servers can also be determined by registering callbacks for server related events using the RegisterServerCallback method.

Note that you should not use this method for devices which are always connected (for example, frame grabbers), since the return value may not correspond to the actual resource accessibility for the corresponding server.

## Demo/Example Usage

Not available

---

## SapManager::IsSystemLocation

```
static BOOL IsSystemLocation();  
static BOOL IsSystemLocation(SapLocation loc);
```

### Parameters

*loc* Valid SapLocation object

## Remarks

Check if the current application is running on the system server, or if the SapLocation object refers to this server.

## Demo/Example Usage

Not available

---

## SapManager::IsStatusOk

```
static BOOL IsStatusOk(const char* functionName, SAPSTATUS status);
```

### Parameters

*functionName* Name of a low-level Sapera function  
*status* Low-level status code returned by the function

### Return Value

Returns TRUE if the low-level function call succeeded, FALSE otherwise

### Remarks

Checks the return value of a Sapera low-level C library function call and reports an error if appropriate, using the current reporting mode. See the SetDisplayStatusMode method for a description of the available modes.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

### Demo/Example Usage

Not available

---

## SapManager::Open

static BOOL Open();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Initiates access to the Sapera LT libraries.

For most applications you do not have to call this method, as the libraries are automatically loaded during the first Sapera LT ++ constructor call in the application code, and automatically unloaded during the last Sapera LT ++ destructor call. For example:

```
// No Sapera LT ++ calls before this line
// This loads the libraries
SapBuffer *buf1 = new SapBuffer();
SapBuffer *buf2 = new SapBuffer();
delete buf1;
// This unloads the libraries
delete buf2;
// No Sapera LT ++ calls after this line
```

There is, however, at least one case for which the default behavior may not be acceptable. If the application code frequently deletes all Sapera LT ++ objects and then reallocates them again, the libraries will be unloaded and reloaded each time, causing a noticeable delay. In this case, you can call the Open method as the first Sapera LT ++ call in the application, with a call to SapManager::Close as the last Sapera LT ++ call. This results in the libraries being loaded and unloaded exactly once, as follows:

```
// No Sapera LT ++ calls before this line
// This loads the libraries
SapManager::Open();
//
// Arbitrary Sapera LT ++ calls, none of which unloads or reloads the libraries
//
// This unloads the libraries
SapManager::Close();
// No Sapera LT ++ calls after this line
```

### Demo/Example Usage

Find Camera Example

---

## SapManager::RegisterServerCallback

static BOOL RegisterServerCallback(SapManager::EventType *eventType*, SapManCallback *callback*, void\* *context* = NULL);

### Parameters

<i>eventType</i>	Manager events for which the application callback function will be called. One or more of the following values may be combined together using a bitwise OR operation:
SapManager::EventServerNew	A new device is connected while a Sapera application is already running
SapManager::EventServerDisconnected	The device corresponding to an existing server is disconnected. (Replaces SapManager::EventServerNotAccessible which is now deprecated.)
SapManager::EventServerConnected	The device corresponding to an existing, unaccessible server is reconnected. (Replaces

	SapManager::EventServerDataBaseFull	SapManager::EventServerNotAccessible, which is now deprecated.)
	SapManager::EventResourceInfoChanged	There is no room in the Sapera server database for a new device that has just been connected
	SapManager::EventServerFile	The information describing a resource (typically its label) has changed
		The information about the progress of the file being transferred
<i>callback</i>	Application callback function to be called each time one of the events specified above is received.	
		The callback function must be declared as: void MyCallback(SapManCallbackInfo *pInfo);
<i>context</i>	Optional pointer to an application context to be passed to the callback function.	

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Registers a callback function for server related events. The callback function provides information on the corresponding event (through the SapManCallbackInfo object). The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

In the callback function, you can obtain the event type that triggered the callback by calling the SapManCallbackInfo::GetEventType method. For all events except EventServerDataBaseFull, you can obtain the index of the server by calling SapManCallbackInfo::GetServerIndex. For the EventResourceInfoChanged event, you can obtain the index of the affected resource by calling SapManCallbackInfo::GetResourceIndex.

The EventResourceInfoChanged event can only occur as a result of modifying the value of the 'DeviceUserID' feature through the SapAcqDevice class.

The EventServerFile event will occur only when a firmware file is being uploaded to a device like a frame-grabber. You can obtain the progress of the upload by calling SapManCallbackInfo::GetFilePercentProgress.

Note that server related events are only available when dealing with Sapera camera devices (GigE-Vision and GenCP), that can be connected and disconnected while a Sapera application is running.

### Demo/Example Usage

Not available

## SapManager::ResetServer

```
static BOOL ResetServer(int serverIndex, BOOL wait = TRUE, SapManCallback pCallback = NULL,
void* pContext = NULL);
static BOOL ResetServer(const char* serverName, BOOL wait = TRUE, SapManCallback pCallback = NULL,
void* pContext = NULL);
static BOOL ResetServer(SapLocation loc, BOOL wait = TRUE, SapManCallback pCallback = NULL,
void* pContext = NULL);
```

### Parameters

<i>serverIndex</i>	Sapera server index
<i>serverName</i>	Sapera server name
<i>loc</i>	Valid SapLocation object
<i>wait</i>	Specifies whether this method should return immediately after resetting the specified server, or if it should wait for the server to be operational again
<i>pCallback</i>	Application callback function to be called when the server is operational again after a reset. The callback function must be declared as: void MyCallback(SapManCallbackInfo* pInfo);
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL, this parameter is ignored.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Resets the hardware device associated with a specific server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

There are three ways to use this method:

<code>wait = TRUE</code>	Returns only when the reset is complete, and the server is operational again
<code>pCallback = don't care</code>	
<code>wait = FALSE</code>	Returns immediately after resetting the server. The application program is then responsible for figuring out when the server is operational again.
<code>pCallback = NULL</code>	
<code>wait = FALSE</code>	Returns immediately after resetting the server, and notifies the application using the callback function when the server is operational again
<code>pCallback != NULL</code>	

You can call the GetServerIndex and GetContext methods of the SapManCallbackInfo class to retrieve the required information from the application callback function.

Note that all other Sapera LT ++ objects must be destroyed before calling this method, otherwise application behavior is undefined. To reset the server, use the following sequence:

- Call Destroy on all Sapera LT ++ objects (SapTransfer, SapBuffer, SapAcquisition, ...)
- Call ResetServer
- Call Create for all needed Sapera LT ++ objects

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

### Demo/Example Usage

Not available

---

## SapManager::UnregisterServerCallback

static BOOL **UnregisterServerCallback**(void);

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Unregisters the callback function for server related events.

### Demo/Example Usage

Not available

---

## SapManager::WriteFile

static BOOL **WriteFile**(int serverIndex, const char \*localFilePath, int deviceFileName);  
static BOOL **WriteFile**(const char\* serverName, const char \*localFilePath, int deviceFileName);  
static BOOL **WriteFile**(SapLocation loc, const char \*localFilePath, int deviceFileName);

### Parameters

<code>serverIndex</code>	Sapera server index
<code>serverName</code>	Sapera server name
<code>loc</code>	Valid SapLocation object
<code>deviceFileName</code>	Name of the device file. See the acquisition device User's Manual for the list of supported files.
<code>deviceFileIndex</code>	Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
<code>localFilePath</code>	Full directory path and filename on the host computer to save the file.

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Writes a file to non-volatile memory on the device. Refer to the acquisition device's User's Manual for the list of supported files.

### Demo/Example Usage

Camera Files Example

# SapManCallbackInfo

The SapManCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapManager class.

```
#include <SapClassBasic.h>
```

## SapManCallbackInfo Class Members

### Construction

SapManCallbackInfo      Class constructor

### Attributes

<u>GetEventType</u>	Gets the manager events that triggered the call to the application callback
<u>GetServerIndex</u>	Gets the Sapera server index associated with the call to the application callback
<u>GetResourceIndex</u>	Gets the Sapera resource index associated with the call to the application callback
<u>GetContext</u>	Gets the application context associated with the callback
<u>GetErrorValue</u>	Gets the low-level Sapera C library error code associated with the callback
<u>GetErrorMessage</u>	Gets the error message associated with the callback
<u>GetFilePercentProgress</u>	Gets the file transfer progress when writing a file to a device.

## SapManCallbackInfo Member Functions

The following are members of the SapManCallbackInfo Class.

### SapManCallbackInfo::SapManCallbackInfo

#### **SapManCallbackInfo(**

```
int serverIndex,  
void* context);
```

#### **SapManCallbackInfo(**

```
SapManager::EventType eventType,  
int serverIndex,  
void* context);
```

#### **SapManCallbackInfo(**

```
SapManager::EventType eventType,  
int serverIndex,  
int resourceIndex,  
void* context);
```

#### **SapManCallbackInfo(**

```
SAPSTATUS errorValue,  
const char* errorMessage,  
void* context);
```

#### **SapManCallbackInfo(**

```
SapManager::EventType eventType,  
int serverIndex,  
void *context,  
int filePercentProgress)
```

### Parameters

<i>eventType</i>	Combination of manager events. See the SapManager::RegisterServerCallback method for the list of possible values.
<i>serverIndex</i>	Sapera server index
<i>resourceIndex</i>	Sapera resource index
<i>context</i>	Pointer to the application context
<i>errorValue</i>	Low-level Sapera C library error code
<i>errorMessage</i>	Error message as a text string
<i>filePercentProgress</i>	File transfer progress

## **Remarks**

SapManager objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapManager uses this class in various situations. The first corresponds to the case when a server is operational again after being reset. The second case corresponds to the callback reporting mode which is set by calling the SapManager::SetDisplayStatusMode method.

The third case corresponds to other server related events, for example, when an acquisition device is physically disconnected. This involves explicitly registering a callback function using the SapManager::RegisterServerCallback method.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetContext**

```
void* GetContext();
```

## **Remarks**

Gets the application context associated with the call to the application callback. See the SapManager::SetDisplayStatusMode method for more details.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetErrorMessage**

```
const char* GetErrorMessage();
```

## **Remarks**

Gets the error message associated with the call to the application callback as a text string. See the SapManager::SetDisplayStatusMode method for more details.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetErrorValue**

```
SAPSTATUS GetErrorValue();
```

## **Remarks**

Gets the low-level Sapera C library error code associated with the call to the application callback. See the SapManager::SetDisplayStatusMode method for more details.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetEventType**

```
SapManager::EventType GetEventType();
```

## **Remarks**

Gets the combination of manager events that triggered the call to the application callback. See the SapManager::RegisterServerCallback method for more details.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetFilePercentProgress**

```
int GetFilePercentProgress() const
```

## **Remarks**

Gets the file transfer progress, as a percentage of the file size, when writing a file to a device.

## **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetResourceIndex**

```
int GetResourceIndex();
```

#### **Remarks**

Gets the Sapera resource index associated with the call to the application callback. See the SapManager::RegisterServerCallback method for more details.

#### **Demo/Example Usage**

Not available

---

### **SapManCallbackInfo::GetServerIndex**

```
int GetServerIndex();
```

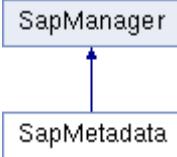
#### **Remarks**

Gets the Sapera server index associated with the call to the application callback. See the SapManager::ResetServer and SapManager::RegisterServerCallback methods for more details.

#### **Demo/Example Usage**

Not available

# SapMetadata



The SapMetadata Class provides functions to manage GigE-Vision camera metadata (for example, Genie-TS and Linea GigE). When enabled, supported metadata (for example, the timestamp or device ID) is available in the SapBuffer object.



The SapMetadata::Create must be called **after** construction of the SapAcqDevice object and **before** the construction of the SapBuffer object.  
Note: the metadata information is available in the SapBuffer object; it is not saved with the image.

To use metadata:

- After successfully calling the SapMetadata::Create function , use the Enable function to turn on metadata.
- Use GetSelectorCount to retrieve the number of available metadata items. The GetSelectorName provides the description of the metadata item.
- Use Select to add metadata items to the buffer. To determine the items that are selected (for example, in a user configuration set), use IsSelected.
- Use the Extract function to obtain the metadata from the buffer.
- Use the GetExtractedResultCount and GetExtractedResult functions to retrieve the metadata.

```
#include <SapClassBasic.h>
```

## SapMetadata Class Members

### Construction

<a href="#"><u>SapMetadata</u></a>	Class constructor.
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources.
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources.

### Operations

<a href="#"><u>IsMetadataSupported</u></a>	Returns whether metadata is supported by the acquisition device.
<a href="#"><u>GetMetadataType</u></a>	Returns the metadata type.
<a href="#"><u>Enable</u></a>	Enables metadata in the buffer.
<a href="#"><u>IsEnabled</u></a>	Returns if metadata is enabled in the buffer.
<a href="#"><u>GetSelectorCount</u></a>	Gets the metadata selector count.
<a href="#"><u>GetSelectorName</u></a>	Gets the specified selector's name.
<a href="#"><u>Select</u></a>	Selects the metadata.
<a href="#"><u>IsSelected</u></a>	Returns if the specified metadata is selected.
<a href="#"><u>Extract</u></a>	Extracts the selected metadata from the buffer.
<a href="#"><u>GetExtractedResultCount</u></a>	Gets the number of extracted metadata items.
<a href="#"><u>GetExtractedResult</u></a>	Gets the specified metadata.
<a href="#"><u>SaveToCSV</u></a>	Saves the metadata to a comma separated values (CSV) file.

## SapMetadata Member Functions

The following are members of the SapMetadata Class.

### SapMetadata::SapMetadata

**SapMetadata**(SapAcqDevice\* *pAcqDevice*, SapBuffer\* *pBuffer*);

**Parameters**

*pAcqDevice* Acquisition device object

*pBuffer* Buffer object

**Remarks**

The SapMetadata constructor does not actually create the low-level Sapera resources. To do this, you must call the SapMetadata::Create method.

**Demo/Example Usage**

GigE Metadata Demo

---

### SapMetadata::Create

BOOL **Create**();

**Return Value**

Returns TRUE if successful, FALSE otherwise.

**Remarks**

Creates all the low-level Sapera resources needed by the acquisition object. The Create function must be called after construction of the SapAcqDevice object and before the construction of the SapBuffer object.

**Demo/Example Usage**

GigE Metadata Demo

---

### SapMetadata::Destroy

BOOL **Destroy**();

**Return Value**

Returns TRUE if successful, FALSE otherwise.

**Remarks**

Destroys all the low-level Sapera resources needed by the acquisition object.

**Demo/Example Usage**

GigE Metadata Demo

---

### SapMetadata::Enable

BOOL **Enable**(BOOL *enable* = TRUE);

**Parameters**

*enable* Sets the enable state for metadata.

**Return Value**

Returns TRUE if successful, FALSE otherwise.

**Remarks**

Enables metadata for the acquisition device and buffers specified during construction of the SapMetadata object.

**Demo/Example Usage**

GigE Metadata Demo

---

### SapMetadata::Extract

BOOL **Extract**();

BOOL **Extract**(UINT *bufferIndex*);

BOOL **Extract**(UINT *bufferIndex*, UINT *lineIndex*);

**Parameters**

*bufferIndex* Buffer index. Possible values are from 0 to ([SapBuffer::GetCount](#) - 1).

*lineIndex* Line index. Possible values are from 0 to ([SapBuffer::GetHeight](#) - 1).

**Return Value**

Returns TRUE if successful, FALSE otherwise.

### Remarks

Extracts the metadata from the buffer specified during construction of the SapMetadata object.

For area scan acquisition, when the buffer count is 1, use the Extract() prototype; when the SapBuffer object contains multiple buffers, use the Extract(buflIndex) prototype.

For line scan acquisition, use the Extract(buflIndex, lineIndex) prototype.

Use the SapMetadata::GetMetadataType to verify the metadata type (per line or frame).

### Demo/Example Usage

GigE Metadata Demo

---

## SapMetadata::GetExtractedResult

BOOL **GetExtractedResult**(UINT *resultIndex*, char\* *name*, UINT *nameLength*, char\* *value*, UINT *valueLength*)  
const;

### Parameters

<i>resultIndex</i>	Result index. Possible values are from 0 to (SapMetadata::GetExtractedResultCount - 1).
<i>name</i>	Metadata item name.
<i>nameLength</i>	Size (in bytes) of the buffer pointed to by <i>name</i> .
<i>value</i>	Metadata value.
<i>valueLength</i>	Size (in bytes) of the buffer pointed to by <i>value</i> .

### Return Value

Returns TRUE if successful, FALSE otherwise.

### Remarks

Extracts the metadata for the specified result index. Use the SapMetadata::GetExtractedResultCount function to get the number of available metadata results.

### Demo/Example Usage

GigE Metadata Demo

---

## SapMetadata::GetExtractedResultCount

UINT **GetExtractedResultCount**();

### Return Value

Returns the number of metadata items for the selected metadata.

### Remarks

Returns the number of available metadata results. Use the SapMetadata::GetExtractedResult function to extract the results.

### Demo/Example Usage

GigE Metadata Demo

---

## SapMetadata::GetSelectorCount

UINT **GetSelectorCount**() const;

### Return Value

Returns the number of available metadata items supported by the acquisition device.

### Remarks

This value determines the range of the *selectorIndex* parameter used by the SapMetadata::GetSelectorName and SapMetadata::IsSelected functions.

### Demo/Example Usage

GigE Metadata Demo

---

## SapMetadata::GetSelectorName

BOOL **GetSelectorName**(UINT *selectorIndex*, char\* *name*, UINT *nameLength*) const;

### Parameters

*selectorIndex* Selector index. Possible values are from 0 to (SapMetadata::GetSelectorCount -1).

*name* Metadata item name.

*nameLength* Size (in bytes) of the buffer pointed to by *name*.

#### Return Value

Returns TRUE if successful, FALSE otherwise.

#### Remarks

Returns the name of the metadata item associated with the specified selector index. The number of metadata items (selectors) is returned by the SapMetadata::GetSelectorCount method.

#### Demo/Example Usage

GigE Metadata Demo

---

### SapMetadata::GetMetadataType

MetadataType **GetMetadataType()**;

#### Return Value

Returns the metadata type. Possible values are:

SapMetadata::MetadataPerFrame      Metadata is inserted per frame.

SapMetadata::MetadataPerLine      Metadata is inserted per line.

SapMetadata::MetadataUnknown      Metadata type is unknown.

#### Remarks

Gets the metadata type for the acquisition device specified during construction of the SapMetadata object.

#### Demo/Example Usage

GigE Metadata Demo

---

### SapMetadata::IsEnabled

BOOL **IsEnabled()**;

#### Return Value

Returns TRUE if metadata is enabled, FALSE otherwise.

#### Demo/Example Usage

GigE Metadata Demo

---

### SapMetadata::IsMetadataSupported

static BOOL **IsMetadataSupported**(SapAcqDevice\* *pAcqDevice*);

#### Parameters

*pAcqDevice* Acquisition device object

#### Return Value

Returns TRUE if metadata is supported for the specified acquisition device object, FALSE otherwise.

#### Remarks

This is a static function and can be called without creating a SapMetadata object.

#### Demo/Example Usage

GigE Metadata Demo

---

### SapMetadata::IsSelected

BOOL **IsSelected**(UINT *selectorIndex*);

#### Parameters

*selectorIndex* Index of the metadata item. Possible values are from 0 to (SapMetadata::GetSelectorCount - 1).

#### Return Value

Returns TRUE if metadata is enabled for the specified metadata item, FALSE otherwise.

#### Demo/Example Usage

---

### SapMetadata::SaveToCSV

BOOL **SaveToCSV**(const char\* *filename*);

#### Parameters

*filename* Name of CSV file to save metadata to.

#### Return Value

Returns TRUE if successful, FALSE otherwise.

#### Remarks

Extracts the metadata from the specified acquisition device and buffer.

#### Demo/Example Usage

GigE Metadata Demo

---

### SapMetadata::Select

BOOL **Select**(UINT *selectorIndex*, BOOL *select* = TRUE);

#### Parameters

*selectorIndex* Selector index. Possible values are from 0 to (SapMetadata::GetSelectorCount -1).

*select* Sets the enable state of the specified metadata item.

#### Return Value

Returns TRUE if successful, FALSE otherwise.

#### Remarks

Sets the enable state for the specified metadata item. By default, metadata items may be enabled/disabled depending on the factory/user settings loaded by the device.

#### Demo/Example Usage

GigE Metadata Demo

# SapPerformance

The SapPerformance Class implements basic benchmarking functionality. It is used by the SapProcessing Class to evaluate the time it takes to process one buffer. You may also use it for your own benchmarking needs.

```
#include <SapClassBasic.h>
```

## SapPerformance Class Members

### Construction

SapPerformance      Class constructor

### Operations

Reset                Resets the internal timer

GetTime              Gets the number of seconds elapsed since the last timer reset

GetTimeMilli          Gets the number of milliseconds elapsed since the last timer reset

GetTimeMicro         Gets the number of microseconds elapsed since the last timer reset

## SapPerformance Member Functions

The following are members of the SapPerformance Class.

---

### SapPerformance::SapPerformance

**SapPerformance();**

#### Remarks

The SapPerformance constructor initializes the internal timer and resets it.

#### Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

---

### SapPerformance::GetTime

**float GetTime(BOOL bReset);**

#### Parameters

*bReset*                Specifies whether the internal timer should be reset after it has been queried

#### Remarks

Gets the number of seconds elapsed since the last timer reset

#### Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

---

### SapPerformance::GetTimeMicro

**float GetTimeMicro(BOOL bReset);**

#### Parameters

*bReset*                Specifies whether the internal timer should be reset after it has been queried

#### Remarks

Gets the number of microseconds elapsed since the last timer reset

#### Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

---

### SapPerformance::GetTimeMilli

**float GetTimeMilli(BOOL bReset);**

#### Parameters

*bReset*                Specifies whether the internal timer should be reset after it has been queried

#### Remarks

Gets the number of milliseconds elapsed since the last timer reset

#### **Demo/Example Usage**

GigE Sequential Grab Demo, Sequential Grab Demo

---

### **SapPerformance::Reset**

`void Reset();`

#### **Remarks**

Resets the internal timer. Calling the GetTime, GetTimeMilli, or GetTimeMicro methods then returns the amount of time elapsed since the reset.

#### **Demo/Example Usage**

GigE Sequential Grab Demo, Sequential Grab Demo

# SapProcessing



The SapProcessing Class is the base class required to implement your own processing. This class cannot be used directly. Rather, derive your own processing class (for example, SapMyProcessing), override the Run method, and insert your custom processing code. You should then call the Execute method from inside your SapTransfer callback method.

The SapProcessing Class is a 'real-time processing template' that simplifies the synchronization between the transfer task and the processing task.

When the Run method is called, you may easily retrieve the index of the next buffer resource that is ready to process. You then simply have to put your custom processing code in the overridden SapProcessing.Run method.

An internal processing thread optimizes buffer processing in real-time. This allows the main application thread to execute without any concerns for the processing task.

An 'auto empty' mechanism allows synchronization between SapProcessing and SapTransfer objects in order to process buffers in real-time without missing any data.

```
#include <SapClassBasic.h>
```

## SapProcessing Class Members

### Construction

<u>SapProcessing</u>	Class constructor
<u>Create</u>	Allocates the low-level Sapera resources
<u>Destroy</u>	Releases the low-level Sapera resources
<b>Attributes</b>	
<u>GetBuffer</u>	Gets/sets the SapBuffer object with the buffer resources to process
<u>SetBuffer</u>	
<u>SetCallbackInfo</u>	Sets the application callback method to call after processing each buffer, and the associated context
<u>GetCallback</u>	Gets the current application callback method
<u>GetContext</u>	Gets the application context associated with the application callback method
<u>GetTime</u>	Gets the execution time for the most recently processed buffer
<u>GetIndex</u>	Gets the index of the current or last processed buffer
<u>IsAutoEmpty</u>	Gets/sets the 'auto-empty' mechanism
<u>SetAutoEmpty</u>	
<u>GetThreadPriority</u>	Gets/sets the execution priority of the processing thread
<u>SetThreadPriority</u>	
<b>Operations</b>	
<u>Init</u>	Initializes the processing index
<u>Execute</u>	Process the next buffer or a specific one, possibly skipping buffers in the process
<u>ExecuteNext</u>	Process the next buffer, without skipping any buffers in the process
<u>Run</u>	Method overridden in application code to implement custom processing

## SapProcessing Member Functions

The following are members of the SapProcessing Class.

### SapProcessing::SapProcessing

```
SapProcessing(SapBuffer* pBuffer, SapProCallback pCallback = NULL, void* pContext = NULL);
```

## Parameters

<i>pBuffer</i>	SapBuffer object with the buffer resources to process
<i>pCallback</i>	Application callback function to be called after each buffer has been processed. The callback function must be declared as: <code>void MyCallback(SapProCallbackInfo* pInfo);</code>
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL, this parameter is ignored.

## Remarks

The SapProcessing constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

This class cannot be instantiated directly. You must first derive a new class from it (for example, SapMyProcessing), override the Run method, and then put your custom processing code within that method.

## Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::Create

`BOOL Create();`

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Creates all the low-level Sapera resources needed by the processing object. Also initializes the processing buffer index using the current SapBuffer index. You must call SapBuffer::Create before this method.

## Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::Destroy

`BOOL Destroy();`

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Destroys all the low-level Sapera resources needed by the processing object. You must call this method before SapBuffer::Destroy.

## Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::Execute

`void Execute();`  
`void Execute(int index);`

### Parameters

*index* Index of the buffer resource to process

### Remarks

If the *index* is specified, the corresponding buffer in the SapBuffer object is processed through the internal processing thread and the Run method. Otherwise, the current buffer is processed.

If you want to process data acquired in real-time in a buffer through the SapTransfer class, simply call the Execute method within the SapTransfer callback function in the application code. This will eventually call the Run method in your derived processing class.

The SapProcessing class will then process as many frames as possible without slowing down the transfer process. This means that some buffers will be skipped if the processing task is too slow to keep up with the acquisition. If you need all frames to be processed, call the ExecuteNext method instead.

## Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::ExecuteNext

```
void ExecuteNext();
```

### Remarks

This method processes the next unprocessed buffer in the SapBuffer object through the internal processing thread and the Run method.

If you want to process data acquired in real-time into a buffer through the SapTransfer class, simply call the ExecuteNext method within the SapTransfer callback method. This will eventually call the Run method in your derived processing class.

The SapProcessing class will then process all the frames and possibly slow down the transfer process if needed. If the processing task is fast enough to keep-up with the incoming frames, ExecuteNext behaves exactly the same way as Execute. Otherwise, the transfer process must be slowed down to give the SapProcessing object the chance to process every frame.

If you want to process as many frames as possible without affecting the transfer process, use the Execute method instead.

Note that this function does not support the SapXferPair::CycleNextEmpty and SapXferPair::CycleNextWithTrash transfer cycle modes.

### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

## SapProcessing::GetBuffer, SapProcessing::SetBuffer

```
SapBuffer* GetBuffer();
BOOL SetBuffer(SapBuffer* pBuffer);
```

### Parameters

*pBuffer* SapBuffer object containing the buffer resources to process

### Remarks

Gets/sets the SapBuffer object with the buffer resources to process. You can only call SetBuffer before the Create method.

### Demo/Example Usage

Not available

---

## SapProcessing::GetCallback

```
SapProCallback GetCallback();
```

### Remarks

Gets the current application callback method. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapProcessing constructor for more details.

### Demo/Example Usage

Not available

---

## SapProcessing::GetContext

```
void* GetContext();
```

### Remarks

Gets the application context associated with the application callback method. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapProcessing constructor for more details.

### Demo/Example Usage

Not available

---

## SapProcessing::GetIndex

```
int GetIndex();
```

### Remarks

When you call GetIndex from within the Run method of your custom processing class, it returns the index of

the current buffer to process. When you call it at any other time, it returns the index of the last processed buffer.

#### Demo/Example Usage

Not available

---

### SapProcessing::GetThreadPriority, SapProcessing::SetThreadPriority

```
int GetThreadPriority();  
void SetThreadPriority(int priority);
```

#### Remarks

Gets/sets the execution priority of the processing thread. The initial value for this attribute is normal priority, unless you construct this object using an existing SapProcessing object.

For a detailed description of this setting, refer to the SetThreadPriority function in the Win32 documentation.

#### Demo/Example Usage

Not available

---

### SapProcessing::GetTime

```
float GetTime();
```

#### Remarks

Gets the execution time for the most recently processed buffer (in milliseconds). The initial value for this attribute is 0.

#### Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

### SapProcessing::Init

```
void Init();
```

#### Remarks

Initializes the processing index from the current buffer index. The Create method automatically performs this action. This ensures correct synchronization between the processing and buffer index. So you normally do not have to call Init.

However, if you use the ExecuteNext method, but do not call it for every frame, then the processing index will not be synchronized with the buffer index. In such a case you must call Init explicitly to restore synchronization.

#### Demo/Example Usage

FlatField Demo, GigE FlatField Demo

---

### SapProcessing::IsAutoEmpty, SapProcessing::SetAutoEmpty

```
BOOL IsAutoEmpty();  
void SetAutoEmpty(BOOL isAutoEmpty);
```

#### Remarks

Gets/sets the 'auto-empty' mechanism, used for synchronizing the transfer and processing tasks in the application program.

By default, the SapTransfer class automatically calls SapBuffer::SetState(SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the processing task can even process it.

In order to correctly synchronize the transfer and processing tasks, you must first disable this behavior by calling SapTransfer::SetAutoEmpty(FALSE). Then call SapProcessing::SetAutoEmpty(TRUE) to enable it in this class instead.

As a result, no images will be acquired in the current buffer as long as the Run method is executing. The buffer state is then reset before the application callback method, if any, is called.

The initial value for this attribute is FALSE, unless you construct this object using an existing SapProcessing object.

#### Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::Run

virtual BOOL **Run()** = 0;

### Remarks

This method is automatically invoked by the internal processing thread whenever a buffer is available for processing.

You first need to derive your own class from SapProcessing. Then override Run, and add your own processing code to it. You can use the GetIndex method to get the index of the buffer to process.

### Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapProcessing::SetCallbackInfo

BOOL **SetCallbackInfo**(SapProCallback *pCallback*, void\* *pContext* = NULL);

### Remarks

Sets the application callback method to call after processing each buffer, and the associated context. You can only call SetCallbackInfo before the Create method.

See the SapProcessing constructor for more details.

### Demo/Example Usage

Not available

# SapProCallbackInfo

The SapProCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapProcessing Class.

```
#include <SapClassBasic.h>
```

## SapProCallbackInfo Class Members

### Construction

[SapProCallbackInfo](#) Class constructor

### Attributes

[GetProcessing](#) Gets the SapProcessing object associated with the processing callback function

[GetContext](#) Gets the application context associated with the SapProcessing callback function

## SapProCallbackInfo Member Functions

The following are members of the SapProCallbackInfo Class.

---

### SapProCallbackInfo::SapProCallbackInfo

**SapProCallbackInfo(SapProcessing\* *pPro*, void\* *context*);**

#### Parameters

*pPro* SapProcessing object that calls the callback function

*context* Pointer to the application context

#### Remarks

SapProcessing objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapProcessing uses this class when notifying the application that a buffer has been fully processed.

#### Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

### SapProCallbackInfo::GetContext

**void\* GetContext();**

#### Remarks

Gets the context information associated with the application callback function. See the SapProcessing constructor for more details.

#### Demo/Example Usage

Not available

---

### SapProCallbackInfo::GetProcessing

**SapProcessing\* GetProcessing();**

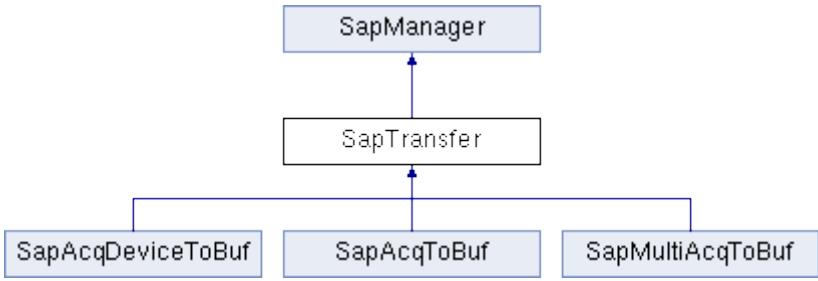
#### Remarks

Gets the SapProcessing object associated with the processing callback function. See the SapProcessing constructor for more details.

#### Demo/Example Usage

Not available

# SapTransfer



The SapTransfer Class implements functionality for managing a generic transfer process, that is, the action of transferring data from one source node to a destination node. All the following classes derived from the SapXferNode Class are considered to be transfer nodes: SapAcquisition, SapAcqDevice, SapBuffer. The following classes, also considered as transfer nodes, are documented in the *Sapera LT ++ Legacy Classes Reference Manual*: SapBufferRoi, SapBufferWithTrash, SapBufferRemote, SapCab, SapDsp, and SapPixPro.

There are also a number of Specialized Transfer Classes available, for example, SapAcqToBuf. These classes are all derived from SapTransfer, and they may be used to implement common transfer configurations.

```
#include <SapClassBasic.h>
```

## SapTransfer Class Members

### Construction

<a href="#"><u>SapTransfer</u></a>	Class constructor
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources
<b>Attributes</b>	
<a href="#"><u>GetLocation</u></a>	Gets/sets the location where the transfer resource is located
<a href="#"><u>SetLocation</u></a>	
<a href="#"><u>AddPair</u></a>	Adds a new pair of source and destination transfer nodes
<a href="#"><u>GetNumPairs</u></a>	Gets the number of pairs of source and destination transfer nodes
<a href="#"><u>GetPair</u></a>	Gets access to a specific transfer pair
<a href="#"><u>RemoveAllPairs</u></a>	Removes all transfer pairs
<a href="#"><u>SetCallbackInfo</u></a>	Sets the application callback method for transfer events and the associated context
<a href="#"><u>SetTrashCallbackInfo</u></a>	Sets the trash buffer application callback method for transfer events
<a href="#"><u>GetCallback</u></a>	Gets the current application callback function for transfer events
<a href="#"><u>GetTrashCallback</u></a>	Gets the current trash buffer application callback function for transfer events
<a href="#"><u>GetContext</u></a>	Gets the application context associated with transfer events
<a href="#"><u>IsGrabbing</u></a>	Checks whether continuous data transfer is currently in progress
<a href="#"><u>IsAutoEmpty</u></a>	Gets/sets the auto-empty mechanism
<a href="#"><u>SetAutoEmpty</u></a>	
<a href="#"><u>IsAutoConnect</u></a>	Gets/sets automatic activation of physical transfer data paths in the Create method
<a href="#"><u>SetAutoConnect</u></a>	
<a href="#"><u>IsConnected</u></a>	Checks whether the physical transfer data paths have been activated
<a href="#"><u>GetStartMode</u></a>	Gets/set the synchronization mode used when starting a data transfer
<a href="#"><u>SetStartMode</u></a>	
<a href="#"><u>IsCycleModeAvailable</u></a>	Gets the availability of a specific buffer cycling mode for a specific transfer pair
<a href="#"><u>GetConnectTimeout</u></a>	Gets/sets the communication timeout value for the Connect method
<a href="#"><u>SetConnectTimeout</u></a>	

<a href="#"><u>GetCounterStampInfo</u></a>	Gets the destination buffer counter stamp capabilities for a specific transfer pair
<a href="#"><u>GetHandle</u></a>	Gets the low-level Sapera handle of the transfer resource
<a href="#"><u>RegisterCallback</u></a>	Registers a callback function for the event associated with a specified name or index
<a href="#"><u>UnregisterCallback</u></a>	Unregisters a callback function on the event associated with a specified name or index

## Operations

<a href="#"><u>Init</u></a>	Performs the setup for data transfers
<a href="#"><u>Connect</u></a>	Activates the physical transfer data paths
<a href="#"><u>Disconnect</u></a>	Deactivates the physical transfer data paths
<a href="#"><u>Select</u></a>	Sets the current source and destination resource indexes
<a href="#"><u>Snap</u></a>	Transfers a predetermined number of frames
<a href="#"><u>Grab</u></a>	Starts continuous data transfer
<a href="#"><u>Freeze</u></a>	Issues a stop request for continuous data transfer
<a href="#"><u>Abort</u></a>	Stops the data transfer immediately using brute force
<a href="#"><u>Wait</u></a>	Waits for complete termination of data transfer
<a href="#"><u>IsCapabilityValid</u></a>	Checks for the availability of a low-level Sapera C library capability
<a href="#"><u>IsParameterValid</u></a>	Checks for the availability of a low-level Sapera C library parameter
<a href="#"><u>GetCapability</u></a>	Gets the value of a low-level Sapera C library capability
<a href="#"><u>SetParameter</u></a>	Gets/sets the value of a low-level Sapera C library parameter
<a href="#"><u>SetParameter</u></a>	

## SapTransfer Member Functions

The following are members of the SapTransfer Class.

---

### SapTransfer::SapTransfer

<b>SapTransfer(</b>	
SapXferCallback <i>pCallback</i> = NULL,	
void* <i>pContext</i> = NULL,	
SapLocation <i>loc</i> = SapLocation::ServerUnknown	
) ;	
<b>SapTransfer(</b>	
SapXferCallback <i>pCallback</i> ,	
SapXferCallback <i>pTrashCallback</i> ,	
void* <i>pContext</i> ,	
SapLocation <i>loc</i> = SapLocation::ServerUnknown	
) ;	

#### Parameters

<i>pCallback</i>	Application callback function to be called each time a transfer event happens. The callback function must be declared as: void MyCallback(SapXferCallbackInfo* <i>pInfo</i> );
<i>pTrashCallback</i>	Application callback function to be called each time a trash buffer transfer event happens
<i>pContext</i>	Optional pointer to an application context to be passed to the callback function. If <i>pCallback</i> is NULL, this parameter is ignored.
<i>loc</i>	SapLocation object specifying the server on which the transfer resource is to be created

#### Remarks

The SapTransfer constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

See the SapXferCallbackInfo class for information on the functions available to retrieve information about registered events. However, the constructor only allows access to a subset of the SapXferCallbackInfo:

- SapXferCallbackInfo::GetTransfer

- SapXferCallbackInfo::GetContext
- SapXferCallbackInfo::GetEventType
- SapXferCallbackInfo::GetEventCount
- SapXferCallbackInfo::IsTrash
- SapXferCallbackInfo::GetPairIndex

Use the SapTransfer::RegisterCallback function if you require access to newer functionality available in SapXferCallbackInfo (if supported by hardware), such as 64-bit event types, custom data, and host and auxiliary timestamps. SapXferCallbackInfo functions available exclusively when registering an event with SapTransfer::RegisterCallback are:

- SapXferCallbackInfo::GetAuxiliaryTimestamp
- SapXferCallbackInfo::GetCustomData
- SapXferCallbackInfo::GetCustomSize
- SapXferCallbackInfo::GetEventInfo
- SapXferCallbackInfo::GetGenericParam0  
SapXferCallbackInfo::GetGenericParam1  
SapXferCallbackInfo::GetGenericParam2  
SapXferCallbackInfo::GetGenericParam3
- SapXferCallbackInfo::GetHostTimestamp

You can use the Specialized Transfer Classes (for example, SapAcqToBuf) instead of using this class directly, since they simplify the process of instantiating SapTransfer objects that correspond to common transfer configurations. If you use this class, you must use the AddPair method to add transfer pairs of source and destination nodes. You must do this before calling the Create method.

Trash buffer functionality is only available when a SapBufferWithTrash object is used as a destination transfer node. In this case, the regular callback function is also used for trash buffers, unless you override it using *pTrashCallback*. If you do not use SapBufferWithTrash, then trash buffer settings are ignored.

The specified *pCallback* and *pContext* apply to all transfer pairs by default, unless you override it for specific pairs using the SapXferPair::SetCallbackInfo method.

By default, regular and trash buffer callback functions are called at each end of frame event, that is, when a complete image has been transferred. You may specify different event types for regular buffers by calling the SapXferPair::GetEventType, SapXferPair::SetEventType method. You cannot change the event type for trash buffers, however.

The server index of the *loc* argument may be set to SapLocation::ServerUnknown. In this case, the most appropriate server for the low-level transfer resource is automatically selected when you call the Create method. The *loc* resource index is ignored.

### **Demo/Example Usage**

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

## **SapTransfer::Abort**

**BOOL Abort();**

### **Return Value**

Returns TRUE if successful, FALSE otherwise

### **Remarks**

Stops data transfers immediately using brute force, without waiting for the current frame to be completely transferred.

You should call Abort only for emergencies. For example, calling Wait after the Snap or Grab methods may fail because of a timeout condition (usually hardware-related). In this case, using Abort is often the only way to correct the situation.

### **Demo/Example Usage**

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo,

## SapTransfer::AddPair

BOOL **AddPair**(SapXferPair &*pair*);

### Parameters

*pair* Transfer pair of source and destination nodes

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Adds a new pair of source and destination transfer nodes to the current object. You can only call this method before the Create method. However, you do not need to call it if you are using the Specialized Transfer Classes.

See the SapXferPair Class for more details.

### Demo/Example Usage

Not available

---

## SapTransfer::Connect

BOOL **Connect**();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Activates the physical transfer data paths associated with a transfer object.

You normally do not need to use this method, as it is called automatically by the Create method. It is useful when used together with the Disconnect method, as in the following case:

```
pXfer->Disconnect();
// Modify some transfer parameters
pXfer->Connect();
```

This allows the modification of transfer parameters (attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method, since these are not accessible after calling Destroy.

The Create method can also skip the call to Connect altogether, if you first call the SetAutoConnect method to turn off auto-connect, as in the following case:

```
pXfer->SetAutoConnect(FALSE);
pXfer->Create();
// Modify some transfer parameters
pXfer->Connect();
```

When calling this method to connect a transfer object with a very large number of buffers, you may encounter a timeout condition. This is due to the fact that the amount of time needed to successfully complete the command is larger than the default Sapera LT command timeout value. In this case, you can use the SetConnectTimeout method to increase this value.

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example

---

## SapTransfer::Create

BOOL **Create**();

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Creates all the low-level Sapera resources needed by the transfer object. Always call this method after the Create methods of source and destination nodes for all transfer pairs.

By default, Create automatically calls the Connect method to activate the physical transfer data paths. Calling SetAutoConnect(FALSE) allows you to change values of transfer parameters (or attributes) through methods in

the SapXferPair Class, or through calls to the SetParameter method, after calling Create. You must then call Connect explicitly to complete the setup of the transfer resource.

When calling this method to create a transfer object with a very large number of buffers, you may encounter a timeout condition. This is due to the fact that the amount of time needed to successfully complete the command is larger than the default Sapera LT command timeout value. In this case, you can use the SetConnectTimeout method to increase this value.

#### **Demo/Example Usage**

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### **SapTransfer::Destroy**

**BOOL Destroy();**

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

#### **Remarks**

Destroys all the low-level Sapera resources needed by the transfer object. Always call this method before the Destroy methods of source and destination nodes for all transfer pairs.

Note that Destroy automatically calls the Disconnect method to deactivate the physical transfer data paths associated with the transfer object.

#### **Demo/Example Usage**

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### **SapTransfer::Disconnect**

**BOOL Disconnect();**

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

#### **Remarks**

Deactivates the physical transfer data paths associated with a transfer object.

You normally do not need to use Disconnect, as it is called automatically by the Destroy method. It is only useful when used together with the Connect method.

See the Connect method for more details.

#### **Demo/Example Usage**

GigE Auto-White Balance Example

---

### **SapTransfer::Freeze**

**BOOL Freeze();**

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

#### **Remarks**

Issues a stop request for the current continuous transfer (started with the Grab method). The actual data transfer will end only after the current frame is completely transferred, so you should call the Wait method immediately after Freeze to ensure correct synchronization.

#### **Demo/Example Usage**

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### **SapTransfer::GetCallback**

SapXferCallback **GetCallback()**;

### Remarks

Gets the current application callback function for transfer events. The initial value for this attribute is NULL unless you specify another value in the constructor.

See the SapTransfer constructor for more details.

### Demo/Example Usage

Not available

---

## SapTransfer::GetCapability

BOOL **GetCapability**(int *cap*, void\* *pValue*);

### Parameters

*cap* Low-level Sapera C library capability to read

*pValue* Pointer to capability value to read back

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

This method allows direct read access to low-level Sapera C library capabilities for the transfer module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapTransfer Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

### Demo/Example Usage

GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo

---

## SapTransfer::GetConnectTimeout, SapTransfer::SetConnectTimeout

int **GetConnectTimeout()**;

BOOL **SetConnectTimeout** (int *timeout*);

### Remarks

Gets/sets the communication timeout override (in milliseconds) for the Connect method.

The time required by Connect can be high when the amount of memory taken by the buffer resources is very large, and can even exceed the Sapera LT communication timeout value (obtained by calling SapManager::GetCommandTimeout). In this case, the call to Connect fails with a timeout condition. The *timeout* argument can then be used to specify a larger amount of time. The largest of this value and of the communication timeout value is then used internally by Connect.

The new timeout value is used either when Connect is called directly by application code, or automatically through the Create method.

The initial value for this attribute is 0.

### Demo/Example Usage

Not available

---

## SapTransfer::GetContext

void\* **GetContext()**;

### Remarks

Gets the application context associated with transfer events. This context is the same for regular and trash buffer callback functions, even if you explicitly specified a different trash buffer function in the SapTransfer constructor or using the SetTrashCallbackInfo method.

The initial value for this attribute is NULL unless you specify another value in the constructor.

See the SapTransfer constructor for more details.

### Demo/Example Usage

Not available

---

## SapTransfer::GetCounterStampInfo

```
const SapXferCounterStampInfo* GetCounterStampInfo(int pairIndex);
```

### Parameters

*pairIndex* Index of the desired transfer pair

### Remarks

Gets the destination buffer counter stamp capabilities for a specific transfer pair.

The returned SapXferCounterStampInfo object has the following attributes:

BOOL IsSupported()	Returns TRUE if the current transfer device can report these capabilities
BOOL IsAvailable()	Returns TRUE if counter stamp is available
int GetMaxValue()	Returns the maximum counter stamp value
SapXferPair::EventType GetEventType()	Returns the possible event types (combined using bitwise OR) that identify the reference point for the counter stamp. See the SapXferPair::GetEventType, SapXferPair::SetEventType method for a list of possible values.
SapXferPair:: CounterStampTimeBaseGetTimeBase()	Returns the possible base units (combined using bitwise OR) used for the counter stamp. See the SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase method for a list of possible values.

### Demo/Example Usage

Not available

---

## SapTransfer::GetHandle

```
CORHANDLE GetHandle();
```

### Remarks

Gets the low-level Sapera handle of the transfer resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

### Demo/Example Usage

Not available

---

## SapTransfer::GetLocation, SapTransfer::SetLocation

```
SapLocation GetLocation();  
BOOL SetLocation(SapLocation location);
```

### Remarks

Gets/sets the location where the transfer resource is located.

If you specify a value for this attribute in the SapTransfer constructor, then it is used as the location of all SapXferPair objects that belong to this transfer object.

If you do not specify a value for this attribute, then it defaults to SapLocation::ServerUnknown. When the Create method is called, each SapXferPair object will then use the most appropriate location using the source and destination transfer nodes for the pair.

### Demo/Example Usage

Not available

---

## SapTransfer::GetNumPairs

```
int GetNumPairs();
```

### Remarks

Gets the number of pairs of source and destination transfer nodes. This value starts at 0 when the transfer object is constructed, increments by 1 at each call to the AddPair method, and is reset to 0 by the RemoveAllPairs method.

### Demo/Example Usage

Not available

---

### SapTransfer::GetPair

SapXferPair\* **GetPair**(int *pairIndex*);

#### Parameters

*pairIndex* Index of the desired transfer pair

#### Remarks

Gets access to a specific pair of source and destination transfer nodes. Valid pair indices go from 0 to the value returned by the GetNumPairs method minus 1.

See the SapXferPair Class for more details.

#### Demo/Example Usage

GigE Camera Demo, GigE Sequential Grab Demo, Sequential Grab Demo

---

### SapTransfer::GetParameter, SapTransfer::SetParameter

BOOL **GetParameter**(int *param*, void\* *pValue*);

BOOL **SetParameter**(int *param*, int *value*);

BOOL **SetParameter**(int *param*, void\* *pValue*);

#### Parameters

*param* Low-level Sapera C library parameter to read or write

*pValue* Pointer to parameter value to read back or to write

*value* New parameter value to write

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the transfer module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapTransfer or SapXferPair Class. Also, directly setting parameter values may interfere with the correct operation of the class.

Since many parameters cannot be changed when the physical transfer data paths are activated, you may need to use the Disconnect and Connect methods when modifying parameter values. See the Connect method for more details.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

#### Demo/Example Usage

Not available

---

### SapTransfer::GetStartMode, SapTransfer::SetStartMode

SapTransfer::StartMode **GetStartMode**();

BOOL **SetStartMode**(SapTransfer::StartMode *startMode*);

#### Parameters

*startMode* The following transfer synchronization modes are available when starting a transfer using the Snap method:

SapTransfer::StartAsynchronous Return immediately without waiting for the transfer to begin

SapTransfer::StartSynchronous For single frame transfers, first wait for any active transfer to end, and return only when the current transfer has been completed.

SapTransfer::StartHalfAsynchronous For single frame transfers, first wait for any active transfer to end, then immediately return without waiting

`SapTransfer::StartSequential`

for the current transfer to begin.

If a multi-level transfer is defined (that is, acquisition to on-board memory to host memory), wait until all frames in the sequence are in the on-board memory before sending them to the host memory.

### Remarks

Gets/sets the synchronization mode used when starting a data transfer. The default value for this attribute is StartAsynchronous.

You can only call SetStartMode before the Create method.

### Demo/Example Usage

Not available

---

## SapTransfer::GetTrashCallback

`SapXferCallback GetTrashCallback();`

### Remarks

Gets the current trash buffer application callback function for transfer events. This function is the same as the one returned using the GetCallback method, unless you explicitly specified a trash buffer callback function in the SapTransfer constructor or using the SetTrashCallbackInfo method

The initial value for this attribute is NULL unless you specify another value in the constructor. See the SapTransfer constructor for more details.

### Demo/Example Usage

Not available

---

## SapTransfer::Grab

`BOOL Grab();`

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Starts a continuous transfer from the source node to the destination node of all transfer pairs in the current SapTransfer object.

Continuous transfers are always started asynchronously, that is, no explicit checking is performed to verify if a previous transfer is still active. If you want to perform this check, then you first need to call the Wait method.

If you call the Select method before Grab, then the transfer will be performed starting at the new current source and destination resources indexes. Otherwise, the transfer will proceed using the indexes from the end of the previous transfer operation (using Snap or Grab). If there is no previous transfer, then appropriate defaults from the call to the Create method will be used.

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

## SapTransfer::Init

`BOOL Init(BOOL resetIndex = TRUE);`

### Parameters

`resetIndex`      TRUE to initialize the buffer index, FALSE otherwise

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

Performs the setup for data transfers. Set `resetIndex` to TRUE if you also want to set all destination buffer resources to the empty state, and set the SapBuffer index to the first buffer in its list (through the SapBuffer::ResetIndex method).

You usually do not have to call Init explicitly, since the Create method already does this.

## Demo/Example Usage

FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Sequential Grab Demo

---

## SapTransfer::IsAutoConnect, SapTransfer::SetAutoConnect

```
BOOL IsAutoConnect();
void SetAutoConnect(BOOL bAutoConnect);
```

### Remarks

Gets/sets automatic activation of physical transfer data paths. Calling the Create method automatically calls the Connect method when this attribute is TRUE.

Setting auto-connect to FALSE allows you to change values of transfer parameters (attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method, after calling Create. You must then call Connect explicitly to complete the setup of the transfer resource.

The initial value for this attribute is TRUE, unless you construct this object using an existing SapTransfer object.

## Demo/Example Usage

Not available

---

## SapTransfer::IsAutoEmpty, SapTransfer::SetAutoEmpty

```
BOOL IsAutoEmpty();
void SetAutoEmpty(BOOL bAutoEmpty);
```

### Remarks

Gets/sets the auto-empty mechanism, used for synchronizing the transfer with the processing and/or view tasks in the application program.

By default, this class automatically calls SapBuffer::SetState(SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the processing or view task can even use it.

In this case, you should call SetAutoEmpty(FALSE) to disable this behavior in this class. You then call SapProcessing:: SetAutoEmpty(TRUE) or SapView:: SetAutoEmpty(TRUE), depending on which processing and view task is executed last. Exactly one of the three classes must empty the buffer.

It is also possible to completely disable the auto-empty mechanism for the SapTransfer, SapProcessing, and SapView, classes. In this case, you must explicitly call SapBuffer::SetState to empty buffers whenever you have finished using their contents.

The auto-empty mechanism does not apply when the destination node is not a SapBuffer object.

The initial value for this attribute is TRUE, unless you construct this object using an existing SapTransfer object.

## Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

---

## SapTransfer::IsCapabilityValid

```
BOOL IsCapabilityValid(int cap);
```

### Parameters

*cap* Low-level Sapera C library capability to check

### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

### Remarks

Checks for the availability of a low-level Sapera C library capability for the transfer module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapTransfer class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

## Demo/Example Usage

GigE Sequential Grab Demo, IO Demo, Sequential Grab Demo

---

## SapTransfer::IsConnected

**BOOL IsConnected();**

### Remarks

Checks whether the physical transfer data paths have been activated. By default, calling the Create method automatically invokes the Connect method, so that IsConnected returns TRUE. If you call SetAutoConnect(FALSE) before calling the Create method, then IsConnected returns FALSE.

If you explicitly call the Connect method, then IsConnected returns TRUE. If you explicitly call the Disconnect method, then IsConnected returns FALSE.

The initial value for this attribute is FALSE.

### Demo/Example Usage

Not available

---

## SapTransfer::IsCycleModeAvailable

**BOOL IsCycleModeAvailable(int pairIndex, SapXferPair::CycleMode cycleMode);**

### Parameters

*pairIndex* Index of the desired transfer pair

*cycleMode* Cycle mode to check for

### Remarks

Gets the availability of a specific buffer cycling mode for a specific transfer pair. Valid pair indices go from 0 to the value returned by the GetNumPairs method minus 1.

See the SapXferPair::GetCycleMode method for a list of valid values for the *cycleMode* argument..

### Demo/Example Usage

Not available

---

## SapTransfer::IsGrabbing

**BOOL IsGrabbing();**

### Remarks

Returns TRUE if continuous data transfer is in progress, FALSE otherwise. Use the Grab method to initiate continuous transfer.

The value of this attribute is only relevant after calling the Create method. Otherwise, it always returns FALSE.

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo

---

## SapTransfer::IsParameterValid

**BOOL IsParameterValid(int param);**

### Parameters

*param* Low-level Sapera C library parameter to check

### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

### Remarks

Checks for the availability of a low-level Sapera C library parameter for the transfer module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapTransfer class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapTransfer::RegisterCallback

```
BOOL RegisterCallback(EventType eventType, SapXferCallback callback, void *context, UINT32 xferElement);
```

#### Parameters

*eventType* Event type. See the acquisition device User's Manual for the list of supported transfer events.

*callback* Address of a user callback function of the following form:

```
void MyCallback(SapAcqDeviceCallbackInfo* pInfo)
{
}
```

*context* Pointer to a user storage (that is, variable, structure, buffer, etc). Can be NULL.

*xFerElement* Possible values are:

ElementPair Sets the callback for a source destination pair

ElementGroup Sets the callback for a source and all its destination pairs

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Registers an event by associating a callback function for the specified event type. When the event occurs in the transfer module, this callback function is called. It provides information on the corresponding event using a SapXferCallbackInfo object. Refer to this class for more details.

Use the SapTransfer::Select function to select the source/destination pair for which to register the callback. If setting a callback using the *ElementGroup* setting, the callback is registered for the all the source/destination pairs for the source of the currently selected source/destination pair.

The context pointer is also returned by the callback function, allowing for the exchange application specific information.

#### Example

```
void MyCallback(SapAcqDeviceCallbackInfo* pInfo)
{
    // Access information using functions of SapAcqDeviceCallbackInfo class
    // ...
}

main()
{
    // ...
    xFer.RegisterCallback("FeatureValueChanged", MyCallback, NULL);
    // ...
    xFer.UnregisterCallback("FeatureValueChanged");
    // ...
}
```

#### Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

---

### SapTransfer::RemoveAllPairs

```
BOOL RemoveAllPairs();
```

#### Remarks

Removes all pairs of source and destination transfer nodes

You can only call this method before the Create method or after the Destroy method.

#### Demo/Example Usage

Not available

---

### SapTransfer::Select

```
BOOL Select(SapXferPair *pPair, int srcIndex = -1, int dstIndex = -1);
```

```
BOOL Select(int pairIndex, int srcIndex = -1, int dstIndex = -1);
```

#### Parameters

*pPair* Pointer to new transfer pair

*srcIndex* New resource index for source transfer node

*dstIndex* New resource index for destination transfer node

*pairIndex* Index of new transfer pair

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Sets a new transfer pair and the current source/destination transfer node resource indexes.

There is usually only one transfer pair per SapTransfer object, in which case the *pairIndex* argument is 0. The source node is usually a SapAcquisition or SapAcqDevice object, in which case the *srcIndex* argument is 0. Since the destination node is usually a SapBuffer object, the *dstIndex* argument then represents a buffer resource index.

Setting *srcIndex* and *dstIndex* to -1 allows for the selection of a new transfer pair while keeping its current source and destination resources indexes.

The Select method is useful in two cases. It allows the selection of pair and resource indexes before changing values of transfer parameters (or attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method. It also allows precise selection of the current transfer node resource indexes before calling the Snap or Grab methods. It is then possible, for example, to know precisely in which buffer resource the next image will be acquired.

#### Demo/Example Usage

Not available

---

### SapTransfer::SetCallbackInfo

BOOL **SetCallbackInfo**(SapXferCallback *pCallback*, void\* *pContext* = NULL);

#### Remarks

Sets the application callback method for transfer events and the associated context. You can only call SetCallbackInfo before the Create method.

See the SapTransfer constructor for more details.

#### Demo/Example Usage

Not available

---

### SapTransfer::SetTrashCallbackInfo

BOOL **SetTrashCallbackInfo**(SapXferCallback *pTrashCallback*);

#### Remarks

Sets the application callback function for trash buffer transfer events. If you do not call SetTrashCallbackInfo, trash buffers use the same callback function as regular buffers. The associated context information remains the same as for regular buffers.

If you set the value of this attribute to NULL, then the application will receive no trash buffer callbacks.

You can only call SetTrashCallbackInfo before the Create method. See the SapTransfer constructor for more details.

---

### SapTransfer::Snap

BOOL **Snap**(int *count* = 1);

#### Parameters

*count* Number of frames to be transferred

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Transfers a finite number of frames (usually 1) from the source node to the destination node of all transfer pairs in the current SapTransfer object.

By default, transfers are started asynchronously. You may need to call the Wait method immediately after Snap to ensure correct synchronization. See the SetStartMode method if you need to use a different synchronization mode for single frame transfers (*count* = 1).

If you call the Select method before Snap, then the transfer will be performed using the new current source and destination resource indexes. Otherwise, the transfer will proceed using the indexes from the end of the previous transfer operation (using Snap or Grab). If there is no previous transfer, then appropriate defaults

from the call to the Create method will be used.

When using this function together with SapXferPair::SetFramesPerCallback, the value of *count* should be a multiple of the number of frames per callback, otherwise, the application behavior is undefined. Typically, the application callback function will not get invoked for any leftover frames. For example, if you acquire 10 frames and the number of frames per callback is 4, then you may not get the application callback for the last two frames.

There is a special case when both the source and destination nodes are SapBuffer objects. First, only one transfer pair is used. Also, the data transfer is actually a buffer to buffer copy operation, with format conversion if necessary. Finally, the start mode is ignored.

#### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example

---

### SapTransfer::UnregisterCallback

**BOOL UnregisterCallback();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Unregisters a callback function on the transfer event.

#### Demo/Example Usage

---

### SapTransfer::Wait

**BOOL Wait(int *timeout*);**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Parameters

*timeout* Maximum amount of time to wait, in milliseconds

#### Remarks

Waits for the complete termination of data transfer. You may want to call Wait after Snap to make certain that the required number of frames have been transferred before proceeding. You should definitely call Wait after initiating continuous transfer with Grab and ending it with Freeze.

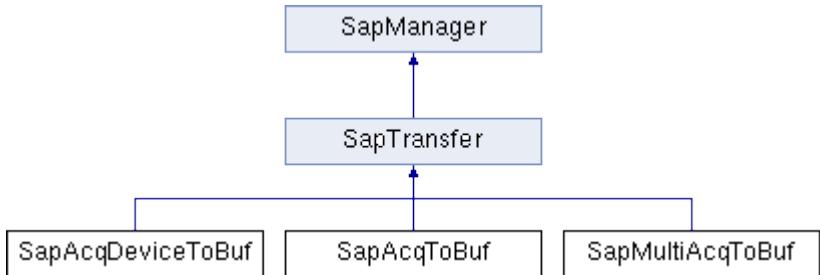
If the specified *timeout* expires, and transfer is still not completed, then Wait returns an error. A common reason for this error is some kind of hardware failure. In this case, call the Abort method to unconditionally terminate the transfer.

You may also get an error if the *timeout* is too small, and does not give the transfer enough time to terminate gracefully. So you should always specify a value large enough to allow one full frame to be transferred. You may even specify a much larger value (like a few seconds), if your application allows it.

#### Demo/Example Usage

GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

# Specialized Transfer Classes



The Specialized Transfer Classes are a set of classes derived from `SapTransfer` that allow you to more easily create the most commonly used transfer configurations.

All the classes have the same naming convention, that is, `SapXxxToYyy`, where `Xxx` and `Yyy` identify the source and destination nodes, respectively. For example, use the `SapAcqToBuf` Class to connect a `SapAcquisition` object to a `SapBuffer` object.

Each of these classes has one or more specific constructors; otherwise, they use the same methods as the [SapTransfer](#) class.

If you need a transfer configuration that is not supported by any of the specialized classes, then you must use the `SapTransfer` class directly instead.

```
#include <SapClassBasic.h>
```

## Common Constructor Arguments

All specialized transfer classes constructors include the following two arguments:

- `pCallback` Application callback function to be called each time a transfer event happens. The callback function must be declared as:  
void MyCallback(SapXferCallbackInfo \*pInfo);
- `pContext` Optional pointer to an application context to be passed to the callback function. If `pCallback` is NULL, this parameter is ignored.

## SapAcqToBuf Class

```
SapAcqToBuf(SapAcquisition* pAcq, SapBuffer* pBuf, SapXferCallback pCallback = NULL,  
void* pContext = NULL);
```

### Parameters

- `pAcq` Source acquisition object  
`pBuf` Destination buffer object

### Remarks

Implements a transfer from an acquisition object to a buffer object

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

## SapAcqDeviceToBuf Class

```
SapAcqDeviceToBuf(SapAcqDevice* pAcqDevice, SapBuffer* pBuf, SapXferCallback pCallback = NULL,  
void* pContext = NULL);
```

### Parameters

- `pAcqDevice` Source acquisition device object  
`pBuf` Destination buffer object

### Remarks

Implements a transfer from an acquisition device object (for example, for a Genie camera) to a buffer object

### Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, GigE Auto-White Balance Example,  
GigE Camera LUT Example, Grab Console Example

## SapMultiAcqToBuf Class

**SapMultiAcqToBuf**(SapAcquisition\* *pAcq*[], SapBuffer\* *pBuf*[], int *numPairs*,  
SapXferCallback *pCallback* = NULL, void\* *pContext* = NULL);

### Parameters

*pAcq* List of source acquisition objects  
*pBuf* List of destination buffer object  
*numPairs* Number of entries in acquisition and buffer lists

### Remarks

Implements a transfer from a series of acquisition objects to a matching number of buffer objects. There is a one-to-one relationship between items in the source list and items in the destination list. All acquisition objects must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method returns TRUE.

### Demo/Example Usage

Not available

# SapView



The SapView Class includes the functionality to display the resources of a SapBuffer object in a window. It allows you to display the current buffer resource, a specific one, or the next one not yet displayed.

An internal thread optimizes buffer display in realtime. This allows the main application thread to execute without any concerns for the display task.

An auto empty mechanism allows synchronization between SapView and SapTransfer objects to show buffers in real-time without missing any data.

```
#include <SapClassBasic.h>
```

## SapView Class Members

### Construction

<a href="#"><u>SapView</u></a>	Class constructor
<a href="#"><u>Create</u></a>	Allocates the low-level Sapera resources
<a href="#"><u>Destroy</u></a>	Releases the low-level Sapera resources
<b>Attributes</b>	
<a href="#"><u>GetBufferr</u></a>	Gets/sets the SapBuffer object with the buffer resources to display
<a href="#"><u>SetBuffer</u></a>	
<a href="#"><u>GetDisplay</u></a>	Gets/sets the SapDisplay object with the display device associated with the view
<a href="#"><u>SetDisplay</u></a>	
<a href="#"><u>GetWindow</u></a>	Gets/sets the GDI window handle used for showing buffers
<a href="#"><u>SetWindow</u></a>	
<a href="#"><u>SetCallbackInfo</u></a>	Sets the application callback method to call after displaying each buffer and the associated context
<a href="#"><u>GetCallback</u></a>	Gets the current application callback method
<a href="#"><u>GetContext</u></a>	Gets the application context associated with the application callback method
<a href="#"><u>GetWidth</u></a>	Gets the width (in pixels) of the displayed buffer area
<a href="#"><u>GetHeight</u></a>	Gets the height (in lines) of the displayed buffer area
<a href="#"><u>GetViewArea</u></a>	Gets the width and height of the viewing area
<a href="#"><u>GetScrollPos</u></a>	Gets the current scrolling position of the viewing area relative to buffer coordinates
<a href="#"><u>GetScrollRange</u></a>	Gets the scrolling range of the viewing area relative to buffer coordinates
<a href="#"><u>GetIndex</u></a>	Gets the index of the last displayed buffer
<a href="#"><u>IsAutoEmpty</u></a>	Gets/sets the auto-empty mechanism
<a href="#"><u>SetAutoEmpty</u></a>	
<a href="#"><u>GetOverlayMode</u></a>	Gets/sets the viewing mode when dealing with buffers of overlay type
<a href="#"><u>SetOverlayMode</u></a>	
<a href="#"><u>GetKeyColor</u></a>	Gets/sets the keying color for buffers of overlay type
<a href="#"><u>SetKeyColor</u></a>	
<a href="#"><u>GetScalingMode</u></a>	Gets/sets the mode specifying how buffer content is scaled to the viewing area
<a href="#"><u>SetScalingMode</u></a>	
<a href="#"><u>GetImmediateMode</u></a>	Gets/sets the view thread bypass mode
<a href="#"><u>SetImmediateMode</u></a>	
<a href="#"><u>GetWindowTitle</u></a>	Gets/sets the title of view windows automatically created by SapView

<u>SetTitle</u>	
<u>HasRange</u>	Checks if the view resource can show a subrange of buffer data bits
<u>GetRangeMinMax</u>	Gets the minimum and maximum viewing range values
<u>GetRange</u>	Gets/sets the viewing range value
<u>SetRange</u>	
<u>GetThreadPriority</u>	Gets/sets the execution priority of the viewing thread
<u>SetThreadPriority</u>	
<u>GetHandle</u>	Gets the low-level Sapera handle of the view resource
<b>Operations</b>	
<u>Init</u>	Initializes the view index
<u>Show</u>	Shows the next buffer or a specific one, possibly skipping buffers in the process
<u>ShowNext</u>	Shows the next buffer, without skipping any buffers in the process
<u>Hide</u>	Hides the currently displayed buffer
<u>GetDC</u>	Gets the Windows Device Context corresponding to the view window
<u>ReleaseDC</u>	Releases the Windows Device Context corresponding to the view window
<u>GetLut</u>	Gets the current view lookup table
<u>ApplyLut</u>	Programs a new view lookup table
<u>OnPaint</u>	Shows the last displayed buffer again following a WM_PAINT message
<u>OnMove</u>	Adjusts the position of the viewing window following a WM_MOVE message
<u>OnSize</u>	Adjusts the size of the viewing window following a WM_SIZE message
<u>OnHScroll</u>	Adjusts the horizontal scrolling position following a WM_HSCROLL message
<u>OnVScroll</u>	Adjusts the vertical scrolling position following a WM_VSCROLL message
<u>IsCapabilityValid</u>	Checks for the availability of a low-level Sapera C library capability
<u>IsParameterValid</u>	Checks for the availability of a low-level Sapera C library parameter
<u>GetCapability</u>	Gets the value of a low-level Sapera C library capability
<u>GetParameter</u>	Gets/sets the value of a low-level Sapera C library parameter
<u>SetParameter</u>	

## SapView Member Functions

The following are members of the SapView Class.

---

### SapView::SapView

```

SapView(
    SapBuffer* pBuffer = NULL,
    HWND hWnd = SapHwndDesktop,
    SapViewCallback pCallback = NULL,
    void* pContext = NULL
); SapView(
    SapDisplay* pDisplay,
    SapBuffer* pBuffer,
    HWND hWnd = SapHwndDesktop,
    SapViewCallback pCallback = NULL,
    void* pContext = NULL
);

```

#### Parameters

<i>pBuffer</i>	SapBuffer object with the buffer resources to display
<i>HWnd</i>	GDI window handle used for displaying buffers
<i>pCallback</i>	Application callback function to be called after each buffer has been displayed. The callback function must be declared as: void MyCallback(SapViewCallbackInfo* pInfo);

*pContext*      Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

*pDisplay*      Display object specifying on which display resource the buffers will be shown

#### Remarks

The SapView constructor does not actually create the low-level Saper resources. To do this, you must call the Create method. In addition to a regular window handle, you may use two special values for the *hWnd* argument. If it is equal to SapHwndDesktop, then buffers will be displayed directly on the desktop. If it is equal to SapHwndAutomatic, then SapView will automatically create a view window (supported on single monitor configurations only). The latter is especially useful in console applications, where you do not have a full GUI at your disposal.

You may specify the *pCallback* and *pContext* arguments in order to be notified each time a new buffer is displayed following to a call to the Show, ShowNext, or OnPaint methods. This may be useful if you need to draw graphics in non-destructive overlay. If you do not specify the *pDisplay* argument, then SapView automatically creates and uses an internal SapDisplay object corresponding to the system display. You must explicitly specify this argument if you use additional SapView objects which are located on displays other than the system display. Another reason to specify the *pDisplay* argument is to speed up creation of the display object, and to eliminate possible related flicker effects. See SapDisplay::GetFormatDetection, SapDisplay::SetFormatDetection methods for details.

#### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### SapView::ApplyLut

**BOOL ApplyLut();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Reprograms the view lookup table. After getting the current LUT using the GetLut method, use the methods in the SapLut Class to manipulate it. Then use ApplyLut to apply the changes.

This feature is currently available only when the SapDisplay object associated with the view is not located on the primary VGA in the system (see SapDisplay::IsPrimaryVGABoard).

#### Demo/Example Usage

Not available

---

### SapView::Create

**BOOL Create();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

#### Remarks

Creates all the low-level Saper resources needed by the view object. Always call this method after SapBuffer::Create.

If you manage the SapDisplay object needed by the view object yourself, you must also call this method after SapDisplay::Create. See the SapView constructor for more details.

#### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

### SapView::Destroy

**BOOL Destroy();**

#### Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Destroys all the low-level Sapera resources needed by the view object. Always call this method before SapBuffer::Destroy.

If you manage the SapDisplay object needed by the view object yourself, you must also call this method before SapDisplay::Destroy. See the SapView constructor for more details.

## Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, File Load MFC Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

## SapView::GetBuffer, SapView::SetBuffer

`SapBuffer* GetBuffer();`  
`BOOL SetBuffer(SapBuffer* pBuffer);`

## Parameters

`pBuffer` SapBuffer object containing the buffer resources to display

## Remarks

Gets/sets the SapBuffer object with the buffer resources to display. You set the initial value for this attribute through the SapView constructor.

You can only call SetBuffer before the Create method.

## Demo/Example Usage

Color Conversion Demo, Color Split Example

---

## SapView::GetCallback

`SapViewCallback GetCallback();`

## Remarks

Gets the current application callback method called after displaying each buffer. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapView constructor for more details.

## Demo/Example Usage

Not available

---

## SapView::GetCapability

`BOOL GetCapability(int cap, void* pValue);`

## Parameters

`cap` Low-level Sapera C library capability to read

`pValue` Pointer to capability value to read back

## Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

This method allows direct read access to low-level Sapera C library capabilities for the View Module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapView Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

## Demo/Example Usage

Not available

---

## SapView::GetContext

`void* GetContext();`

## **Remarks**

Gets the application context associated with the application callback method. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapView constructor for more details.

## **Demo/Example Usage**

Not available

---

### **SapView::GetDC**

**BOOL GetDC(HDC\* pDC);**

#### **Parameters**

*pDC* Pointer to display context value

#### **Return Value**

Returns TRUE if successful, FALSE otherwise

## **Remarks**

Gets the Windows Device Context corresponding to the current view area.

If the current SapView object does not use the system display (see SapDisplay::GetType), then GetDC returns the Windows Device Context corresponding to the entire display instead.

## **Demo/Example Usage**

Not available

---

### **SapView::GetDisplay, SapView::SetDisplay**

**SapDisplay\* GetDisplay();**

**BOOL SetDisplay(SapDisplay\* pDisplay);**

#### **Parameters**

*pDisplay* SapDisplay object specifying where the buffer resources are shown

## **Remarks**

Gets/sets the SapDisplay object specifying where the buffer resources are shown.

If you explicitly specify a SapDisplay object in the SapView constructor or through SetDisplay, then GetDisplay returns that object. If you do not, then SapView automatically creates an internal SapDisplay object when calling the Create method, and destroys it when calling the Destroy method. In this case, GetDisplay returns the internal object.

You can only call SetDisplay before the Create method.

## **Demo/Example Usage**

Color Conversion Demo, FlatField Demo, GigE Camera Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo

---

### **SapView::GetHandle**

**CORHANDLE GetHandle();**

## **Remarks**

Gets the low-level Sapera handle of the view resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

## **Demo/Example Usage**

Not available

---

### **SapView::GetHeight**

**int GetHeight();**

## **Remarks**

Gets the height (in lines) of the displayed buffer area. This value is equal to the minimum of the buffer height and the viewing area height width.

The value returned by GetHeight is only relevant after calling the Create method.

## Demo/Example Usage

Not available

---

## SapView::GetImmediateMode, SapView::SetImmediateMode

```
BOOL GetIMMEDIATEMode();  
void SetIMMEDIATEMode (BOOL immediateMode);
```

### Remarks

Gets/sets the view thread bypass mode.

By default, this mode is off, therefore calling the Show and ShowNext methods wake up an internal thread to handle buffer display. Since showing images is often a time-consuming process, this allows the calling thread to do other things instead.

If immediate mode is active, then the Show and ShowNext methods bypass the thread, and images are shown in the context of the calling thread instead.

The initial value for this attribute is FALSE.

## Demo/Example Usage

Not available

---

## SapView::GetIndex

```
int GetIndex();
```

### Remarks

Gets the index of the last displayed buffer. It is initialized to the current buffer index (usually 0) when you call the Create method. From then on, it is automatically updated following calls to the Show or ShowNext methods.

## Demo/Example Usage

Not available

---

## SapView::GetKeyColor, SapView::SetKeyColor

```
SapDataRGB GetKeyColor();  
BOOL SetKeyColor(SapDataRGB keyColor);
```

### Remarks

Gets/sets the keying color when dealing with buffers of overlay type (SapBuffer::TypeOverlay). See the SapDataRGB class for a description of the related data type.

For an 8-bit display mode, that is, when the SapDisplay::GetPixelDepth method returns 8, then only the red color component is relevant.

The initial value for this attribute corresponds to black. When calling the Create method, if the current viewing mode is overlay, then its value will be initialized using the current low level keying color value.

You can only call SetKeyColor after the Create method.

## Demo/Example Usage

Not available

---

## SapView::GetLut

```
SapLut* GetLut();
```

### Remarks

Gets the current view lookup table, which has already been automatically created and initialized when calling the Create method. You may manipulate the LUT through the methods in the SapLut Class, and reprogram it using the ApplyLut method.

GetLut returns NULL if the current view resource does not support lookup tables.

This feature is currently available only when the SapDisplay object associated with the view is not located on the primary VGA in the system (see SapDisplay::IsPrimaryVGABoard).

## Demo/Example Usage

Not available

---

## SapView::GetOverlayMode, SapView::SetOverlayMode

SapView::OverlayMode **GetOverlayMode()**;  
BOOL **SetOverlayMode**(SapView::OverlayMode *overlayMode*);

### Parameters

<i>overlayMode</i>	Viewing mode for buffers of overlay type, can be one of the following values:
SapView:: OverlayNone	Overlay mode is not initialized yet
SapView:: OverlayAlwaysOnTop	No color keying scheme is enabled. Buffer contents are displayed directly using the display adapter overlay hardware. This is the fastest method; however, other windows will not be displayed correctly if they overlap the Sapera application.
SapView:: OverlayAutoKeying	A destination color keying scheme is enabled. Source buffer pixels are displayed only if the corresponding pixel on the display has the key color. Each time a buffer is shown following calls to the Show or ShowNext methods, the current keying color is painted on the view surface. Also, the OnPaint method only repaints the keying color on the part of the view area that becomes visible again. This is usually the default mode.
SapView:: OverlayManualKeying	Similar to auto-keying mode, except that you are responsible for painting the key color in the view area. This gives you more flexibility as to where the overlay image should be displayed.

### Remarks

Gets/sets the viewing mode when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

The initial value for this attribute is OverlayNone. If you do not call SetOverlayMode before the Create method, then the latter will initialize its value appropriately.

### Demo/Example Usage

Not available

---

## SapView::GetParameter, SapView::SetParameter

BOOL **GetParameter**(int *param*, void\* *pValue*);  
BOOL **SetParameter**(int *param*, int *value*);  
BOOL **SetParameter**(int *param*, void\* *pValue*);

### Parameters

<i>param</i>	Low-level Sapera C library parameter to read or write
<i>pValue</i>	Pointer to parameter value to read back or to write
<i>value</i>	New parameter value to write

### Return Value

Returns TRUE if successful, FALSE otherwise

### Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the View Module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapView Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapView::GetRange, SapView::SetRange

int **GetRange()**;

```
BOOL SetRange(int range);
```

### Remarks

Gets/sets the viewing range value. Before using GetRange and SetRange, you should first check for availability of this feature using the HasRange and GetRangeMinMax methods.

The range value is the number of bits (starting from the most significant) that are not shown on the display. The default value is 0, that is, the most significant bits are shown. This is a problem when not all bits are used, for example, 10-bit data stored in the low-order bits of a 16-bit buffer. In this case, you should set the value to 6 for correct results.

You can only call GetRange and SetRange after the Create method.

### Demo/Example Usage

Not available

---

## SapView::GetRangeMinMax

```
void GetRangeMinMax(int* pRangeMin, int* pRangeMax);
```

### Parameters

*pRangeMin* Pointer to returned minimum range value

*pRangeMax* Pointer to returned maximum range value

### Remarks

Gets the minimum and maximum viewing range values allowed for the SetRange method. If both values are 0, then you cannot change the range.

You can only call GetRangeMinMax after the Create method.

### Demo/Example Usage

Not available

---

## SapView::GetScalingMode, SapView::SetScalingMode

```
SapView::ScalingMode GetScalingMode();
```

```
BOOL SetScalingMode(SapView::ScalingMode scalingMode, BOOL keepAspectRatio = FALSE);
```

```
BOOL SetScalingMode(float zoomHorz, float zoomVert);
```

```
BOOL SetScalingMode(SapViewScaleParams &srcParams, SapViewScaleParams &dstParams);
```

### Parameters

<i>scalingMode</i>	SapView:: ScalingNone  SapView:: ScalingFitToWindow  SapView:: ScalingZoom  SapView:: ScalingUserDefined	There is a one-to-one correspondence between buffer data and pixels shown in the view area. This is the default mode.  Displayed buffer contents are scaled so that they are shown completely in the view area. This results in distorted images if the width/height aspect ratio of the buffer is different from the aspect ratio of the view area.  Displayed buffer contents are scaled independently in the horizontal and vertical directions  Buffer contents are displayed using custom user-specified settings
<i>keepAspectRatio</i>	Set to TRUE to keep the image aspect ratio when using ScalingFitToWindow mode	
<i>zoomHorz</i>	Horizontal zooming factor to apply to displayed buffer contents	
<i>zoomVert</i>	Vertical zooming factor to apply to displayed buffer contents	
<i>srcParams</i>	Buffer area to be shown in the specified region of the viewing area	
<i>dstParams</i>	Region of the viewing area that will show the specified buffer area	

### Remarks

Gets/sets the mode specifying how buffer content is scaled to the viewing area.

The first form of this method allows you to specify one of two predefined modes: a one-to-one relationship between buffer contents and the view area (ScalingNone), or displaying buffer contents completely (ScalingFitToWindow).

The second form allows you to specify independent horizontal and vertical scaling factors (ScalingZoom). These apply to displayed images only, they do not affect buffer data. This results in distorted images if the factors are different.

The third form gives you complete control over the scaling mode (ScalingUserDefined). You need to specify the exact region in the source buffer and in the destination view area. SapView then automatically calculates the appropriate horizontal and vertical scaling factors.

The *srcParams* and *dstParams* arguments both define rectangular areas, as follows:

**SapViewScaleParams**(int *left*, int *top*, int *width*, int *height*)

The initial value for this attribute is ScalingNone.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapView::GetScrollPos

**POINT GetScrollPos()**;

#### Remarks

Gets the current scrolling position (as a Windows POINT structure) of the viewing area relative to buffer coordinates. The initial value is (0,0) and changes automatically through calls to the OnHScroll and OnVScroll methods. The maximum value depends on the scrolling range (see SapView::GetScrollRange).

Depending on the current view scaling mode, the scrolling position remains fixed at (0,0) if the buffer contents fit entirely within the view area.

The value returned by GetScrollPos is only relevant after calling the Create method.

See the SetScalingMode method for details.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapView::GetScrollRange

**SIZE GetScrollRange()**;

#### Remarks

Gets the scrolling range (as a Windows SIZE structure) of the viewing area relative to buffer coordinates. This range determines the maximum value of the scrolling position.

Depending on the current view scaling mode, the scrolling range is initialized from the number of lines and columns of the view buffer that cannot be shown in the view area. If its value is (0,0), then scrolling is disabled.

The value returned by this method is only relevant after calling the Create method.

See the SetScalingMode method for details.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapView::GetThreadPriority, SapView::SetThreadPriority

**int GetThreadPriority();**

**void SetThreadPriority(int priority);**

#### Remarks

Gets/sets the execution priority of the view thread. The initial value for this attribute is normal priority, unless you construct this object using an existing SapView object.

For a detailed description of this setting, refer to the SetThreadPriority function in the Win32 documentation.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapView::GetViewArea

**BOOL GetViewArea(int\* width, int\* height);**

#### Remarks

Gets the width and height of the viewing area. The value returned by this method is only relevant after calling the Create method.

See also the GetWidth and GetHeight methods.

#### Demo/Example Usage

---

### SapView::GetWidth

`int GetWidth();`

#### Remarks

Gets the width (in pixels) of the displayed buffer area. This value is equal to the minimum of the buffer width and the viewing area width.

The value returned by GetWidth is only relevant after calling the Create method.

#### Demo/Example Usage

Dual Acquisition Demo

---

### SapView::GetWindow, SapView::SetWindow

`HWND GetWindow();`  
`BOOL SetWindow(HWND hWnd);`

#### Parameters

`hWnd` GDI window handle used for displaying buffers

#### Remarks

Gets/sets the GDI window handle used for displaying buffers.

In addition to a regular window handle, you may use two special values. If `hWnd` is equal to `SapHwndDesktop`, then buffers will be displayed directly on the desktop. If it is equal to `SapHwndAutomatic`, then `SapView` will automatically create a view window (supported on single monitor configurations only). The latter is especially useful in console applications, where you do not have a full GUI at your disposal.

If you do not specify a value for this attribute in the `SapView` constructor, then it defaults to `SapHwndDesktop`.

You can only call SetWindow before the Create method.

#### Demo/Example Usage

Not available

---

### SapView::GetWindowTitle, SapView::SetWindowTitle

`BOOL GetWindowTitle(char* title);`  
`void SetWindowTitle (const char* title);`

#### Remarks

Gets/sets the title of view windows automatically created by `SapView`. This is the case when you specify `hWnd` equal to `SapHwndAutomatic` in the `SapView` constructor, or if you use the `SetWindow` method to accomplish the same goal.

When using `GetWindowTitle`, make certain that the destination string can hold at least 128 characters.

You can only call these methods after the Create method.

#### Demo/Example Usage

Not available

---

### SapView::HasRange

`BOOL HasRange();`

#### Remarks

Checks if the view resource can show a subrange of buffer data bits. This is useful when the number of significant bits is less than the number of bit per pixel for the buffer, for example, data coming from a 10-bit camera stored in a 16-bit buffer.

Use the `SetRange` method to set the viewing range value.

You can only call this method after the Create method.

#### Demo/Example Usage

Not available

---

### SapView::Hide

```
void Hide();
```

#### Remarks

Hides the currently displayed buffer. This is only relevant when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

#### Demo/Example Usage

Not available

---

### SapView::Init

```
void Init();
```

#### Remarks

Initializes the view index from the current buffer index. The Create method automatically performs this action. This ensures correct synchronization between the view and buffer index. Therefore, you normally do not have to call Init.

However, if you use the ShowNext method, but do not call it for every frame, then the view index will not be synchronized with the buffer index. In such a case you must call Init explicitly to restore synchronization.

#### Demo/Example Usage

Not available

---

### SapView::IsEmpty, SapView::SetIsEmpty

```
BOOL IsEmpty();
```

```
void SetIsEmpty(BOOL isEmpty);
```

#### Remarks

Gets/sets the 'auto-empty' mechanism, used for synchronizing the transfer and view tasks in the application program.

By default, the SapTransfer Class automatically calls SapBuffer::SetState(SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the view task can even show it. Although this is usually not a critical issue, there are cases in which you need to avoid this.

In order to correctly synchronize the transfer and view tasks, you must first disable this behavior by calling SapTransfer::SetIsEmpty(FALSE). Then call SapView::SetIsEmpty(TRUE) to enable it in this class instead.

As a result, no images will be acquired in the current buffer as long as buffer contents have not been shown following calls to the Show or ShowNext methods. The buffer state is then reset before the application callback method, if any, is called.

The initial value for this attribute is FALSE, unless you construct this object using an existing SapView object.

#### Demo/Example Usage

Not available

---

### SapView::IsCapabilityValid

```
BOOL IsCapabilityValid(int cap);
```

#### Parameters

*cap* Low-level Sapera C library capability to check

#### Return Value

Returns TRUE if the capability is supported, FALSE otherwise

#### Remarks

Checks for the availability of a low-level Sapera C library capability for the view module. Call this method before GetCapability to avoid invalid or not available capability errors.

Note that this method is rarely needed. The SapView class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

#### Demo/Example Usage

Not available

---

## SapView::IsParameterValid

BOOL **IsParameterValid**(int *param*);

### Parameters

*param* Low-level Sapera C library parameter to check

### Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

### Remarks

Checks for the availability of a low-level Sapera C library parameter for the view module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapView class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

### Demo/Example Usage

Not available

---

## SapView::OnHScroll

void **OnHScroll**(int *hPosition*);

### Parameters

*hPosition* New horizontal scrolling position

### Remarks

Call this method from your application WM\_HSCROLL message handler to adjust the horizontal scrolling position.

### Demo/Example Usage

Not available

---

## SapView::OnMove

void **OnMove**();

### Remarks

Call this method from your application WM\_MOVE message handler to adjust the position of the viewing window.

### Demo/Example Usage

Not available

---

## SapView::OnPaint

void **OnPaint**();

### Remarks

Call this method from your application WM\_PAINT message handler to show the last displayed buffer again.

### Demo/Example Usage

Not available

---

## SapView::OnSize

void **OnSize**();

### Remarks

Call this method from your application WM\_SIZE message handler to adjust the size of the viewing window

### Demo/Example Usage

File Load MFC

---

## SapView::OnVScroll

void **OnVScroll**(int *vPosition*);

## Parameters

*vPosition* New vertical scrolling position

## Remarks

Call this method from your application WM\_VSCROLL message handler to adjusts the vertical scrolling position.

## Demo/Example Usage

Not available

---

## SapView::ReleaseDC

**BOOL ReleaseDC();**

## Return Value

Returns TRUE if successful, FALSE otherwise

## Remarks

Releases the Windows Device Context corresponding to the current view area.

If the current SapView object does not use the system display (see SapDisplay::GetType), then this method releases the Windows Device Context corresponding to the entire display instead.

## Demo/Example Usage

Not available

---

## SapView::SetCallbackInfo

**BOOL SetCallbackInfo(SapViewCallback *pCallback*, void\* *pContext* = NULL);**

## Remarks

Sets the application callback method to call after showing each buffer and the associated context.

You can only call SetCallbackInfo before the Create method. See the SapView constructor for more details.

## Demo/Example Usage

Not available

---

## SapView::Show

**void Show();**  
**void Show(int *index*);**

## Parameters

*index* Index of the buffer resource to show

## Remarks

If the *index* is specified, the corresponding buffer in the SapBuffer object is shown through the internal view thread. Otherwise, the current buffer is shown.

If the SapBuffer object has only one buffer resource, that is, if the SapBuffer::GetCount method returns 1, then *index* is ignored, and is assumed to be 0.

If you want to display data acquired in realtime in a buffer through the SapTransfer Class, simply call the Show method within the SapTransfer callback function in application code.

The SapView Class will then show as many frames as possible without slowing down the transfer process. This means that some buffers will be skipped if the view task is too slow to keep up with the acquisition. If you need all frames to be shown, call the ShowNext method instead.

For multiformat buffers (for example, SapFormatRGB888\_MONO8 or RGB161616\_MONO16) the SapBuffer::SetPage function determines which part (RGB or Mono) of the buffer is displayed. There is no need to call SapView::Destroy or Create when switching buffer pages.

## Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Sequential Grab Demo, Grab Demo, Color Split Example, File Load Console Example, GigE Auto-White Balance, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab Lut Example, Grab MFC Example

---

## SapView::ShowNext

```
void ShowNext();
```

#### Remarks

This method shows the next undisplayed buffer in the SapBuffer object through the internal view thread. If you want to display data acquired in real-time into a buffer through the SapTransfer Class, simply call the ShowNext method within the SapTransfer callback method.

The SapView Class will then show all the frames and possibly slow down the transfer process if needed. If the view task is fast enough to keep-up with the incoming frames, ShowNext behaves exactly the same way as Show. Otherwise, the transfer process must be slowed down to give the SapView object the chance to show every frame.

If you want to show as many frames as possible without affecting the transfer process, use the Show method instead.

#### Demo/Example Usage

Not available

# SapViewCallbackInfo

The SapViewCallbackInfo Class acts as a container for storing all arguments to the callback function for SapView.

```
#include <SapClassBasic.h>
```

## SapViewCallbackInfo Class Members

### Construction

[SapViewCallbackInfo](#) Class constructor

### Attributes

[GetView](#) Gets the SapView object associated with the view callback function

[GetContext](#) Gets the application context associated with the SapView callback function

## SapViewCallbackInfo Member Functions

The following are members of the SapViewCallbackInfo Class.

### SapViewCallbackInfo::SapViewCallbackInfo

**SapViewCallbackInfo**(SapView\* *pView*, void\* *context*);

#### Parameters

*pView* SapView object that calls the callback function

*context* Pointer to the application context

#### Remarks

SapView objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapView uses this class when notifying the application that a buffer has been shown.

#### Demo/Example Usage

Color Conversion Demo

### SapViewCallbackInfo::GetContext

void\* **GetContext()**;

#### Remarks

Gets the context information associated with the application callback function. See the SapView constructor for more details.

#### Demo/Example Usage

Color Conversion Demo

### SapViewCallbackInfo::GetView

SapView\* **GetView()**;

#### Remarks

Gets the SapView object associated with the view callback function. See the SapView constructor for more details.

#### Demo/Example Usage

Not available

# SapXferCallbackInfo

The SapXferCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapTransfer Class.

```
#include <SapClassBasic.h>
```

## SapXferCallbackInfo Class Members

### Construction

SapXferCallbackInfo Class constructor

### Attributes

<u>GetTransfer</u>	Gets the SapTransfer object associated with transfer events
<u>GetContext</u>	Gets the application context associated with transfer events
<u>GetCustomData</u>	Gets the data associated with a custom transfer event
<u>GetCustomSize</u>	Gets the size of the custom data returned by GetCustomData
<u>GetEventType</u>	Gets the transfer events that triggered the call to the application callback
<u>GetEventCount</u>	Gets the current count of transfer events
<u>GetEventInfo</u>	Gets the low-level Sapera handle of the event info resource
<u>GetGenericParam0</u>	Gets generic parameters supported by some events
<u>GetGenericParam1</u>	
<u>GetGenericParam2</u>	
<u>GetGenericParam3</u>	
<u>IsTrash</u>	Checks if the current transfer event is associated with a trash buffer
<u>GetPairIndex</u>	Gets the index of the transfer pair associated with the current transfer event
<u>GetAuxiliaryTimestamp</u>	Gets the auxiliary timestamp associated with transfer events.
<u>GetHostTimestamp</u>	Gets the host timestamp associated with transfer events.

## SapXferCallbackInfo Member Functions

The following are members of the SapXferCallbackInfo Class.

### SapXferCallbackInfo::SapXferCallbackInfo

**SapXferCallbackInfo(**

```
    SapTransfer* pXfer,  
    void* context,  
    SapTransfer::EventType eventType,  
    int eventCount,  
    BOOL isTrash,  
    int pairIndex  
)
```

```
SapXferCallbackInfo(
```

```
    SapTransfer *pXfer,  
    void *context,  
    COREVENTINFO eventInfo,  
    BOOL isTrash,  
    int pairIndex)
```

### Parameters

<i>pXfer</i>	SapTransfer object that calls the callback function
<i>context</i>	Pointer to the application context
<i>eventType</i>	Combination of transfer events. See the SapXferPair::GetEventType method for a list of possible values.
<i>eventCount</i>	Current transfer event count
<i>eventInfo</i>	Low-level Sapera handle of the event info resource

*isTrash* TRUE if the current transfer event is associated with a trash buffer, FALSE otherwise

*pairIndex* Transfer pair index for the current transfer event

### Remarks

SapTransfer objects create an instance of this class before each call to the transfer callback method in order to combine all function arguments into one container.

The *pContext* argument takes the value specified in the SapTransfer Class constructor; *eventType* identifies the combination of events that triggered the call to the callback function; and *eventCount* increments by one at each call, starting at 1. The counter is reinitialized each time you call the SapTransfer::Snap or SapTransfer::Grab method.

By default, the event count is associated with the destination node for the transfer. This usually corresponds to a buffer object, and each buffer resource in the object gets its own count. The SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource method allows the count to be associated with the source node instead. Since this usually corresponds to an acquisition object, the count then increases at every acquired frame.

The *pairIndex* argument identifies the transfer pair associated with the callback. *isTrash* is only relevant when the destination node for this pair is a SapBufferWithTrash object.

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

## SapXferCallbackInfo::GetAuxiliaryTimestamp

BOOL GetAuxiliaryTimestamp(UINT64 \*auxTimestamp);

### Parameters

*auxTimestamp* Address of a pointer to receive the auxiliary timestamp

### Remarks

Gets the auxiliary timestamp associated with transfer events. Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

### Demo/Example Usage

Not available

---

## SapXferCallbackInfo::GetContext

void\* GetContext();

### Remarks

Gets the application context associated with transfer events. See the SapTransfer constructor for more details.

### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

---

## SapXferCallbackInfo::GetCustomData

BOOL GetCustomData(void\*\* customData);

### Parameters

*customData* Address of a pointer to receive the address to the data buffer

### Remarks

Gets the address of a buffer containing the data associated with a custom transfer event. You must not free the buffer after you are finished using it.

This functionality is usually not supported, except for special versions of certain acquisition devices. See the device User's Manual for more information on availability.

### Example

```
void MyCallback(SapXferCallbackInfo* pInfo)
{
```

```

    // Retrieve the data buffer
    void* pCustomData;
    pInfo->GetCustomData(&pCustomData);

    // Use the data buffer
    //...
}

```

### Demo/Example Usage

Not available

## SapXferCallbackInfo::GetCustomSize

**BOOL GetCustomSize(int\* customSize);**

### Parameters

*customSize* Address of an integer to return the value

### Remarks

Gets the size of the custom data returned by the GetCustomData method.

### Demo/Example Usage

Not available

## SapXferCallbackInfo::GetEventCount

**int GetEventCount();**  
**BOOL GetEventCount(int \*eventCount);**

### Parameters

*eventCount* Pointer to the variable to hold the event count

### Remarks

Gets the current count of transfer events. The initial value is 1 and increments after every call to the transfer callback function. The counter is reinitialized each time you call the SapTransfer::Snap or SapTransfer::Grab methods.

By default, the event count is associated with the destination node for the transfer. This usually corresponds to a buffer object, and each buffer resource in the object gets its own count. The SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource method allows the count to be associated with the source node instead. Since this usually corresponds to an acquisition object, the count then increases at every acquired frame.

### Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example, Grab MFC Example

## SapXferCallbackInfo::GetEventInfo

**COREVENTINFO GetEventInfo();**

### Remarks

Gets the low-level Sapera handle of the event info resource. You should not use this method unless you need a handle to the low-level C API to access some functionality not exposed in the C++ API.

### Demo/Example Usage

Not available

## SapXferCallbackInfo::GetEventType

**SapXferPair::EventType GetEventType();**  
**BOOL GetEventType(SapXferPair::EventType \*eventType);**

### Parameters

*eventType* Pointer to the integer variable to hold the event type

### Remarks

Gets the combination of transfer events that triggered the call to the application callback. See the [SapXferPair::GetEventType](#) method for the list of possible values.

Note that, when the event type is SapXferPair::EndOfLine or SapXferPair::EndOfNLines, the line number for

which the transfer callback function is called is not returned through this function, the corresponding bits are always set to 0.

#### Demo/Example Usage

Not available

---

#### SapXferCallbackInfo::GetGenericParam0

#### SapXferCallbackInfo::GetGenericParam1

#### SapXferCallbackInfo::GetGenericParam2

#### SapXferCallbackInfo::GetGenericParam3

```
BOOL GetGenericParam0(int* paramValue);
BOOL GetGenericParam1(int* paramValue);
BOOL GetGenericParam2(int* paramValue);
BOOL GetGenericParam3(int* paramValue);
```

#### Parameters

*paramValue* Address of an integer where the parameter value is written

#### Remarks

Gets any of the four generic parameters supported by some transfer events. You should use aliases instead when they are available. See the acquisition device User's Manual for a list of transfer events using generic parameters.

#### Demo/Example Usage

Not available

---

#### SapXferCallbackInfo::GetHostTimestamp

```
BOOL GetHostTimestamp(UINT64 *hostTimestamp);
```

#### Parameters

*hostTimestamp* Address of a pointer to receive the host timestamp

#### Remarks

Gets the host timestamp associated with transfer events.

Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

Note that not all acquisition devices support this timestamp. See the device User's Manual for more information on the availability of this value.

#### Demo/Example Usage

Not available

---

#### SapXferCallbackInfo::GetPairIndex

```
int GetPairIndex();
```

#### Remarks

Gets the index of the transfer pair associated with the current transfer event. Use this index together with the SapTransfer::GetPair method to access the corresponding SapXferPair object.

#### Demo/Example Usage

Not available

---

#### SapXferCallbackInfo::GetTransfer

```
SapTransfer* GetTransfer();
```

#### Remarks

Gets the SapTransfer object associated with transfer events. See the SapTransfer constructor for more details.

#### Demo/Example Usage

Not available

---

#### SapXferCallbackInfo::IsTrash

```
BOOL IsTrash();
```

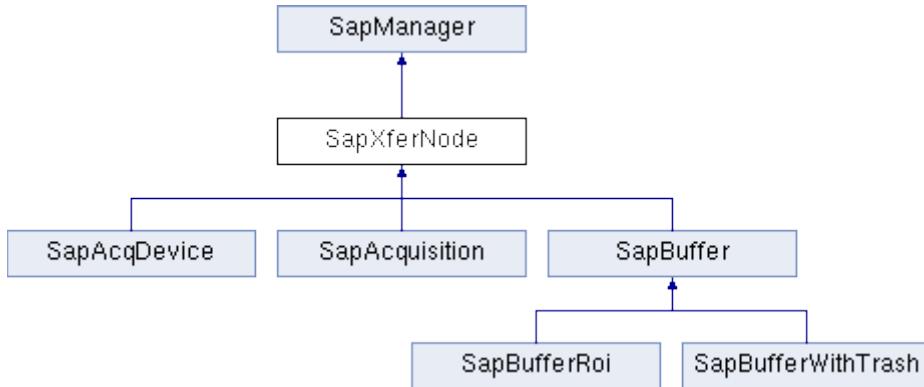
#### Remarks

Checks if the current transfer event is associated with a trash buffer. This is only relevant when the destination node for the current pair is a SapBufferWithTrash object.

#### Demo/Example Usage

Color Conversion Demo, Dual Acquisition Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo

# SapXferNode



The SapXferNode Class implements functionality to manipulate a transfer node object. The SapXferPair Class uses two of these objects to create a transfer pair. The SapTransfer Class then uses this pair to implement a transfer configuration.

You should not instantiate SapXferNode objects directly. Rather, you will use one of its derived classes in your applications. All the following classes are directly derived from SapXferNode: SapAcquisition, SapAcqDevice, SapBuffer, SapBufferRoi, and SapBufferWithTrash.

```
#include <SapClassBasic.h>
```

## SapXferNode Class Members

### Construction

[SapXferNode](#) Class constructor

### Attributes

[GetLocation](#) Gets/sets the location where the transfer node resource is located

[SetLocation](#)

[GetSrcNode](#) Gets/sets the source transfer node object used for compatibility of parameters with other transfer node objects

[SetSrcNode](#)

[GetSrcPort](#) Gets the source port number for this node

[GetXferParams](#) Gets/sets the transfer parameters structure used for compatibility of parameters with other transfer node objects

[SetXferParams](#)

[GetHandle](#) Gets the low-level Sapera handle of the transfer node resource

[GetServer](#) Gets the low-level Sapera handle of the server for the transfer node resource

[GetXferNodeType](#) Gets the type of the current SapXferNode derived object

## SapXferNode Member Properties

The following properties are members of the SapXferNode Class.

### SapXferNode::SapXferNode

**SapXferNode**(*SapLocation loc*);

**SapXferNode**(*SapLocation loc*, *SapXferNode\* pSrcNode*);

**SapXferNode**(*SapLocation loc*, *SapXferParams xferParams*);

### Parameters

*loc* *SapLocation* object specifying the server where the transfer node resource is located and the index of the resource on this server

*pSrcNode* Existing SapXferNode object from which parameters for the current object will be extracted for compatibility of transfer parameters.

`xferParams` Transfer parameters structure used for compatibility of parameters with other transfer node objects.

### Remarks

You should not instantiate SapXferNode objects directly. Rather, use one of the derived classes in the application. All the following classes are derived from SapXferNode: SapAcquisition, SapAcqDevice, SapBuffer, SapBufferRoi, and SapBufferWithTrash.

### Demo/Example Usage

Not available

---

## SapXferNode::GetHandle

`CORHANDLE GetHandle();`

### Remarks

Gets the low-level Sapera handle of the transfer node resource, which you may then use from the low-level Sapera functionality. The exact type of handle depends on the current derived class. The handle is only valid after you call the SapTransfer::Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

### Demo/Example Usage

Not available

---

## SapXferNode::GetLocation, SapXferNode::SetLocation

`SapLocation GetLocation();  
BOOL SetLocation(SapLocation loc);`

### Remarks

Gets/sets the location where the transfer node resource is located. You can only call SetLocation before the SapTransfer::Create method.

### Demo/Example Usage

Not available

---

## SapXferNode::GetServer

`CORSERVER GetServer();`

### Remarks

Gets the low-level Sapera handle of the server on which the transfer node resource is located. You may then use this handle from the low-level Sapera functionality. The handle is only valid after you call the SapTransfer::Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

### Demo/Example Usage

Not available

---

## SapXferNode::GetSrcNode, SapXferNode::SetSrcNode

`SapXferNode* GetSrcNode();  
BOOL SetSrcNode(SapXferNode* pSrcNode, int srcPort = 0);`

### Remarks

Gets/sets the source transfer node object used for compatibility of parameters with other transfer node objects.

For SetSrcNode, the optional `srcPort` argument represents the source port number for the node, which applies only to a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*).

You can only call SetSrcNode before the SapTransfer::Create method.

### Demo/Example Usage

Not available

---

## SapXferNode::GetSrcPort

`int GetSrcPort();`

## Remarks

Gets the source port number for this node. This applies only to a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*).

## Demo/Example Usage

Not available

---

## SapXferNode::GetXferNodeType

SapXferNode::XferNodeType **GetXferNodeType()**;

## Remarks

Gets the type of the current SapXferNode derived object as one of the following values:

SapXferNode::NodeTypeUnknown	Unknown object type
SapXferNode::NodeTypeAcqDevice	Corresponds to a SapAcqDevice object
SapXferNode::NodeTypeAcquisition	Corresponds to a SapAcquisition object
SapXferNode::NodeTypeBuffer	Corresponds to a SapBuffer object
SapXferNode::NodeTypeDsp	Corresponds to a SapDsp object

Note that the following node types apply only to older products.: NodeTypeCab, NodeTypeDsp, and NodeTypePixPro. See the *Sapera LT ++ Legacy Classes Reference Manual* for related classes.

## Demo/Example Usage

Not available

---

## SapXferNode::GetXferParams, SapXferNode::SetXferParams

SapXferParams **GetXferParams**(int *portIndex* = 0);

BOOL **SetXferParams**(SapXferParams *xferParams*, int *portIndex* = 0);

## Remarks

Gets/sets the transfer parameters structure used for compatibility of parameters with other transfer node objects. The optional *portIndex* argument represents the port number for the node, which applies only to a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*).

You can only call this function before the SapTransfer::Create method.

## Demo/Example Usage

Not available

# SapXferPair

The SapXferPair Class describes a pair of source and destination transfer nodes.

If your application uses the SapTransfer Class directly, then you must add transfer pairs yourself before calling the SapTransfer::Create method. If your application uses one of the Specialized Transfer Classes instead, then the class constructor adds all the pairs automatically.

```
#include <SapClassBasic.h>
```

## SapXferPair Class Members

### Construction

[SapXferPair](#) Class constructor

### Attributes

[GetSrc](#)

Gets the source node for this pair

[GetSrcPort](#)

Gets the source node port number for this pair (CAB only)

[GetSrcIndex](#)

Gets the source node resource index for this pair

[GetDst](#)

Gets the destination node for this pair

[GetDstPort](#)

Gets the destination node port number for this pair (CAB only)

[IsRegCallback](#)

Checks if a callback method will be registered for this transfer pair

[SetCallbackInfo](#)

Sets the application callback method for transfer events and the associated context

[SetTrashCallbackInfo](#)

Sets the application callback method for transfer events in the trash buffer and the associated context

[GetCallback](#)

Gets the current application callback function for transfer events

[GetTrashCallback](#)

Gets the current application callback function for transfer events in the trash buffer

[GetContext](#)

Gets the application context associated with transfer events

[GetTrashContext](#)

Gets the application context associated with transfer events in the trash buffer

[GetEventType](#)

Gets/sets the combination of registered transfer event types

[SetEventType](#)

Gets/sets the location at which the count of transfer events increases

[GetEventCountSource](#)

Gets/sets the location at which the count of transfer events increases

[SetEventCountSource](#)

[GetCycleMode](#)

Gets/sets the buffer cycling mode when the destination node is a SapBuffer object

[SetCycleMode](#)

Gets/sets the flipping (that is, mirroring) mode for transferred images

[GetFlipMode](#)

[SetFlipMode](#)

[GetCounterStampTimeBase](#)

Gets/sets the base units used for counter stamps of destination buffers.

[SetCounterStampTimeBase](#)

[GetFramesPerCallback](#)

Gets/sets the number of transferred frames that trigger a notification from the acquisition device to user level code

[SetFramesPerCallback](#)

[GetFramesOnBoard](#)

Gets/sets the number of internal buffers to be used on a source acquisition node

[SetFramesOnBoard](#)

## SapXferPair Member Functions

The following are members of the SapXferPair Class.

### SapXferPair::SapXferPair

[SapXferPair\(](#)

SapXferNode\* pSrc,

```

SapXferNode* pDst,
BOOL regCallback = TRUE
);
SapXferPair(
    SapXferNode* pSrc,
    int srcPort,
    SapXferNode* pDst,
    int dstPort,
    BOOL regCallback = TRUE
);

```

#### Parameters

<i>pSrc</i>	Source node for this pair
<i>pDst</i>	Destination node for this pair
<i>regCallback</i>	If TRUE, a callback method will be registered for this pair
<i>srcPort</i>	Source node port number or resource index for this pair
<i>dstPort</i>	Destination node port number for this pair

#### Remarks

The SapXferPair constructor defines a transfer pair as a combination of one source and one destination node, both of which are objects derived from the SapXferNode Class. This means they can be objects of one of the following classes: SapAcquisition, SapAcqDevice, SapBuffer, SapBufferRoi, and SapBufferWithTrash.

If *regCallback* is TRUE, then the SapTransfer object containing this pair automatically registers a callback function when you call the SapTransfer::Create method. By default, the callback function and application context specified in the SapTransfer::SapTransfer constructor are used. You may override these for a specific pair by calling the SetCallbackInfo method in this class.

If *regCallback* is FALSE, then no callback function is registered. Use this option when you do not need notification of transfer events for this pair.

The *srcPort* argument applies to two cases only. If the source node is a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*), then it identifies the source data port number. If the source node is a SapBuffer object, then it identifies the source buffer resource index. In all other cases, *srcPort* is ignored.

The *dstPort* argument applies only in one case. If the destination node is a SapCab or SapPixPro object, then it identifies the destination data port number. In all other cases, *dstPort* is ignored.

#### Demo/Example Usage

Not available

### **SapXferPair::GetCallback, SapXferPair::GetTrashCallback**

```

SapXferCallback GetCallback();
SapXferCallback GetTrashCallback();

```

#### Remarks

Gets the current application callback function for transfer events for the current pair. If NULL, then the callback function specified in the associated SapTransfer object applies to the pair. You can also use GetTrashCallback to retrieve the same information for the trash buffer (if any).

The initial value for this attribute is NULL.

#### Demo/Example Usage

Not available

### **SapXferPair::GetContext, SapXferPair::GetTrashContext**

```

void* GetContext();
void* GetTrashContext();

```

#### Remarks

Gets the application context associated with transfer events for the current pair. If NULL, then the context specified in the associated SapTransfer object applies to the pair. You can also use GetTrashContext to retrieve the same information for the trash buffer (if any).

The initial value for this attribute is NULL.

#### Demo/Example Usage

---

**SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase**

SapXferPair::CounterStampTimeBase **GetCounterStampTimeBase()**  
**BOOL SetCounterStampTimeBase** (SapXferPair:: CounterStampTimeBase *counterStampTimeBase*);

**Parameters**

<i>counterStampTimeBase</i>	Counter stamp units. Can be one of the following:
SapXferPair::CounterStampMicroSec	Microseconds
SapXferPair::CounterStampMilliSec	Milliseconds
SapXferPair::CounterStampLine	Line valid or horizontal sync signal
SapXferPair::CounterStampLineTrigger	External line trigger of shaft encoder pulse
SapXferPair::CounterStampFrame	Frame valid or vertical sync signal
SapXferPair::CounterStampExtFrameTrigger	External frame trigger signal
SapXferPair::CounterStampShaftEncoder	Shaft encoder input (before drop or/and multiply factors).

**Remarks**

Gets/sets the base units used for counter stamps of destination buffers. Individual values have no meaning by themselves; however, subtracting counter stamp values for two buffer resources gives the amount of time (or a number of signal occurrences) elapsed between a common reference point for their respective data transfers.

See the SapTransfer::GetCounterStampInfo method to find out which common reference point is used for the current transfer pair.

The initial value for this attribute is CounterStampMicroSec.

Depending on the current transfer device, you may be allowed to call SetCounterStampTimeBase at any time. However, you should still call this method before calling SapTransfer::Create or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

Note, for frame grabbers that support the acquisition timestamp (see SapAcquisition::IsTimeStampAvailable), the acquisition timestamp is used; the timestamp base is set using the [SapAcquisition::SetTimeStampBase](#) function.

Note also that this function is not available for Genie cameras; use the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue function with the 'TimestampCounter' feature.

**Demo/Example Usage**

Sequential Grab Demo

---

**SapXferPair::GetCycleMode, SapXferPair::SetCycleMode**

SapXferPair::CycleMode **GetCycleMode()**  
**BOOL SetCycleMode** (SapXferPair::CycleMode *cycleMode*);

**Parameters**

<i>cycleMode</i>	The available buffer cycling modes differ by the way in which they specify which buffer resource gets the next data transfer.
	The empty state refers to the case in which buffer data has been completely processed and may be overwritten. It is set by application code as soon as it has finished processing buffer data.
	The full state refers to the case in which buffer data has not been processed since its latest data transfer. It is set by the transfer device as soon as a data transfer has completed.
	The current buffer is the one in which the latest data transfer occurred.
	The next buffer is the one immediately after the current buffer, with wraparound to the first buffer at the end of the list.
	The trash buffer is defined as the last buffer in the list for the WithTrash modes only. Its state is always considered to be empty by the transfer device.
	The cycling mode can be one of the following values:
SapXferPair::CycleUnknown	Unknown cycle mode.

SapXferPair::CycleAsynchronous	Always transfer to the next buffer, regardless of its state.
SapXferPair::CycleSynchronous	The first transfer always occurs in the currently selected buffer. From then on, if next buffer is empty, then transfer to next buffer; otherwise, transfer to current buffer.
SapXferPair::CycleWithTrash	If next buffer is empty, then transfer to the next buffer; otherwise, transfer to the trash buffer. Repeat transferring to the trash buffer as long as the next buffer is full.
SapXferPair::CycleOff	Always transfer to the current buffer.
SapXferPair::CycleNextEmpty	If next buffer is empty, then transfer to next buffer; otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to current buffer.
SapXferPair::CycleNextWithTrash	If next buffer is empty, then transfer to next buffer; otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to trash buffer. Repeat transferring to the trash buffer as long as there is no empty buffer in the list.

### Remarks

Gets/sets the buffer cycling mode when the destination node is a SapBuffer object.

The initial value for this attribute is CycleUnknown. This means that the associated SapTransfer Class uses a CycleWithTrash cycle mode for a SapBufferWithTrash object; otherwise, it uses CycleAsynchronous. Call SetCycleMode if you want to override this value for the current transfer pair.

Depending on the current transfer device, you may be allowed to call SetCycleMode at any time. However, you should still call this method before calling SapTransfer::Create or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

The current transfer device may not support all possible cycling modes. You can use the SapTransfer::IsCycleModeAvailable method to check if the desired mode is supported.

### Demo/Example Usage

GigE Camera Demo

## SapXferPair::GetDst

SapXferNode\* **GetDst()**;

### Remarks

Gets the destination node for this pair as an object derived from the SapXferNode Class. See the SapXferNode constructor for a list of derived classes.

### Demo/Example Usage

Not available

## SapXferPair::GetDstPort

int **GetDstPort()**;

### Remarks

Gets the destination node port number for this pair. This applies only when the node is a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*).

### Demo/Example Usage

Not available

## SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource

SapXferPair::EventCountSource **GetEventCountSource()**;  
BOOL **SetEventCountSource**(SapXferPair::EventCountSource *eventCountSource*);

### Parameters

<i>eventCountSource</i>	Resource type where the transfer event count increases. Can be one of the following:	
SapXferPair::EventCountNone	No event count available	
SapXferPair::EventCountDst	Count is linked to the destination node	
SapXferPair::EventCountSrc	Count is linked to the source node	

## Remarks

Gets/sets the resource type at which the count of transfer events increases. The destination node normally corresponds to a buffer object, so that each buffer resource in the object gets its own count. The source node usually corresponds to an acquisition object, so that the count increases at every acquired frame.

The initial value for this attribute is EventCountSourceDst.

Depending on the current transfer device, you may be allowed to call SetEventCountSource at any time. However, you should still call this method before calling SapTransfer::Create or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

## Demo/Example Usage

Not available

---

### SapXferPair::GetEventType, SapXferPair::SetEventType

SapXferPair::EventType **GetEventType()**;  
BOOL **SetEventType**(SapXferPair::EventType *eventType*);

## Parameters

*eventType* Transfer events for which the application callback function will be called. One or more of the following values may be combined together using bitwise a OR operation:

SapXferPair::EventNone	No events
SapXferPair::EventStartOfField	Start of field (odd or even)
SapXferPair::EventStartOfOdd	Start of odd field
SapXferPair::EventStartOfEven	Start of even field
SapXferPair::EventStartOfFrame	Start of frame
SapXferPair::EventEndOfField	End of field (odd or even)
SapXferPair::EventEndOfOdd	End of odd field
SapXferPair::EventEndOfEven	End of even field
SapXferPair::EventEndOfFrame	End of frame
SapXferPair::EventEndOfLine	After a specific line number <i>eventType</i> = SapXferPair::EventEndOfLine   <i>lineNum</i> Note that <i>lineNum</i> only applies to SetEventType, its value is not returned when calling GetEventType, the corresponding bits are set to 0.
SapXferPair::EventEndOfNLines	After a specific number of lines (linescan cameras only) <i>eventType</i> = SapXferPair::EventEndOfNLines   <i>numLines</i> Note that <i>numLines</i> only applies to SetEventType, its value is not returned when calling GetEventType, the corresponding bits are set to 0.
SapXferPair::EventEndOfTransfer	End of transfer, that is, after all frames have been transferred following calls to SapTransfer::Snap or SapTransfer::Grab/SapTransfer::Freeze.
SapXferPair::EventLineUnderrun	The number of active pixels per line received from a video source is less than it should be.
SapXferPair::EventFieldUnderrun	The number of active lines per field received from a video source is less than it should be.

## Remarks

The initial value for this attribute is EventEndOfFrame.

You can only call SetEventType before calling SapTransfer::Create or SapTransfer::Connect if you use the SapTransfer::SetAutoConnect method to turn off the auto-connect mechanism.

## Demo/Example Usage

Not available

---

### SapXferPair::GetFlipMode, SapXferPair::SetFlipMode

SapXferPair::FlipMode **GetFlipMode()**;  
BOOL **SetFlipMode**(SapXferPair::FlipMode *flipMode*);

## Parameters

<i>flipMode</i>	SapXferPair::FlipNone	No flipping
	SapXferPair::FlipHorizontal	Transferred images are flipped horizontally
	SapXferPair::FlipVertical	Transferred images are flipped vertically

## Remarks

Gets/sets the flipping (that is, mirroring) mode for transferred images for the current transfer pair.

The initial value for this attribute is FlipNone.

Depending on the current transfer device, you may be allowed to call SetFlipMode at any time. However, you should still call this method before calling SapTransfer::Create or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

## Demo/Example Usage

Not available

---

## SapXferPair::GetFramesOnBoard, SapXferPair::SetFramesOnBoard

```
int GetFramesOnBoard();
BOOL SetFramesOnBoard(int numFrames);
```

## Remarks

Gets/sets the number of internal buffers to be used on a source acquisition node.

The value returned by GetFramesOnBoard is only valid after calling the SapTransfer::Create function (or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism). If this value is equal to 0, it means that the acquisition hardware has no internal buffers.

Since the acquisition hardware usually has a default number of internal buffers which is appropriate in most cases, there is usually no need to call SetFramesOnBoard. If you do, however, you should always use the following sequence:

```
SetFramesOnBoard(numFrames);
SapTransfer::Create();
newNumFrames = GetFramesOnBoard();
```

If the value returned by GetFramesOnBoard is less than the original *numFrames*, it means that there is not enough internal memory for all the buffers, and it indicates the number of buffers which have in fact been allocated.

## Demo/Example Usage

Sequential Grab Demo

---

## SapXferPair::GetFramesPerCallback, SapXferPair::SetFramesPerCallback

```
int GetFramesPerCallback();
BOOL SetFramesPerCallback(int numFrames);
```

## Remarks

Gets/sets the number of transferred frames that trigger a notification from the acquisition device to user level code.

This is particularly useful when the acquisition device has a high frame rate. In this case, the large amount of communication between the device and the host can result in significant CPU overhead, which may negatively affect performance. In this case, call the SetFramesPerCallback method with a value larger than 1 to reduce this overhead.

It is important to note that the number of frames per callback is an internal optimization for the current transfer pair in the SapTransfer class only, with the only noticeable effect being improved performance in some cases. This means that the application callback function will still be called for every acquired frame.

The default value for this attribute is 1.

You can only call SetFramesPerCallback before calling the SapTransfer::Create method or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

## Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

---

## SapXferPair::GetSrc

```
SapXferNode* GetSrc();
```

#### Remarks

Gets the source node for this pair as an object derived from the SapXferPair Class. See the SapXferPair constructor for a list of derived classes.

#### Demo/Example Usage

Not available

---

### SapXferPair::GetSrcIndex

```
int GetSrcIndex();
```

#### Remarks

Gets the source node resource index for this pair. This applies only when the node is a SapBuffer object.

#### Demo/Example Usage

Not available

---

### SapXferPair::GetSrcPort

```
int GetSrcPort();
```

#### Remarks

Gets the source node port number for this pair. This applies only when the source node is a SapCab or SapPixPro object (see the *Sapera LT ++ Legacy Classes Reference Manual*).

#### Demo/Example Usage

Not available

---

### SapXferPair::IsRegCallback

```
BOOL IsRegCallback();
```

#### Remarks

Returns TRUE if the SapTransfer object containing this pair automatically registers a callback function when you call the SapTransfer::Create method, FALSE otherwise.

The default value for this attribute is TRUE, unless you specify otherwise in the SapXferPair constructor.

#### Demo/Example Usage

Not available

---

### SapXferPair::SetCallbackInfo, SapXferPair::SetTrashCallbackInfo

```
BOOL SetCallbackInfo(SapXferCallback pCallback, void* pContext);
```

```
BOOL SetTrashCallbackInfo(SapXferCallback pCallback, void* pContext);
```

#### Remarks

Sets the application callback method for transfer events and the associated context for the current pair only. This overrides any callback and context specified in the SapTransfer constructor. You can also use SetTrashCallbackInfo if you need a different callback function for the trash buffer (if any).

You can only call SetCallbackInfo or SetTrashCallbackInfo before calling the SapTransfer::Create method or SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

See the SapTransfer constructor for more details.

#### Demo/Example Usage

Not available

# SapXferParams

The SapXferParams Class stores parameters needed by a transfer task managed by the SapTransfer Class.

When building a destination transfer node object, use the transfer parameters from the source node to ensure transfer compatibility between the two. You may do this either by specifying the source SapXferNode object in the destination node constructor, or by directly specifying the appropriate SapXferParams object.

```
#include <SapClassBasic.h>
```

## SapXferParams Class Members

### Construction

[SapXferParams](#) Class constructor

### Attributes

[GetFrameType](#), Gets/sets the field interlacing type in a frame

[SetFrameType](#)

[GetFieldOrder](#), Gets/sets the field order for interlaced frames

[SetFieldOrder](#)

[GetWidth](#), Gets/sets the width (in pixels) of one frame

[SetWidth](#)

[GetHeight](#), Gets/sets the height (in lines) of one frame

[SetHeight](#)

[GetFormat](#), Gets/sets the data format of the transferred data

[SetFormat](#)

[GetPixelDepth](#), Gets/sets the number of significant bits of the transferred data

[SetPixelDepth](#)

[GetPixelShift](#), Gets/sets the difference between the pixel depth and the number of bits in the data format (obsolete)

[SetPixelShift](#)

[GetParameters](#), Gets/sets all the parameters in one operation

[SetParameters](#)

## SapXferParams Member Functions

The following are members of the SapXferParams Class.

### SapXferParams::SapXferParams

**SapXferParams();**

### Remarks

The SapXferParams constructor initializes its members to default (but probably incorrect) values. Use the other methods in this class to properly set these values.

### Demo/Example Usage

Not available

### SapXferParams::GetFieldOrder, SapXferParams::SetFieldOrder

```
SapXferParams::FieldOrder GetFieldOrder();  
void SetFieldOrder(SapXferParams::FieldOrder fieldOrder);
```

### Parameters

*fieldOrder* Field order can be one of the following values:

SapXferParams::FieldOrderOddEven The odd field is transferred before the even field

SapXferParams::FieldOrderEvenOdd The even field is transferred before the odd field

SapXferParams::FieldOrderNext

The next field is transferred, whether it is odd or even

### Remarks

Gets/sets the field order for interlaced frames. Does not apply for progressive video.

### Demo/Example Usage

Not available

---

## SapXferParams::GetFormat, SapXferParams::SetFormat

```
SapFormat GetFormat();  
void SetFormat(SapFormat format);
```

### Remarks

Gets/sets the pixel format of the transferred data. See the SapBuffer::SapBuffer constructor for possible values for *format*.

### Demo/Example Usage

Dual Acquisition Demo

---

## SapXferParams::GetFrameType, SapXferParams::SetFrameType

```
SapXferParams::FrameType GetFrameType();  
void SetFrameType(SapXferParams::FrameType frameType);
```

### Parameters

<i>frameType</i>	Field interlacing can be one of the following values: SapXferParams::FrameInterlaced SapXferParams::FrameProgressive	Video fields are interlaced Video fields are non-interlaced (progressive video)
------------------	--	--

### Remarks

Gets/sets the field interlacing type in a frame.

### Demo/Example Usage

Not available

---

## SapXferParams::GetHeight, SapXferParams::SetHeight

```
int GetHeight();  
void SetHeight(int height);
```

### Remarks

Gets/sets the height (in lines) of one frame

### Demo/Example Usage

Dual Acquisition Demo

---

## SapXferParams::GetParameters, SapXferParams::SetParameters

```
void GetParameters(SapXferParams::FrameType* frameType, SapXferParams::FieldOrder* fieldOrder,  
int* width, int* height, int* format, int* pixelDepth, int* pixelShift);  
void SetParameters (SapXferParams::FrameType frameType, SapXferParams::FieldOrder fieldOrder,  
int width, int height, int format, int pixelDepth, int pixelShift);
```

### Remarks

Gets/sets all the parameters in one operation. See the GetFrameType and GetFieldOrder methods for possible values for *frameType* and *fieldOrder*.

### Demo/Example Usage

Not available

---

## SapXferParams::GetPixelDepth, SapXferParams::SetPixelDepth

```
int GetPixelDepth();  
void SetPixelDepth(int pixelDepth);
```

### Remarks

Gets/sets the number of significant bits of the transferred data. This value is extracted from SapAcquisition objects to determine the number of bits containing actual data. The range of possible values is given by the SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax methods.

#### Demo/Example Usage

Dual Acquisition Demo

---

#### SapXferParams::GetPixelShift, SapXferParams::SetPixelShift

```
int GetPixelShift();  
void SetPixelShift(int pixelShift);
```

#### Remarks

Gets/sets the difference between the pixel depth and the number of bits in the data format for image display purposes.

These methods are **obsolete**, since Saper LT now automatically manages the image display pixel shift using the buffer pixel depth.

#### Demo/Example Usage

Not available

---

#### SapXferParams::GetWidth, SapXferParams::SetWidth

```
int GetWidth();  
void SetWidth(int width);
```

#### Remarks

Gets/sets the width (in pixels) of one frame

#### Demo/Example Usage

Dual Acquisition Demo



# Appendix A: Sapera LT and GenICam

## What is GenICam?

GenICam™ is an international standard that allows a single application programming interface (API) to control any compliant video source, regardless of its vendor, feature set, or interface technology (GigE Vision®, Camera Link®, etc.).

GenICam consists of four modules:

- GenApi: This module defines the format of an XML file that captures the features of a device. GenApi also specifies how to access and control the features. All GenICam-compliant devices must contain an XML file that conforms to this format.
- Standard Features Naming Convention (SFNC): This module standardizes the names of more than 220 commonly used camera features. To comply with GigE Vision, seven of the features are mandatory. The rest are either recommended or optional. Compliance with the naming convention is important for interoperability, as it frees application software from the complexity of situations where vendors call the same feature by different names, such as, 'Brightness' and 'Gain'.
- GenTL: This module defines a software interface for accessing image data from a generic transport layer.
- CLProtocol: This module allows cameras that comply with the Camera Link® standard to be accessed through GenApi. It defines the format of a dynamic-link library that converts a vendor-specific serial protocol to a GenApi interface.

There are two levels of compliance to GenICam:

- GenICam-compliance: where a product either provides or interprets a compliant XML file.
- GenICam TL-compliance: where a product exposes a transport layer compatible with GenTL.

Currently, Teledyne DALSA offers several cameras with GenICam and GigEVision compliance.

## Using Sapera LT with GenICam-compliant Devices

Sapera LT uses the SapAcqDevice and SapFeature classes to access the GenICam features of a device.

A SapAcqDevice object is created for each acquisition device and provides access to the list of features, events and files that are supported on the device. SapAcqDevice also allows the registering and unregistering of callback functions on an event.

### Features

A SapFeature object can be accessed for each feature on the device and provides more detailed information on the actual feature, such as its access mode, minimum and maximum values, enumerations, and so forth, as well as information used for integrating feature access into graphical user interfaces, such as the feature category.

Feature values can be read and written to using the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue functions. To get more information on a feature, retrieve the SapFeature

object for this specific feature using the `SapAcqDevice::GetFeatureInfo` function. See the [Sapera LT ++ – Modifying Camera Features](#) and [Sapera .NET – Modifying Camera Features](#) sections in the [Sapera LT User's Manual](#) for more information and examples on how to access and modify features.

## Selectors

A selector is a fundamental concept of GenICamSFNC; it allows using a single feature to control multiple components of the same feature. For example the Gain feature might have three components: Red, Green, and Blue. The `SapFeature::IsSelector`, `SapFeature::GetSelectedFeatureCount`, `SapFeature::GetSelectedFeatureName`, `SapFeature::GetSelectingFeatureIndex` and corresponding `GetSelecting` functions allow the user to query information about the selector.

## File Transfer

Sapera LT simplifies the transfer of files to and from devices with the `SapAcqDevice::GetFileCount` and `SapAcqDevice::GetFileNameByIndex` functions, which allow for the enumeration of the available device files. The `SapAcqDevice::WriteFile` and `SapAcqDevice::ReadFile` functions are used to transfer the file in and out of the device.

# Notes on the Sapera LT GenICam Implementation

The following functions have GenICam specific notes about their implementation:

- `SapAcqDevice::GetUpdateFeatureMode`, `SapAcqDevice::SetUpdateFeatureMode`: only the `UpdateFeatureAuto` mode is implemented. Therefore, the `SapAcqDevice::UpdateFeaturesFromDevice` and `SapAcqDevice::UpdateFeaturesToDevice` functions are not implemented.
- `SapFeature::GetPollingTime`: GenICam does not provide polling information to the user, therefore this function always returns 0.
- `SapFeature::IsSavedToConfigFile`, `SapFeature::SetSavedToConfigFile`: The `SapFeature` class provides functions to control which features are saved to the device configuration file. In GenICam, this is hardcoded by the device manufacturer in the device description file. Therefore, the `SapFeature::IsSavedToConfigFile`, `SapFeature::SetSavedToConfigFile` has no effect, and returns False when the value is read.
- `SapFeature` class: the retrieval of feature enumeration properties is currently not implemented; only the name and value can be retrieved.

## Events

The `SapAcqDevice` object always provides two events; "`FeatureInfoChanged`" and "`FeatureValueChanged`". These events are related to feature state changes and not the device. Since GenICam does not give information on what changed in the feature, only "`FeatureInfoChanged`" events are generated; the "`FeatureValueChanged`" is never generated.

## Type

GenAPI interface mapping to `SapFeature` types.

GenICam Interface	Sapera Type
IInteger	<code>SapFeature::TypeInt64</code>
IFloat	<code>SapFeature::TypeDouble</code>
IString	<code>SapFeature::TypeString</code>
IEnumeration	<code>SapFeature::TypeEnum</code>
ICommand	<code>SapFeature::TypeBool</code> (write only)
IBoolean	<code>SapFeature::TypeBool</code>

IRegister	SapFeature::TypeArray
ICategory	Not exported; the category is a property of the feature.
ISelector	The selector is a property of the feature regardless of its type.
IPort	This is the interface to the underlying transport technologies; it is not exported to the user.

You can retrieve the type of a feature using the SapFeature::GetType function. If the type returned is TypeArray, reading /writing to this feature must use a SapBuffer or SapLut object.

Currently the ICommand is mapped to a SapFeature::TypeBool. Setting any value will execute that action and return when the action is complete. One limitation of this mapping is that if the action takes more than the Sapera timeout, setting the value might return false even if the action succeeded.

## GigEVision in Sapera LT

The Sapera LT GigEVision implementation is based on the 1.0 specification, but supports devices up to the 1.2 specification with some limitations.

The SapAcqDevice module uses the device manifest table to choose which XML file to download from the camera. Priority is given to the first GenICam device descriptions file using schema 1.1, otherwise schema 1.0 is used.

### Channels

When a SapAcqDevice object is created, the control and messaging channels are in exclusive mode, meaning that only the currently connected application can control the device. The first streaming channel is opened when a SapTransfer object is connected. In addition, the control channel always uses the heartbeat.

Currently, Sapera LT does not support the following GigEVision 1.2 functionality: action command, extended status code, primary application switchover, pending ack, and event data.

### Acquisition

GigEVision defines certain mandatory features that are related to the acquisition. In the current implementation these features are managed by the SapTransfer module and not presented to the user. The SapTransfer::Grab and SapTransfer::Snap functions control the following features: "AcquisitionMode", "AcquisitionFrameCount" and "AcquisitionStart". The SapTransfer::Freeze function controls the "AcquisitionStop". The SapTransfer::Abort function controls the "AcquisitionAbort".

Currently, data can only be sent to one host. Note that some information from the data leader cannot be retrieved by the user, such as Block Id, Width, Height, Offset X and Offset Y, Padding X and Padding Y. In addition, buffers cannot receive images larger than the destination buffer size.

### Streaming

Under Sapera LT, streaming is managed by a SapTransfer module. The concept is based on a pool of buffers. The SapTransfer module fills a buffer with data coming from the device. When all data is received for a buffer, the buffer is delivered through the use of a callback function.

Currently, Sapera LT does not support the following functionality described in the GigEVision 1.2 specification: unconditional streaming, multiple streams and non-streaming devices.

### Cycling

When the first packet of a GigEVision block (leader) is received, it is assigned a buffer by the SapTransfer module to receive the data block. The choice of buffer assigned to a new GigEVision block depends on the cycling mode; the cycling mode is set using the SapXferPair::GetCycleMode, SapXferPair::SetCycleMode function.

The supported cycling modes are:

- SapXferPair::CycleAsynchronous
- SapXferPair::CycleSynchronous
- SapXferPair::CycleWithTrash
- SapXferPair::CycleOff
- SapXferPair::CycleNextEmpty
- SapXferPair::CycleNextWithTrash.

Currently, the trash buffer must be a real buffer, and cannot be of type *SapBuffer::TypeDummy*.

In the event that some packets are lost and not recoverable, the state of the buffer is set as *SapBuffer::StateOverflow*.

## Transfer Callback

The SapTransfer module initiates callback functions based on events. The only supported event types for GigEVision are: *SapXferPair::EventEndOfFrame* and *SapXferPair::EventEndOfTransfer*.

The *SapXferPair::EventEndOfFrame* event informs the user when all data of a GigEVision block is received. At this point, the buffer is controlled by the user until its state is set to empty.

The *SapXferPair::EventEndOfTransfer* event might be sent at the same time as a *SapXferPair::EventEndOfFrame* if the end of the frame also marks the end of a transfer. Currently, the *SapXferPair::EventEndOfTransfer* event is only implemented when using *SapTransfer::Snap* since it is not possible to know if a block is the last of a transfer when the block is received.

To know when a transfer is stopped the *SapTransfer::Wait* function should be used.

## Time Stamp

As opposed to the traditional frame grabber, the timestamp is managed by the acquisition and not the transfer. When a buffer is delivered, *SapBuffer::GetCounterStamp* function returns the 32 least significant bits of the timestamp in the data leader. Control of the timestamp and information about the frequency can be retrieved through features of the *SapAcqDevice*.

Therefore, the *SapXferPair::GetCounterStampTimeBase*, *SapXferPair::SetCounterStampTimeBase* and *SapXferPair::GetEventCountSource*, *SapXferPair::SetEventCountSource* functions are not implemented.

## Variable Frame Length

When acquiring images of variable length, the image buffer is allocated using the maximum expected image height. To determine the actual number of lines in an image, use the *SapBuffer::GetSpaceUsed* function to return how many lines were acquired in the last received buffer. This is necessary to avoid processing lines in the buffer from previous acquisitions that were not overwritten by the current image acquisition (to improve performance, buffers are overwritten but not flushed).

## Payload Type

The Sapera LT only supports the Image payload type; File and Chunk payloads are not supported for the moment.

The Extended Chunk payload is partially supported; it is possible to acquire the data in a buffer, but the specific image and chunk portions of the buffer are not reported.

## Pixel Format

The Sapera LT supports the following GigEVision pixel formats:

Mono8	BayerGR8	BayerRG8	BayerGB8
-------	----------	----------	----------

Mono8Signed Mono10 Mono12 Mono14 Mono16	BayerGR10 BayerGR12 BayerGR16	BayerRG10 BayerRG12 BayerRG16	BayerGB10 BayerGB12 BayerGB16
BayerBG8 BayerBG10 BayerBG12 BayerBG16	BGR8Packed BGRA8Packed BGR12Packed BGR10Packed	BayerBG8 BayerBG10 BayerBG12 BayerBG16	YUV422Packed YUV411Packed YUV422_YUYV_Packed

# Appendix B: Obsolete Classes

The SapBayer and SapGraphics classes have been deprecated and are no longer officially supported. However, the classes will continue to compile. The SapBayer class has been replaced by the SapColorConversion class.

Refer to the Saper LT ++ LT Legacy Classes Reference Manual for full descriptions of all obsolete classes.

# Contact Information



**TELEDYNE DALSA**  
Everywhereyoulook™

## Sales Information

**Visit our web site:**

[www.teledynedalsa.com/imaging](http://www.teledynedalsa.com/imaging)

**Email:**

<mailto:info@teledynedalsa.com>

### Canadian Sales

Teledyne DALSA — Head office  
605 McMurray Road  
Waterloo, Ontario, Canada, N2V 2E9  
Tel: 519 886 6000  
Fax: 519 886 8023

Teledyne DALSA — Montreal office  
880 Rue McCaffrey  
St. Laurent, Quebec, Canada, H4T 2C7  
Tel: (514) 333-1301  
Fax: (514) 333-1388

### USA Sales

Teledyne DALSA — Billerica office  
700 Technology Park Drive  
Billerica, Ma. 01821  
Tel: (978) 670-2000  
Fax: (978) 670-2010

### European Sales

Teledyne DALSA GMBH  
Felix-Wankel-Str. 1  
82152 Krailling, Germany  
Tel: +49 – 89 89 – 54 57 3-80  
Fax: +49 – 89 89 – 54 57 3-46

### Asian Sales

Teledyne DALSA Asia Pacific  
Ikebukuro East 13F  
3-4-3 Higashi Ikebukuro,  
Toshima-ku, Tokyo, Japan  
Tel: +81 3 5960 6353  
Fax: +81 3 5960 6354

Shanghai Industrial Investment Building  
Room G, 20F, 18 North Cao Xi Road,  
Shanghai, China 200030  
Tel: +86-21-64279081  
Fax: +86-21-64699430

---

## Technical Support

Submit any support question or request via our web site:

Technical support form via our web page:  
Support requests for imaging product  
installations,  
Support requests for imaging applications

<http://www.teledynedalsa.com/imaging/support>

Camera support information

Product literature and driver updates

When encountering hardware or software problems, please have the following documents included in your support request:

- The Sapera Log Viewer .txt file
- The PCI Diagnostic PciDiag.txt file
- The Device Manager BoardInfo.txt file



Note, the Sapera Log Viewer and PCI Diagnostic tools are available from the Windows start menu shortcut **Start•All Programs•Teledyne DALSA•Sapera LT•Tools**.

The Device Manager utility is available as part of the driver installation for your Teledyne DALSA device and is available from the Windows start menu shortcut **Start•All Programs•Teledyne DALSA•<Device Name>•Device Manager**.