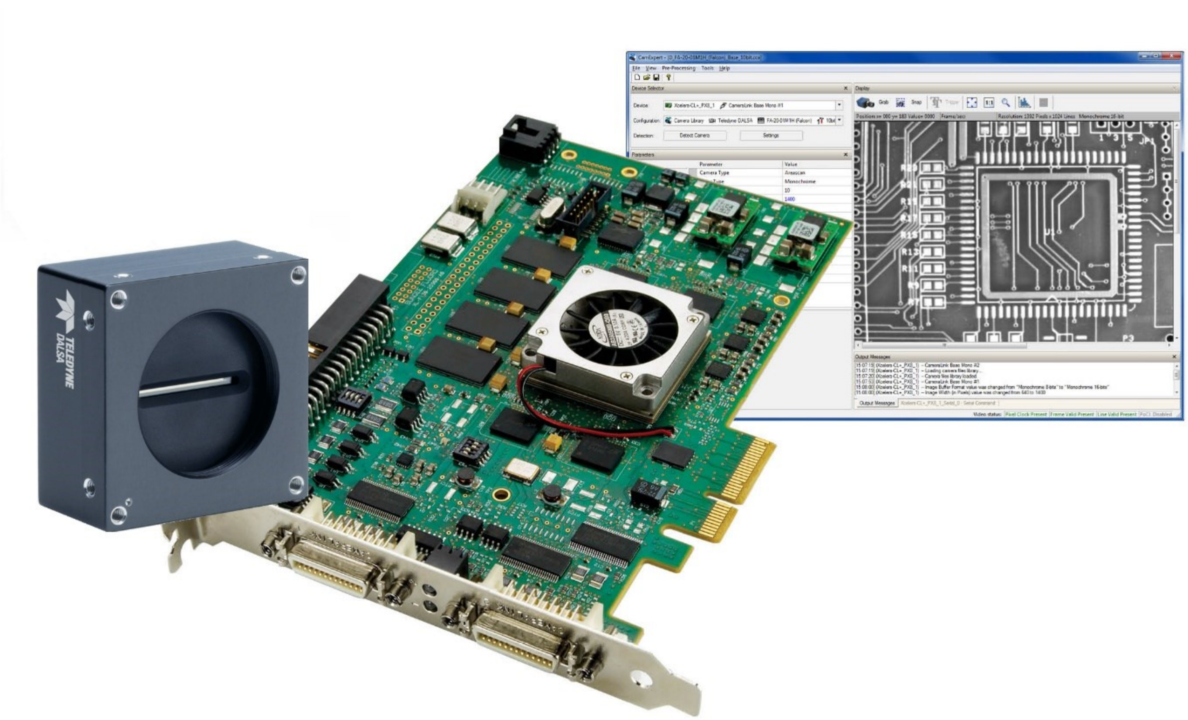
Sapera LT .NET ™ 8.7

Programmer's Manual

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



P/N: OC-SAPM-LTDNP

www.teledynedalsa.com

TELEDYNE DALSA
Everywhereyoulook™

NOTICE

Printed on August 26, 2022

Document Number: OC-SAPM-SPPP0
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, a business unit of Teledyne Digital Imaging Inc., 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 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

Getting Started

About Sapera .NET

Sapera .NET is an Application Programming Interface (API) for Sapera LT. It provides access to the power of the Sapera LT ++ API directly from managed applications written using the .NET Framework within Visual Studio. Sapera .NET provides high-level classes reducing application code complexity while its architecture reflects the underlying low-level Sapera LT architecture. This provides the user with a high-level of flexibility while keeping the simplicity and compactness of object-oriented code.

The Sapera .NET classes contain commonly used Sapera code usable with many imaging applications. These classes are for the user-interface and are hardware independent. They address the basic concepts of imaging applications, such as acquisition, data transfer, processing, and display. Their main purpose is to simplify application code by considerably reducing the number of calls to low-level Sapera LT functions.
Hardware independent classes allow one application to control different Teledyne DALSA devices through the same API. It also guarantees seamless migration to any future Teledyne DALSA hardware product supported by Sapera LT. The modular architecture provides the user with high programming flexibility and readability.

The Sapera .NET interface is composed of three main parts:

- Properties describe the current state of a class. They typically correspond to Get and Set methods in Sapera LT ++.
- Methods are used to invoke control tasks. They correspond to most other methods in Sapera LT ++.
- Events are signals sent to the application program to inform it of conditions occurring within the classes. They typically correspond to callback functions in Sapera LT ++.

There are several advantages to using Sapera .NET classes versus the Sapera LT ++ API:

- Their language-independent interface supports any .NET language, such as C#.
- There is no need for header files and import libraries simplifying the integration of Sapera LT into an application.
- Sapera .NET classes easily integrate with third-party .NET components within the same application.

	If your application requires image processing or GPU optimization, Sapera Processing, a full-featured image processing library, is available as a separate software package. For more information see For more information see http://www.teledynedalsa.com/en/products/imaging/vision-software/.

Notable changes in Sapera LT 8.7

With continuing improvements of its API in mind, and to keep up with the rapid evolution of the technological landscape, here are some of the notable changes included in this new version.

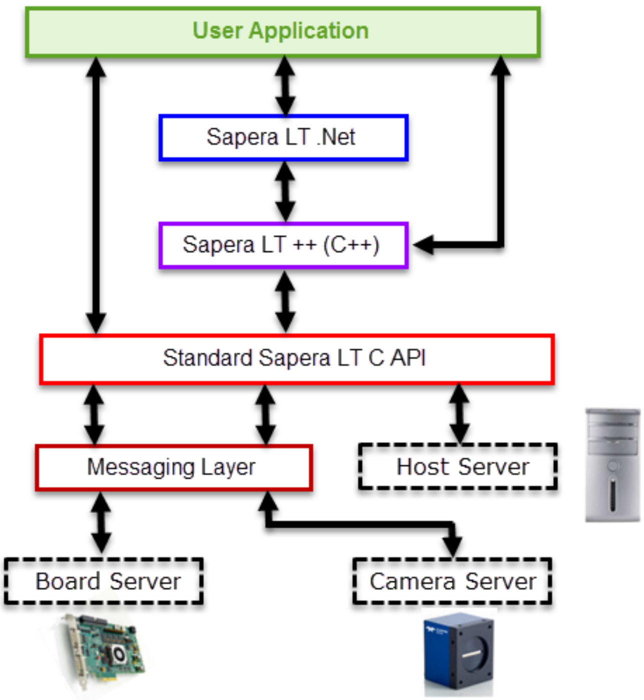
- Sapera LT 8.7 introduces support for .NET (formerly .NET Core) and Visual Studio® 2019 (see Note about .NET Framework and .NET), and includes projects for console examples and GUI demos.
- The new version no longer supports VB.NET, only C# language.
- Version 8.7 marks the end of support for Visual Studio 2010 and 2012.

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?

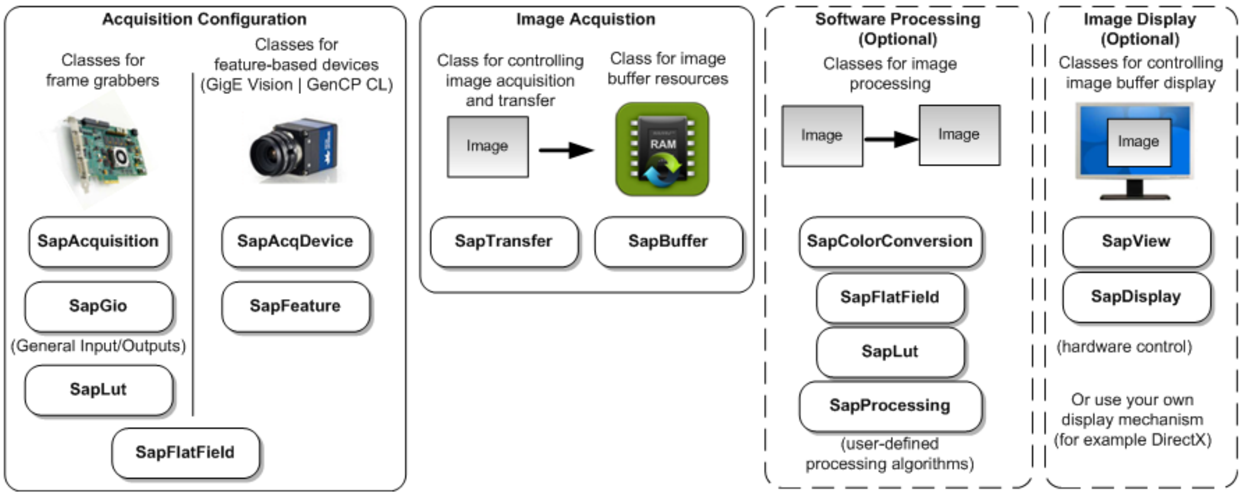
A Sapera Server is an abstract representation of a physical device like a frame grabber, a camera, or a desktop PC. In general, a Teledyne DALSA board is a server. Some processing boards, however, may contain several servers; this is true when using multi-processor boards.

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

Sapera .NET currently supports the following compilers (C# only):

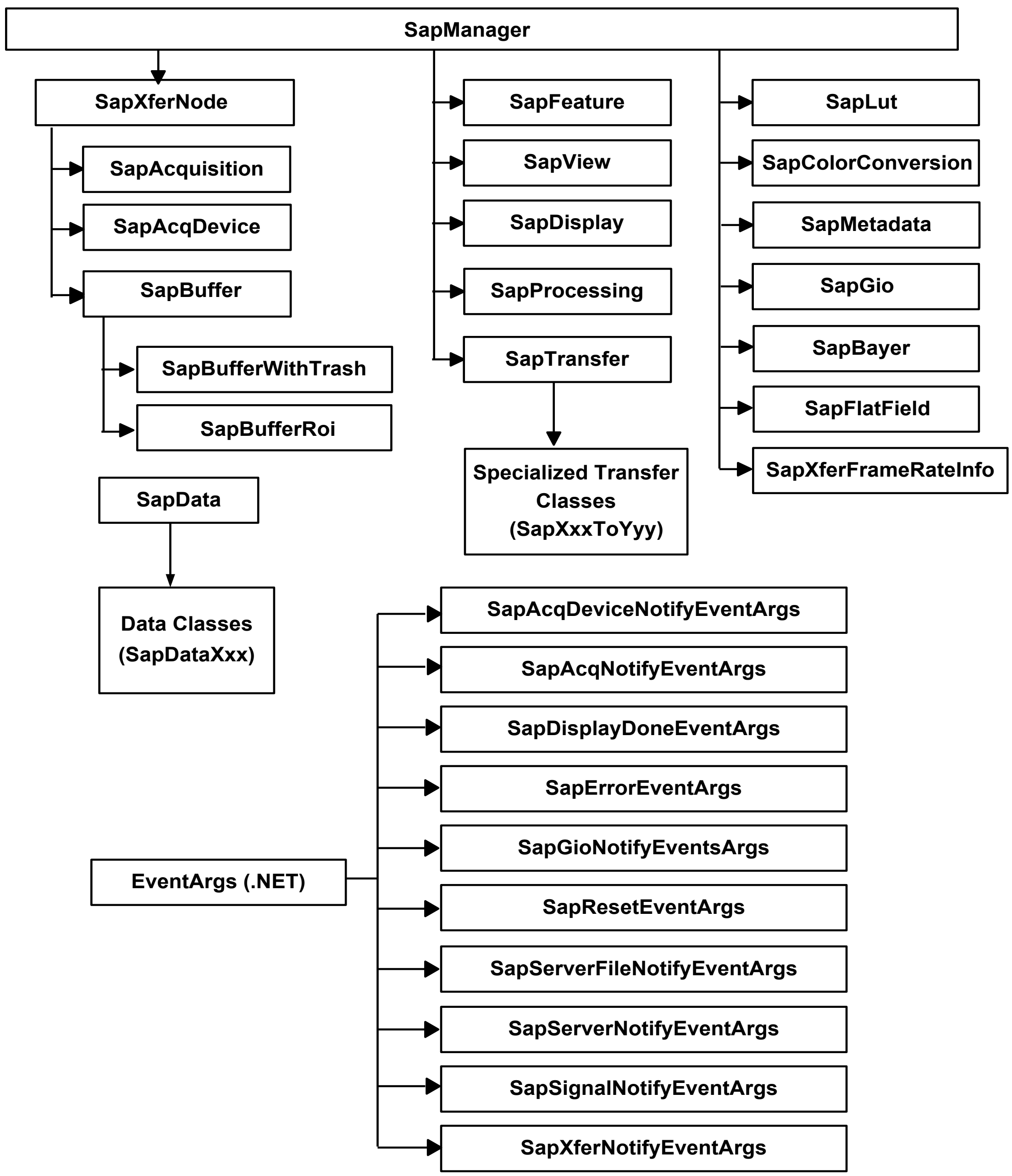
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017
- Microsoft Visual Studio 2019

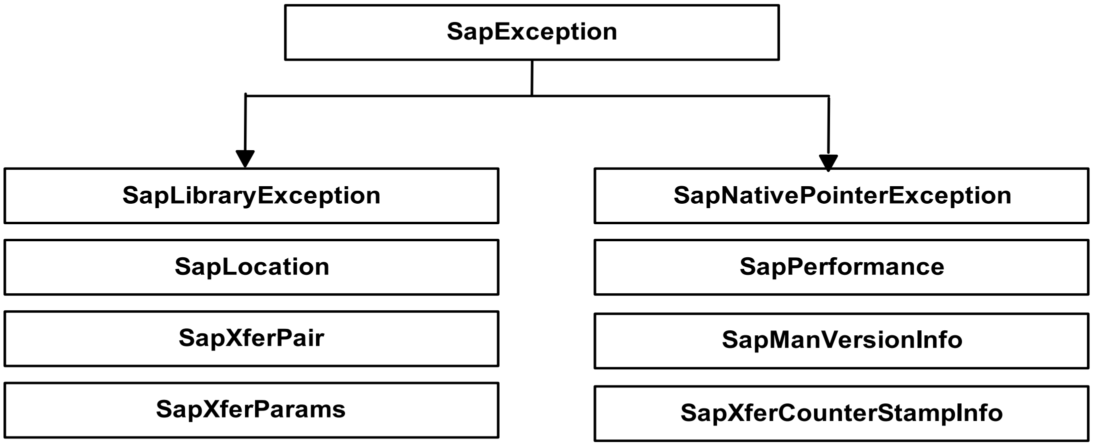
File Locations

Description	Location
Dynamic-link libraries (DLLs)	**Sapera\Components\NET\Bin**

Hierarchy Charts

Basic Class Hierarchy Chart




SapException
SapLibraryException
SapNativePointerException
SapLocation
SapPerformance
SapXferPair
SapManVersionInfo
SapXferParams
SapXferCounterStampInfo

Using Sapera .NET

Note about .NET Framework and .NET

In 2019, Microsoft® released .NET Framework 4.8, officially the last version of the framework. It is being replaced by .NET (formerly .NET Core), an open-source, cross-platform software framework.

Keeping with these advancements, Sapera LT, starting from version 8.7, will support both .NET Framework and .NET technologies. This concurrent support will remain for the foreseeable future, as .NET Framework is widely used. It will also facilitate transition for early adopters of .NET.

For .NET users or potential users, keep in mind that:

- Sapera LT 8.7 does not support .NET Core 3.1, only .NET 5 and later versions.
- You must use the 64-bit installation of Sapera LT.
- You must use Visual Studio® 2019 or later, as earlier versions do not support .NET 5.
- The DLL to include as a reference in Visual Studio 2019 projects is DALSA.SaperaLT.SapClassBasic.Core.dll, located in the (SaperaDir)\Components\NET\Bin folder.
- There are new Visual Studio 2019 C# solutions for .NET:
    - (SaperaDir)\Examples\NET\SapNETCSExamples_2019.Core.sln and
    - (SaperaDir)\Demos\NET\SapNETCSDemos_2019.Core.sln.
- The C# source code used in the examples and demos is the same for .NET Framework and for .NET since it is the same Sapera .NET API.

Sapera .NET – Creating an Application

The instructions below describe how to create a Sapera .NET application using C# in Visual Studio 2013/2015/2017/2019.

Sapera .NET DLLs

The following files are provided with Sapera LT.

File Name	Description	Location
DALSA.SaperaLT.SapClassBasic.dll	.NET Framework classes DLL	**Sapera\Components\NET\Bin**
DALSA.SaperaLT.SapClassBasic.Core.dll	.NET 5 classes DLL	**Sapera\Components\NET\Bin**

Creating a .NET Framework C# Application

- From the main menu select **File** > **New** > **Project.**
- In the New Project dialog, on the left, select **Installed** > **Templates** > **Visual C#** > **Windows**.
- In the Templates list select **Windows Forms Application** or **Console Application**.
- After the project has been successfully created go to Solution Explorer. Right-click **References** and select **Add Reference**.
- In the Reference Manager window, open the **Browse** node on the left, and select **DALSA.SaperaLT.SapClassBasic.dll**. If the entry is not present, click the **Browse** button at the bottom, and select **Sapera\Components\NET\Bin\DALSA.SaperaLT.SapClassBasic.dll**.
- In each source file that needs to access the Sapera .NET classes, add the following statement:

  ```
  using DALSA.SaperaLT.SapClassBasic;
  ```

If you get a processor architecture mismatch warning when compiling the project, change the platform target as follows:

- Open the project Properties, go to the **Build** category, change the **Platform target** setting to either *x64* (for 64-bit OS) or *x86* (for 32-bit OS), then recompile.

Creating a .NET C# Application

Visual Studio 2019 must be installed, as well as .NET 5.0 or later.

- Create a new project.
- In the Create a new project dialog, do one or the other according to application type:

  For a console application, select as indicated:
  - Language: *C#*
  - Platform: *Windows*
  - Project type: *Console*
  - Template: *Console Application* (DO NOT choose the template for .NET Framework)

  For a GUI application, select as indicated:
  - Language: *C#*
  - Platform: *Windows*
  - Project type: *Desktop*
  - Template: *Windows Forms App* (DO NOT choose the template for .NET Framework)
- Enter a project name and location.
- In the **Target Framework** list, choose *.NET 5.0*. Click **Create**.
- After the project has been successfully created, go to Solution Explorer, right-click the project name, then select **Add** > **Project Reference**.
- In the Reference Manager dialog, expand the **Browse** node on the left, and select DALSA.SaperaLT.SapClassBasic.Core.dll. If this entry is not present:
  - Click the **Browse** button on the bottom.
  - Select Sapera\Components\NET\Bin\DALSA.SaperaLT.SapClassBasic.Core.dll.
- In each source file that needs to access the Sapera .NET classes, add the following statement:

```
using DALSA.SaperaLT.SapClassBasic;
```

If you get a processor architecture mismatch warning when compiling the project, change the platform target as follows:

- Open the project Properties dialog, go to the **Build** category, change the **Platform target** setting to *x64*, then recompile.

Demos and Examples

Refer to the *Sapera LT GettingStarted for Frame Grabbers and Getting Started for GigE Vision Cameras manuals* for a description of the Sapera LT demos as well as available examples.

Source code for all demos and examples is provided.

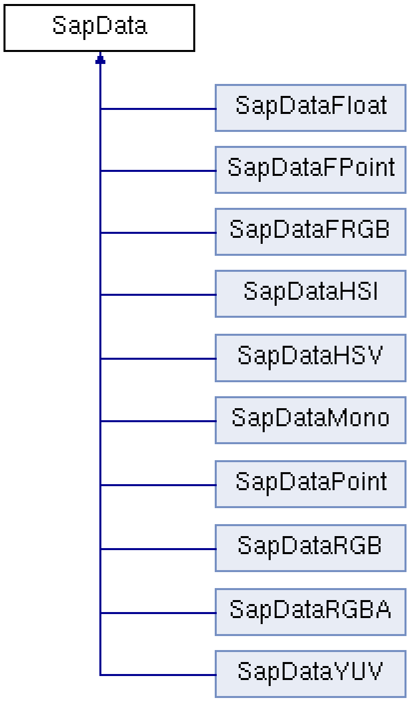
Basic Class Reference

The reference material for Sapera .NET uses the C# language. The following table lists data type equivalents between C# and the corresponding .NET type.

Native .NET	C#
System.Void	void
System.Int32	int
System.UInt32	uint
System.Boolean	bool
System.Single	float
System.Double	double
System.String	string
System.Int64	long
System.UInt64	ulong
System.IntPtr	System.IntPtr

Also, to keep notation short, the 'DALSA.SaperaLT.SapClassBasic' namespace prefix is omitted for the Sapera .NET reference material.

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 ++ .NET 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 **FormatType** (read-only property)

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

SapDataCoord3D Class

Purpose

Encapsulates one element supporting Sapera Coord3D data types.

SapDataCoord3D(int *a*, int *c*, int *r*);
SapDataCoord3D();

Class constructor, where the *a*, *c* and *r* arguments specify the X, Z and reflectance components as an initial value other than (0, 0).

int **A**(); (read/write property)

Returns the current value of A component of the data element.

int **C**(); (read/write property)

Returns the current value of C component of the data element.

int **R**(); (read/write property)

Returns the current value of R component of the data element.

Demo/Example Usage

Not available

SapDataCoord3D_PC Class

Purpose

Encapsulates one element supporting Sapera Coord3D data types.

SapDataCoord3D_PC(float *x*, float *y*, float *z*);
SapDataCoord3D_PC();

Class constructor, where the *x*, *y* and *z* arguments specify the a 3D point cloud coordinate as an initial value other than (0.0, 0.0, 0.0).

float **X**(); (read/write property)

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

float **Y**(); (read/write property)

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

float **Z**(); (read/write property)

Returns the current value of Z component of the data element.

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

void **Dispose**();
Frees unmanaged memory used internally by a SapDataFRGB .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

float **Red** (read/write property)
Returns the red component of the current value of the data element

float **Green** (read/write property)
Returns the green component of the current value of the data element

float **Blue** (read/write property)
Returns the blue component of the current value of 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

void **Dispose**();
Frees unmanaged memory used internally by a SapDataHSI .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int **H** (read/write property)
Returns the H component of the current value of the data element

int **S** (read/write property)
Returns the S component of the current value of the data element

int **I** (read/write property)
Returns the I component of the current value of the data element

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

void **Dispose**();
Frees unmanaged memory used internally by a SapDataHSV .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int **H** (read/write property)
Returns the H component of the current value of the data element

int **S** (read/write property)
Returns the S component of the current value of the data element

int **V** (read/write property)
Returns the V component of the current value of the data element

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

void **Dispose**();
Frees unmanaged memory used internally by a SapDataMono .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int **Mono** (read/write property)
Returns the current value of the data element

Demo/Example Usage

Example Common Utiltities

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

void **Dispose**();

Frees unmanaged memory used internally by a SapDataRGB .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int **Red** (read/write property)

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

int **Green** (read/write property)

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

int **Blue** (read/write property)

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

Demo/Example Usage

Example Common Utiltities

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

void **Dispose**();

Frees unmanaged memory used internally by a SapDataRGBA .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int Red (read/write property)

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

int Green (read/write property)

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

int Blue (read/write property)

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

int Alpha (read/write property)

Returns the alpha component of the current value of 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

void **Dispose**();

Frees unmanaged memory used internally by a SapDataYUV .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

int **Y** (read/write property)

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

int **U** (read/write property)

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

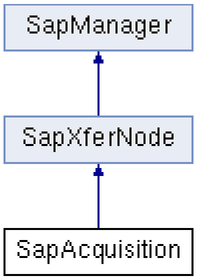
int **V** (read/write property)

Returns the V component of the current value of 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.

Namespace: DALSA.SaperaLT.SapClassBasic

Note: GigE Vision 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
Dispose	Frees unmanaged memory resources
Properties	
AcqNotifyContext	Application specific data for acquisition events
CameraSelector	Current camera selector value
CamIoControl	Custom camera control I/O description
ColorConversionAvailable	Availability of hardware-based color conversion
ConfigFileName	Name of the acquisition configuration file (CCF)
EventType	Registered acquisition event types
FlatFieldAvailable	Availability of hardware-based flat-field correction
Flip	Flipping (that is, mirroring) mode for acquired images
ImageFilterAvailable	Gets availability of hardware-based image filter
ImageFilterEnable	Gets the current image filter enable value
Label	Text description of the acquisition resource
LutEnable	Enables/disables the acquisition lookup tables
Luts	Complete list of acquisition lookup tables
NumLuts	Number of available acquisition look-up tables
NumPlanarInputs	Number of cameras used for acquiring into vertical planar buffers
PlanarInputs	Current configuration for acquiring into vertical planar buffers
SerialPortName	Name of the serial port attached to the current acquisition device
SignalNotifyContext	Application specific data for signal status notification events
SignalNotifyEnable	Enables/disables signal status notification events

SignalStatus	Current status of input acquisition signals
TimeStampAvailable	Gets availablity of hardware-based timestamp
TimeStampBase	Gets/sets the timestamp base unit
ColorConversionAvailable	Gets available of hardware-based gains for white balance control
Methods	
ApplyLut	Reprograms an acquisition lookup table
CustomCommand	Issues a low-level custom command specific to the acquisition hardware
DisableEvent	Disables all acquisition event types
EnableEvent	Enables acquisition event types
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetImageFilter, SetImageFilter	Gets/sets the values of the acquisition image filter
GetImageFilterKernelSize	Gets the image filter kernel size
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterSize	Gets the number of bytes required for an acquisition parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
IsSignalStatusActive	Gets the current status of input acquisition signals
IsSignalStatusAvailable	Checks for availability of the status of input acquisition signals
LoadImageFilter	Loads a hardware-based image filter kernel from file
ResetTimeStamp	Resets the acquisition hardware timestamp counter to zero
SaveImageFilter	Save a hardware-based image filter kernel from file
SaveParameters	Saves the acquisition parameters to an acquisition configuration file (CCF)
SoftwareTrigger	Simulates a trigger to the acquisition device
Events	
AcqNotify	Notification of acquisition related hardware events
SignalNotify	Notification of signal status related hardware events

SapAcquisition Member Functions

The following are members of the SapAcquisition Class.

SapAcquisition.SapAcquisition (constructor)

SapAcquisition();
SapAcquisition(SapLocation *location***);**
SapAcquisition(SapLocation *location*, string *configFileName***);**

Parameters

location	SapLocation object specifying the server where the acquisition resource is located and the index of the acquisition resource on this server.
configFileName	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.

Remarks

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

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, Multi-Board Sync Grab Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapAcquisition.AcqNotify Event

SapAcqNotifyHandler **AcqNotify**

Description

Notifies the application of acquisition related hardware events. Use the EventType property to set the hardware events that the application needs to be notified of. Use the AcqNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void Acq_AcqNotify(Object sender, SapAcqNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapAcquisition object for which the event has been registered.

Demo/Example Usage

Sequential Grab Demo

SapAcquisition.AcqNotifyContext Property

System.Object **AcqNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the AcqNotify event is invoked. This can be any object instance derived from the System.Object base type. See the AcqNotify event description for more details.

Demo/Example Usage

Sequential Grab Demo

SapAcquisition.ApplyLut Method

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. Valid values for *index* are from 0 to the value returned by the NumLuts property, minus 1. This value will be 0 most the time to specify the first (and often the only) LUT.

Use the Luts property to gain access to all acquisition LUTs, then use the methods in the SapLut Class to manipulate them. 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.CameraSelector Property

int **CameraSelector** (read/write)

Description

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 property is 0. It is then set according to the current acquisition device value when calling the Create method.

You cannot change the value of this property before calling the Create method, or during live acquisition, that is, when the SapTransfer.Grabbing property returns **true**.

Demo/Example Usage

Not available

SapAcquisition.CamIoControl Property

SapAcqCamIoControl[] **CamIoControl** (read/write)

Description

Custom camera control I/O description. When setting this property, the array of SapAcqCamIoControl objects may have from 1 to 32 entries. When reading this property, the returned array always has 32 entries.

The SapAcqCamIoControl object has the following properties:

string	Label	User defined descriptive label of the camera control (for example, BIN or GAIN)
int	ConnectorInput	Pin Connector Description
int	NumBits	Number of bits needed for this control, where 0 means that the control is not used by the acquisition hardware.
SapAcquisition.SignalLevel	Level	SapAcquisition.SignalLevel.TTL SapAcquisition.SignalLevel. RS422SapAcquisition.SignalLevel.LVDS
SapAcquisition.SignalDirection	Direction	SapAcquisition.SignalDirection.Input SapAcquisition.SignalDirection.Output
SapAcquisition.SignalPolarity	Polarity	Used only for information purposes by the application: SapAcquisition.SignalPolarity.ActiveLow SapAcquisition.SignalPolarity.ActiveHigh
int	Value	The default value of the control when used as an output. If a bit is set to 1, the corresponding output will be set to on/high, otherwise, it will be set to off/low.

Here are examples of how to use this property:

```
SapAcqCamIoControl[] currentCamIoControl = acq.CamIoControl;

SapAcqCamIoControl[] newCamIoControl = new SapAcqCamIoControl[2];
for (int i = 0; i < newCamIoControl.Length; i++)
newCamIoControl[i] = new SapAcqCamIoControl();

// Initialize fields of newCamIoControl[0] and newCamIoControl[1]
…
acq.CamIoControl = newCamIoControl;
```

For more information about custom camera controls, see the *Sapera LT Acquisition Parameters Reference Manual.*

Demo/Example Usage

Bayer Demo, FlatField Demo

SapAcquisition.ColorConversionAvailable Property

bool **ColorConversionAvailable** (read-only)

Description

Availability of hardware-based color conversion. You can only read this property after calling the Create method.

Demo/Example Usage

ColorConversion Demo, FlatField Demo

SapAcquisition.ConfigFileName Property

string **ConfigFileName** (read/write)

Description

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 change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapAcquisition.Create Method

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, Multi-Board Sync Grab Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapAcquisition.CustomCommand Method

bool **CustomCommand**(int *command*, System.IntPtr *inData*, int *inDataSize*, System.IntPtr *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 Method

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, Multi-Board Sync Grab Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapAcquisition.DisableEvent Method

bool **DisableEvent**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Disables all registered acquisition event types: see the SapAcquisition.EventType Property for a description of available events.

Demo/Example Usage

Not available

SapAcquisition.Dispose Method

void **Dispose()**;

Remarks

Frees unmanaged memory used internally by a SapAcquisition .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapAcquisition object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, Flat Field Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapAcquisition.EnableEvent Method

bool **EnableEvent**(SapAcquisition.AcqEventType *eventType*);

Parameters

eventType	Low-level command ID

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Enables an acquisition event type, or a combination of registered acquisition event types, for which the AcqNotify event will occur. One or more values may be combined together using a bitwise OR operation: see the SapAcquisition.EventType Property for a description of available events.

Demo/Example Usage

Not available

SapAcquisition.EventType Property

SapAcquisition.AcqEventType **EventType (**read/write)

Description

Combination of registered acquisition event types for which the AcqNotify event will occur. One or more of the following values may be combined together using a bitwise OR operation:

AcqEventType.DataOverflow	Data overflow occurred during live acquisition. This usually occurs if the acquisition device cannot sustain the data rate of the incoming images.
AcqEventType.EndOfEven	End of even field
AcqEventType.EndOfField	End of any field (odd or even)
AcqEventType.EndOfFrame	End of frame
AcqEventType.EndOfLine	After a specific line number After a specific line number is acquired. When used, the event type must be ORed with an unsigned integer (max 65535) representing the line number after which the callback function has to be called: EventType property = EndOfLine | *lineNum* Note that *lineNum* only applies when writing to the EventType property; its value is not returned when reading this property and the corresponding bits are set to 0.
AcqEventType.EndOfNLines	After a specific number of lines (linescan cameras only) is acquired. When used, the event type must be ORed with an unsigned integer (max 65535) representing the number of lines after which the callback function has to be called: EventType property = EndOfNLines | *NumLines* Note that *numLines* only applies when writing to the EventType property; its value is not returned when reading this property and the corresponding bits are set to 0.
AcqEventType.EndOfOdd	End of odd field
AcqEventType.ExternalTrigger	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.

AcqEventType.ExternalTriggerIgnored	Dropped an external trigger event. This usually occurs when the external trigger rate is faster then the acquisition frame rate.
AcqEventType.ExternalTriggerTooSlow	The detected external line trigger rate is too slow for the hardware to process. This can usually occur when using the shaft encoder multiplier.
AcqEventType.FrameLost	Lost a frame during live acquisition. This usually occurs if there is not enough bandwidth to transfer images to host memory.
AcqEventType.HsyncLock	Detected a horizontal sync unlock to lock condition.
AcqEventType.HsyncUnlock	Detected a horizontal sync lock to unlock condition.
AcqEventType.LineTriggerTooFast	The detected line trigger rate is too fast for the hardware to process. This can occur when using the shaft encoder multiplier.
AcqEventType.LinkError	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.
AcqEventType.LinkLock	Detected all required lanes are locked (for HSLink and CLHS cameras only).
AcqEventType.LinkUnlock	Detected at least one of the required lanes lost the link lock (for HSLink and CLHS cameras only).
AcqEventType.NoHsync	
AcqEventType.None	No events
AcqEventType.NoPixelClk	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.
AcqEventType.NoVsync	
AcqEventType.PixelClk	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.
AcqEventType.Shaft EncodeReverseCountOverflow	Detected an overflow of the shaft encoder reverse counter.
AcqEventType.StartOfEven	Start of even field
AcqEventType.StartOfField	Start of any field (odd or even)
AcqEventType.StartOfFrame	Start of frame
AcqEventType.StartOfOdd	Start of odd field
AcqEventType.VerticalSync	Vertical sync detected, even if not acquiring
AcqEventType.VerticalTimeout	Detected a vertical timeout. You can set the timeout value by calling the SetParameter method for SapAcquisition.Prm.VERTICAL_TIMEOUT_DELAY.
AcqEventType.VirtualFrame	Equivalent to StartOfFrame for linescan cameras

Note that you will not usually need to catch acquisition events. They must not be confused with the transfer event mechanism used in almost all applications. If you need acquisition events, review the User's Manual for your acquisition hardware to find which ones are supported. For transfer related events, see the SapTransfer class for more information.

The initial value for this property is EventNone.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Sequential Grab Demo

SapAcquisition.FlatFieldAvailable Property

bool **FlatFieldAvailable** (read-only)

Description

Availability of hardware-based flat-field correction. You can only read this property after calling the Create method.

Demo/Example Usage

FlatField Demo

SapAcquisition.Flip Property

SapAcquisition.FlipMode **Flip** (read/write)

Possible Values

FlipMode.None	No flipping
FlipMode.Horizontal	Acquired images are flipped horizontally
FlipMode.Vertical	Acquired images are flipped vertically

Remarks

Flipping (that is, mirroring) mode for acquired images. The initial value for this property is FlipNone.

You can only change the value of this property after calling the Create method.

Demo/Example Usage

Not available

SapAcquisition.GetCapability Method

bool **GetCapability**(SapAcquisition.Cap *capID*, out int *capValue*);
bool **GetCapability**(SapAcquisition.Cap *capID*, out SapAcquisition.Val *capValue*);
bool **GetCapability**(SapAcquisition.Cap *capID*, out int[] *capValue*);
bool **GetCapability**(SapAcquisition.Cap *capID*, out SapAcquisition.Val[] *capValue*);

Parameters

capID	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the acquisition module.

Use the GetCapabilityType method to find out which version of GetCapability to use. If the return value is SapCapPrmType.Int32, then *capValue* is an integer. If this value is SapCapPrmType.Int32Array, then *capValue* is an uninitialized integer array with an unknown number of elements.

The following examples show how to declare this array and call GetCapability:

```
int[] capValue;
result = acq.GetCapability(capId, out capValue);
```

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORACQ_CAP_INTERFACE becomes SapAcquisition.Cap.INTERFACE

You can also use the versions of GetCapability which take a SapAcquisition.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORACQ_VAL_INTERFACE_DIGITAL becomes SapAcquisition.Val.INTERFACE_DIGITAL

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

Demo/Example Usage

Grab Demo

SapAcquisition.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapAcquisition.Cap *capID*);

Parameters

capID	Low-level Sapera capability for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.Int32Array	Array of 32-bit integers

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapAcquisition.GetImageFilter, SapAcquisition.SetImageFilter Methods

BOOL **GetImageFilter**(int *filterIndex*, SapBuffer *kernel*);
BOOL **SetImageFilter**(int *filterIndex*, SapBuffer *kernel*);

Parameters

filterIndex	Kernel filter index.
kernel	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.ImageFilterAvailable Property 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 Method. The values can be accessed using the SapBuffer.ReadElement Method and SapBuffer.WriteElement Method.

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).

Note for color calibration coefficients, the matrix size is 4x3 and is of type SapFormatFloat (the divisor is not used).

Demo/Example Usage

Not available

SapAcquisition.GetImageFilterKernelSize Method

BOOL **GetImageFilterKernelSize**(int *filterIndex*, ImageFilterKernelSize *size*);

filterIndex Kernel filter index.

size Kernel size. Possible values are:

ImageFilterKernelSize.Size1x1
ImageFilterKernelSize.Size2x2
ImageFilterKernelSize.Size3x3
ImageFilterKernelSize.Size4x4
ImageFilterKernelSize.Size5x5
ImageFilterKernelSize.Size6x6
ImageFilterKernelSize.Size7x7
ImageFilterKernelSize.Size8x8
ImageFilterKernelSize.Size4x3

Remarks

Gets acquisition hardware image filter kernel size.

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

Demo/Example Usage

Not available

SapAcquisition.GetParameter, SapAcquisition.SetParameter Methods

bool **GetParameter**(SapAcquisition.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapAcquisition.Prm *paramId*, out SapAcquisition.Val *paramValue*);
bool **GetParameter**(SapAcquisition.Prm *paramId*, out string *paramValue*);
bool **GetParameter**(SapAcquisition.Prm *paramId*, out int[] *paramValue*);

bool **SetParameter**(SapAcquisition.Prm *paramId*, int *paramValue*, bool *updateNow*);
bool **SetParameter**(SapAcquisition.Prm *paramId*, SapAcquisition.Val *paramValue*, bool *updateNow*);
bool **SetParameter**(SapAcquisition.Prm *paramId*, string *paramValue*, bool *updateNow*);
bool **SetParameter**(SapAcquisition.Prm *paramId*, int[] *paramValue*, bool *updateNow*);

Parameters

paramId Low-level Sapera parameter to read or write

paramValue Parameter value to read or 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 parameters for the acquisition module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.Int32Array, then *paramValue* is an integer array (uninitialized for GetParameter) with an unknown number of elements. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter).

The following examples show how to declare an uninitialized array and call GetParameter:

```
int[] paramValue;
result = acq.GetParameter(paramId, out paramValue)
```

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORACQ_PRM_INTERFACE becomes SapAcquisition.Prm.INTERFACE

You can also use the versions of GetParameter/SetParameter which take a SapAcquisition.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORACQ_VAL_INTERFACE_DIGITAL becomes SapAcquisition.Val.INTERFACE_DIGITAL

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.

Demo/Example Usage

Bayer Demo, FlatField Demo, Sequential Grab Demo

SapAcquisition.GetParameterSize Method

static unsigned int **GetParameterSize**(Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

Returns the number of bytes required for an acquisition parameter.

Remarks

Use the SapAcquisition.IsParameterAvailable Method to verify if a parameter is supported and the SapAcquisition.GetParameter, SapAcquisition.SetParameter Methods to access and modify paramter settings.

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

Demo/Example Usage

Not available

SapAcquisition.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapAcquisition.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.Int32Array	Array of 32-bit integers
SapCapPrmType.IntPtr	32-bit integer pointer
SapCapPrmType.String	Text string

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapAcquisition.ImageFilterAvailable Property

bool **ImageFilterAvailable**();

Return Value

Returns **true** if the hardware acquisition image filter is supported, **false** otherwise

Remarks

If the hardware acquisition image filter is supported, use the SapAcquisition.GetImageFilter, SapAcquisition.SetImageFilter Methods to access the kernel buffer. The size of the image filter is retrieved using the SapAcquisition.GetImageFilterKernelSize Method. The values can be accessed using the SapBuffer.ReadElement Method and SapBuffer.WriteElement Method.

Demo/Example Usage

Not available

SapAcquisition.ImageFilterEnable Property

bool **ImageFilterEnable**();

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.ImageFilterAvailable Property.

Demo/Example Usage

Not available

SapAcquisition.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapAcquisition.Cap *capId*);

Parameters

capId Low-level Sapera 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 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 Basic Modules Reference Manual* for a description of all capabilities and their possible values. Also see the GetCapability method for more information about the syntax to use for *capId*.

Demo/Example Usage

Not available

SapAcquisition.IsParameterAvailable Method

bool **IsParameterAvailable**(SapAcquisition.Prm *paramId*);

Parameters

paramId Low-level Sapera 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 parameter for the acquisition module. Call this method before GetParameter to avoid invalid or not available parameter errors.

Note that this method is rarely needed. The SapAcquisition 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. Also see the GetParameter method for more information about the syntax to use for *paramId*.

Demo/Example Usage

Not available

SapAcquisition.IsSignalStatusActive Method

bool **IsSignalStatusActive**(SapAcquisition.AcqSignalStatus *signalStatus*);

Parameters

signalStatus	Status signal to inquire. See the IsSignalStatusAvailable method for a list of possible values.

Return Value

Returns **true** if the the specified status signals have been detected by the acquisition device, **false** otherwise

Remarks

Since many signals may be detected at the same time, values may be combined together using a bitwise OR operation.

Demo/Example Usage

Not available

SapAcquisition.IsSignalStatusAvailable Method

bool **IsSignalStatusAvailable**(SapAcquisition.AcqSignalStatus *signalStatus*);

Parameters

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

AcqSignalStatus.None	No signal.
AcqSignalStatus.HsyncPresent	Horizontal sync signal (analog video source) or line valid (digital video source).
AcqSignalStatus.VsyncPresent	Vertical sync signal (analog video source) or frame valid (digital video source).
AcqSignalStatus.PixelClkPresent	Pixel clock signal.
AcqSignalStatus.PixelClkPresent /	Pixel clock signal. For CameraLink devices, this status returns true if a clock signal is detected on the base cable.
AcqSignalStatus.PixelClk1Present	
AcqSignalStatus.PixelClk2Present	Pixel clock signal. For CameraLink devices, this status returns true if a clock signal is detected on the medium cable.
AcqSignalStatus.PixelClk3Present	Pixel clock signal. For CameraLink devices, this status returns true if a clock signal is detected on the full cable.
AcqSignalStatus.PixelClkAllPresent	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.
AcqSignalStatus.ChromaPresent	Color burst signal (valid for NTSC and PAL)
AcqSignalStatus.HsyncLock	Successful lock to an horizontal sync signal, for an analog video source
AcqSignalStatus.VsyncLock	Successful lock to a vertical sync signal, for an analog video source
AcqSignalStatus.PowerPresent	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.
AcqSignalStatus.PoCLActive	Power to the camera is present on the Camera Link cable

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.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapAcquisition.Label Property

string **Label** (read-only)

Description

Text description of the acquisition resource. This property 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.LoadImageFilter Method

bool **LoadImageFilter**(UINT32 *filterIndex*, string *fileName*);

Parameters

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

This function also supports loading color calibration coefficients files (*.ccor), if supported by the device.

Demo/Example Usage

Not available

SapAcquisition.LutEnable Property

bool **LutEnable (**read/write)

Description

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. The initial value for this property is **false**. It is then set according to the current acquisition device value when calling the Create method.

Note that some acquisition devices do not support this feature.

Demo/Example Usage

Grab LUT Demo

SapAcquisition.Luts Property

SapLut[] **Luts** (read-only)

Description

Complete list of acquisition lookup tables. All available LUTs on the acquisition device are automatically created and initialized when calling the Create method. You can manipulate them through the methods in the SapLut class, and reprogram them using the ApplyLut method.

Here are examples of how to retrieve this property:

```
SapLut[] allLuts = acq.Luts;
if (allLuts != null) …
```

Note that the examples check for a null value for this property, which is the case if the current acquisition device does not support lookup tables.

Demo/Example Usage

Grab LUT Demo

SapAcquisition.NumLuts Property

int **NumLuts** (read-only)

Description

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.NumPlanarInputs Property

int **NumPlanarInputs** (read-only)

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.PlanarInputs Property

bool[] **PlanarInputs (**read/write)

Description

Current configuration for synchronous acquisition into vertical planar buffers, where all cameras are synchronized together.

Individual entries in the array (number of entries = value of NumPlanarInputs property) 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 the array 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.

Here are examples of how to retrieve this property:

```
bool[] planarInputs = acq.PlanarInputs;
```

Note that you can only access this property after calling the Create method.

Demo/Example Usage

Not available

SapAcquisition.ResetTimeStamp Method

bool **ResetTimeStamp**

Description

Resets the acquisition hardware timestamp counter to zero

Demo/Example Usage

Not available

SapAcquisition.SaveImageFilter Method

bool **SaveImageFilter**(UINT32 *filterIndex*, string *fileName*);

Parameters

filterIndex	Filter index into which to load the kernel.
fileName	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 the SapAcquisition.LoadImageFilter Method to load previously saved kernels.

Demo/Example Usage

Not available

SapAcquisition.SaveParameters Method

bool **SaveParameters**(string *configFileName*);

Parameters

configFileName	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.SerialPortName Property

string **SerialPortName** (read-only)

Description

Name of the serial port attached to the current acquisition device.

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

Demo/Example Usage

Not available

SapAcquisition.SignalNotify Event

SapSignalNotifyHandler **SignalNotify**

Description

Notifies the application of signal status related events. Use the SignalNotifyEnable property to enable or disable this notification. Use the SignalNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void Acq_SignalNotify(Object sender, SapSignalNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapAcquisition object for which the event has been registered.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapAcquisition.SignalNotifyContext Property

System.Object **SignalNotifyContext (**read/write)

Description

Supplies application specific data when the application event handler for the SignalNotify event is invoked. This can be any object instance derived from the System.Object base type. See the SignalNotify event description for more details.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapAcquisition.SignalNotifyEnable Property

bool **SignalNotifyEnable (**read/write)

Description

Enables/disables signal status notification events. The initial value for this property is **false**.

You can only set the value of this property after calling the Create method.

See the SignalNotify event for more details.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapAcquisition.SignalStatus Property

SapAcquisition.AcqSignalStatus **SignalStatus (**read-only)

Description

Current status of input acquisition signals. One or more values may be combined together using bitwise OR. See the IsSignalStatusAvailable method for a list of possible values.

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

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapAcquisition.SoftwareTrigger Method

bool **SoftwareTrigger**(SapAcquisition.SoftwareTriggerType *triggerType*);

Parameters

triggerType	Trigger type may be one of the following values	
	SoftwareTrigger.External	External trigger
	SoftwareTrigger.ExternalFrame	External frame trigger
	SoftwareTrigger.ExternalLine	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 SapAcquisition.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 SapAcquisition.Cap.SOFTWARE_TRIGGER capability.

Demo/Example Usage

Not available

SapAcquisition.TimeStampAvailable Property

bool **TimeStampAvailable**(read only)

Return Value

Returns **true** if successful, **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 the SapBuffer.CounterStamp Property.

Demo/Example Usage

Sequential Grab Demo

SapAcquisition.TimeStampBase Property

AcqTimeStampBase **TimeStampBase** (read/write)

Possible Values

The time base may be one of the following:

FrameTrigger	The time base is in valid external frame trigger received (does not count the ones that are ignored).
FrameValid	The time base is in frame valid signals received.
HundredNanoSecond	The time base is in 100 nano seconds.
LineTrigger	The time base is in external line trigger or shaft encoder pulse (after drop/multiply operation).
LineValid	The time base is in line valid signals received.
MicroSecond	The time base is in micro seconds.
MilliSecond	The time base is in milli seconds.
NanoSecond	The time base is in nano seconds.
None	Time base is not available.
PixelClock	The time base is in camera pixel clock.
ShaftEncoder	The time base is in external line trigger or shaft encoder pulse (before drop/multiply operation)

Remarks

Gets/sets the acquisition device timestamp base units.

SapAcquisition.WhiteBalanceAvailable Property

bool **WhiteBalanceAvailable** (read-only)

Description

Availability of hardware-based gains for white balance control. You can only read this property after calling the Create method.

Demo/Example Usage

ColorConversion Demo, FlatField Demo

SapAcqNotifyEventArgs

The SapAcqNotifyEventArgs class contains the arguments to the application handler method for the SapAcquisition.AcqNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapAcqNotifyEventArgs Class Members

Properties

AuxTimeStamp	Gets the auxiliary timestamp associated with acquisition events or signal status reporting
Context	Application context associated with acquisition device events
CustomData	Data associated with a custom event
CustomSize	Size of the custom data returned by CustomData property
EventCount	Current count of acquisition events
EventType	Acquisition events that triggered the invocation of the application event handler
GenericParamValue0	Generic properties supported by some events
GenericParamValue1	
GenericParamValue2	
GenericParamValue3	
HostTimeStamp	Gets the host timestamp associated with acquisition events or signal status reporting

SapAcqNotifyEventArgs Member Properties

The following are members of the SapAcqNotifyEventArgs Class.

SapAcqNotifyEventArgs.AuxTimeStamp Property

long **AuxTimeStamp** (read-only)

Description

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 SapAcqNotifyEventArgs.HostTimeStamp Property).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

SapAcqNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with acquisition events. See the AcqNotifyContext property of the SapAcquisition class for more details.

Demo/Example Usage

Sequential Grab Demo

SapAcqNotifyEventArgs.CustomData Property

System.IntPtr **Context** (read-only)

Description

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.

Demo/Example Usage

Not available

SapAcqNotifyEventArgs.CustomSize Property

int **CustomSize (**read-only)

Description

Size of the custom data returned by the CustomData property.

Demo/Example Usage

Not available

SapAcqNotifyEventArgs.EventCount Property

int **EventCount (**read-only)

Description

Current count of acquisition events. The initial value is 1 and increments every time the event handler method is invoked.

Demo/Example Usage

Sequential Grab Demo

SapAcqNotifyEventArgs.EventType Property

SapAcquisition.AcqEventType **EventType** (read-only)

Description

Combination of acquisition events that triggered the invocation of the application event handler. Since it is possible for multiple events to trigger one such invocation, this property may actually return a combination of many events, using a bitwise OR operator. See the SapAcquisition.EventType property for the list of possible values.

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

Demo/Example Usage

Not available

SapAcqNotifyEventArgs.GenericParamValue0 SapAcqNotifyEventArgs.GenericParamValue1 SapAcqNotifyEventArgs.GenericParamValue2 SapAcqNotifyEventArgs.GenericParamValue3 Properties

int **GenericParamValue0**
int **GenericParamValue1**
int **GenericParamValue2**
int **GenericParamValue3** (read-only)

Description

Any of the four generic properties supported by some events. You should use aliases instead when they are available. See the acquisition device User's Manual for a list of events using generic properties.

Demo/Example Usage

Not available

SapAcqNotifyEventArgs.HostTimeStamp Property

long **HostTimeStamp** (read-only)

Description

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.

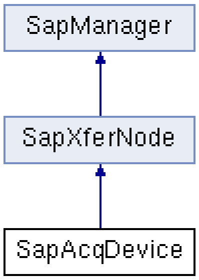
Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

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

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapAcqDevice Class Members

Construction	
SapAcqDevice	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AcqDeviceNotifyContext	Application specific data for acquisition events
ConfigFileName	Name of the acquisition configuration file (CCF)
CanEnableLut	Checks if the camera lookup table (LUT) may be enabled/disabled.
CategoryCount	Number of unique feature category names supported by the acquisition device
CategoryPathNames	Full path names of all unique feature categories supported by the acquisition device
ConfigurationName	Configuration name to be used when saving the device features using the SaveFeatures method
EventCount	Number of events supported by the acquisition device
EventNames	Names of all events supported by the acquisition device
FeatureCount	Number of features supported by the acquisition device
FeatureNames	Names of all features supported by the acquisition device
FileAccessAvailable	Availability of file access by the acquisition device
FileCount	Number of files supported by the acquisition device
FileNames	File names supported by the acquisition device
FlatFieldAvailable	Availability of hardware-based flat-field correction
Label	Text description of the acquisition device

LutEnable	Enables or disables the camera lookup table (LUT) module, that is, all available LUT selectors are affected together.
ModeName	Mode name to be used when saving the device features using the SaveFeatures method
RawBayerOutput	Returns whether or not the acquisition device output is raw Bayer
ReadOnly	Checks if the class has read-only access to the acquisition device
UpdateMode	Mode by which features are written to the acquisition device
Methods	
ApplyLut	Reprograms a camera lookup table (LUT).
DeleteDeviceFile	Deletes a file from the acquisition device
DisableEvent	Disables the event associated with a specified name or index
EnableEvent	Enables the event associated with a specified name or index
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetCategoryFeatureCount	Number of features within a category
GetCategoryFeatureIndexes	Array of feature indexes within a category
GetEventIndexByName	Returns the index of an event associated with a specified name
GetFeatureIndexByName	Returns the index of a feature associated with a specified name
GetFeatureInfo	Returns information on a feature associated with a specified name or index
GetFeatureValue	Returns the value of a feature associated with a specified name or index
GetFileIndexByName	Returns the index of a device file associated with a specified name
GetFileProperty	Gets a property of a specific file on the acquisition device
GetLut	Returns a SapLut object that contains the current format, number of entries and data of the specified lookup table (LUT).
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsEventAvailable	Returns whether or not an event is supported by the acquisition device
IsEventEnabled	Checks if the event associated with a specified name or index is enabled
IsFeatureAvailable	Returns whether or not a feature is supported by the acquisition device
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
LoadFeatures	Loads all the features from a configuration file
ReadFile	Reads a file from an acquisition device
SaveFeatures	Saves all (or a subset of) features to a configuration file.
StartFeatureRecording	Starts recording feature setting changes to file
PauseFeatureRecording	Pauses recording feature setting changes
ResumeFeatureRecording	Resumes recording feature setting changes
StopFeatureRecording	Stops recording feature setting changes
LoadFeatureRecording	Loads and applies feature setting changes from file
IsFeatureRecordingStarted	Returns if feature recording is started

SetFeatureValue	Sets the value of a feature associated with a specified name or index
UpdateFeaturesFromDevice	Gets all the features from the acquisition device at once
UpdateFeaturesToDevice	Sets all the features to the acquisition device at once
UpdateLabel	Updates the device label.
WriteFile	Writes a file to an acquisition device
Events	
AcqDeviceNotify	Notification of acquisition device related events

SapAcqDevice Member Functions

The following are members of the SapAcqDevice Class.

SapAcqDevice.SapAcqDevice (constructor)

SapAcqDevice();
SapAcqDevice(SapLocation *location*);
SapAcqDevice(SapLocation *location*, bool *readOnly*);
SapAcqDevice(SapLocation *location*, string *configFileName*);

Parameters

location	SapLocation object specifying the server where the acquisition device is located and the index of the acquisition device on this server.
readOnly	Set to **true** to force read-only access to the device. If another application is already accessing the device (through this class) use this option 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.
configFileName	Name of the acquisition configuration file (CCF) that describes all the acquisition parameters. A CCF file can be created using the *CamExpert* 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 three constructors are used when no configuration file is required. In such a case the default parameters of the acquisition device are used. Using the third constructor, you can 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 fourth 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 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.AcqDeviceNotify Event

SapAcqDeviceNotifyHandler **AcqDeviceNotify**

Description

Notifies the application of acquisition device related events. Use the EnableEvent and DisableEvent methods to set the hardware events that the application needs to be notified of. Use the AcqDeviceNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void AcqDevice_AcqDeviceNotify(Object sender, SapAcqDeviceNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapAcqDevice object for which the event has been registered.

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.AcqDeviceNotifyContext Property

System.Object **AcqDeviceNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the AcqNotify event is invoked. This can be any object instance derived from the System.Object base type. See the AcqNotify event description for more details.

Demo/Example Usage

GigE FlatField Demo

SapAcqDevice.ApplyLut Method

bool **ApplyLut**(string *lutSelector*);

Parameters

lutSelector   String identifying the lookup table to modify

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Reprograms a camera lookup table (LUT).

The ApplyLut method saves the modifications made to a SapLut object, thereby reprogramming a LUT.

The *lutSelector* argument specifies the LUT to modify. Note that camera LUTs are identified as a string instead of an index, matching the SFNC LUTSelector feature. If the camera has only one valid LUT selector value, you can specify a null value for this argument.

Due to the dynamic nature of camera lookup tables, every call to GetLut will return a different SapLut object. For this reason, make sure to call the GetLut method right before modifying the LUT data with the SapLut class methods, and to call the ApplyLut method right after modification to save the changes.

Demo/Example Usage

Grab LUT Example

SapAcqDevice.CanEnableLut Property

bool **CanEnableLut** (read-only)

Description

Checks if the camera lookup table (LUT) may be enabled/disabled.

The initial value for this property is false. It is then set according to the availability of the LUT enable feature on the current acquisition device when calling the Create method.

Note that some acquisition devices do not support enabling or disabling the LUT.

Demo/Example Usage

Not available

SapAcqDevice.CategoryCount

int **CategoryCount** (read-only)

Description

Returns the number of unique feature categories supported by the acquisition device. This is equivalent to getting the information for all available features (by reading the FeatureCount category, then calling GetFeatureInfo), retrieving the category name for each (by reading the SapFeature.Category property), and then counting the unique category names.

After reading this property, you can read the CategoryPathNames property to retrieve full path names for individual features.

The value of this property is only meaningful after calling the Create method.

Demo/Example Usage

Not available

SapAcqDevice.CategoryPathNames

string[] **CategoryPathNames (**read-only)

Description

Full path names of all unique feature categories supported by the acquisition device. You can also read the value of the CategoryCount property to get the total number of categories.

The path names are 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.

Here are examples of how to retrieve this property:

```
string[] categoryPathNames = acqDevice.CategoryPathNames;
```

Demo/Example Usage

Not available

SapAcqDevice.ConfigFileName Property

string **ConfigFileName** (read/write)

Description

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 proeprty in the SapAcqDevice constructor. If you use the default constructor, then this value is null.

You can only change the value of this property before the Create method.

Demo/Example Usage

GigE Metadata Demo, GigE Camera Compression Demo

SapAcqDevice.ConfigurationName Property

string **ConfigurationName** (read/write)

Description

Configuration name to use when saving the device features with 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 property is automatically updated.

Demo/Example Usage

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

SapAcqDevice.Create Method

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 Method

bool DeleteDeviceFile(string *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 FileCount property, minus 1, are valid.

Remarks

Deletes the specified file on the device.

To find out which device files names are available, use the FileCount property together with the GetFileNameByIndex method.

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

Demo/Example Usage

Camera Files Example

SapAcqDevice.Destroy Method

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.DisableEvent Method

bool DisableEvent(string *eventName*);
bool DisableEvent(int *eventIndex*);

Parameters

eventName	Name of the event. See the acquisition device User's Manual for the list of supported events.
eventIndex	Index of the event. Valid values for this index are from 0 to the value returned by the EventCount property, minus 1.

Remarks

Disables acquisition device notification events. You can only call this method after the Create method.

See the AcqDeviceNotify event for more details.

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapAcqDevice .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapAcqDevice object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

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.EnableEvent Method

bool EnableEvent(string *eventName*);
bool EnableEvent(int *eventIndex*);

Parameters

eventName	Name of the event. See the acquisition device User's Manual for the list of supported events.
eventIndex	Index of the event. Valid values for this index are from 0 to the value returned by the EventCount property, minus 1.

Remarks

Enables acquisition device notification events. You can only call this method after the Create method.

See the AcqDeviceNotify event for more details.

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.EventCount Property

int **EventCount** (read-only)

Description

Number of events supported by the acquisition device. Different devices do not necessarily support the same event set. The value of this property is only meaningful after calling the Create method.

Demo/Example Usage

Sequential Grab Demo, Camera Events Example

SapAcqDevice.EventNames Property

string[] **EventNames (**read-only)

Description

Names of all available acquisition device events. This property is especially useful when converting an event index (retrieved from the arguments to an event handler function) to the corresponding name.

Here are examples of how to retrieve this property:

```
string[] eventNames = acqDevice.EventNames;
```

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.FeatureCount Property

int **FeatureCount (**read-only)

Description

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 property, minus 1.

The value of this property is only meaningful after calling the Create method.

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice.FeatureNames Property

string[] **FeatureNames** (read-only)

Description

Names of all available acquisition device features. This property is especially useful when converting a feature index (retrieved from the arguments to an event handler function) to the corresponding name.

Here are examples of how to retrieve this property:

```
string[] featureNames = acqDevice.FeatureNames;
```

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.FileAccessAvailable Property

bool **FileAccessAvailable** (read-only)

Description

Availability of file access by the acquisition device. If this property is **false**, then you should not use the FileCount and FileNames properties, and also the GetFileIndexByName, GetFileProperty, WriteFile, ReadFile, and DeleteDeviceFile methods.

Demo/Example Usage

Not available

SapAcqDevice.FileCount Property

int **FileCount** (read-only)

Description

Number of files supported by the acquisition device. Use the returned value together with the FileNames property to get a list of supported device file names.

Demo/Example Usage

Camera Files Example

SapAcqDevice.FileNames Property

string[] **FileNames** (read-only)

Description

Names of all available acquisition device files. Use this property together with the FileCount property to find out which device files names are available.

Here are examples of how to retrieve this property:

```
string[] fileNames = acqDevice.FileNames;
```

Demo/Example Usage

Camera Files Example

SapAcqDevice.FlatFieldAvailable Property

bool **FlatFieldAvailable (**read-only)

Description

Availability of hardware-based flat-field correction. You can only read this property after calling the Create method.

Demo/Example Usage

FlatField Demo

SapAcqDevice.GetCapability Method

bool **GetCapability**(SapAcqDevice.Cap *capId*, out int *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the acquisition device module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapAcqDevice class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORACQDEVICE_CAP_PERSISTENCE becomes SapAcqDevice.Cap.PERSISTENCE

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

Demo/Example Usage

Not available

SapAcqDevice.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapAcqDevice.Cap *capID*);

Parameters

capID	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapAcqDevice.GetCategoryFeatureCount Method

bool **GetCategoryFeatureCount**(int *categoryIndex*, out int *featureCount*);

Parameters

categoryIndex	Index of the category. All indices from 0 to the value returned by the CategoryCount property, minus 1, are valid.
featureCount	Number of features in the specified category

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the number of features within a specified category. Use the SapAcqDevice.GetCategoryFeatureIndexes method to return the list of indexes associated with each category.

Demo/Example Usage

Not available

SapAcqDevice.GetCategoryFeatureIndexes Method

bool **GetCategoryFeatureIndexes**(int *categoryIndex*, out int[] *indexList*);

Parameters

categoryIndex	Index of the category. All indices from 0 to the value returned by the CategoryCount property, minus 1, are valid.
indexList	Feature indexes, as an array of 32-bit values.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns an array of indices corresponding to each feature in the category. These indexes are the same as those we use when enumerating features using the SapAcqDevice.FeatureCount and SapAcqDevice.FeatureNames properties. The number of features in a category is returned by the SapAcqDevice.GetCategoryFeatureCount method.

Note that some internal camera features are not available, even though they are accounted for in SapAcqDevice.GetCategoryFeatureCount. In this case, the *indexList* entries for these features are set to -1 to indicate that they must be skipped.

Demo/Example Usage

Not available

SapAcqDevice.GetEventIndexByName Method

bool **GetEventIndexByName**(string *eventName*, out 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 method 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.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *eventIndex* argument.

Demo/Example Usage

Not available

SapAcqDevice.GetFeatureIndexByName Method

bool **GetFeatureIndexByName**(string *featureName*, out 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 method 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.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *featureIndex* argument.

Demo/Example Usage

Not available

SapAcqDevice.GetFeatureInfo Method

bool **GetFeatureInfo**(string *featureName*, SapFeature *feature*);
bool **GetFeatureInfo**(int *featureIndex*, SapFeature *feature*);

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 FeatureCount property, minus 1, are valid.
feature	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.

For enumeration features, it is possible to use this function to retrieve the information for individual enumeration values by using the "EnumerationName.ValueName" form for the *featureName* argument. In this case, it is then possible to retrieve the description of each enumeration value by readlng the Description property.

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.GetFeatureValue Method

bool **GetFeatureValue**(string *featureName*, out int *featureValue*);
bool **GetFeatureValue**(string *featureName*, out uint *featureValue*);
bool **GetFeatureValue**(string *featureName*, out long *featureValue*);
bool **GetFeatureValue**(string *featureName*, out ulong *featureValue*);
bool **GetFeatureValue**(string *featureName*, out float *featureValue*);
bool **GetFeatureValue**(string *featureName*, out double *featureValue*);
bool **GetFeatureValue**(string *featureName*, out bool *featureValue*);
bool **GetFeatureValue**(string *featureName*, out string *featureString*);
bool **GetFeatureValue**(string *featureName*, SapBuffer *featureBuffer*);
bool **GetFeatureValue**(string *featureName*, SapLut *featureLut*);

bool **GetFeatureValue**(int *featureIndex*, out int *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out uint *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out long *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out ulong *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out float *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out double *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out bool *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out string *featureString*);
bool **GetFeatureValue**(int *featureIndex*, SapBuffer *featureBuffer*);
bool **GetFeatureValue**(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 FeatureCount property, minus 1, are valid.
featureValue	Returns the value of the specified feature. You must choose the which function overload to use according to the feature type.
featureString	Returns the content of a string feature
featureBuffer	SapBuffer object for retrieving a buffer feature
featureLut	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 reading the SapFeature.DataType property. 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 the SapFeature.DataAccessMode property.

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.SiUnit and SapFeature.SiToNativeExp10 properties.

Also, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *featureValue* argument.

Demo/Example Usage

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

SapAcqDevice.GetFileIndexByName Method

bool **GetFileIndexByName**(string *fileName*, out int *fileIndex*)

Parameters

fileName	Name of the device file. See the acquisition device User's Manual for the list of supported files.
fileIndex	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 method is useful in building a list of indexes associated with the file names you commonly use. You can then access those device files by index to increase performance.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *fileIndex* argument.

Demo/Example Usage

Not available

SapAcqDevice.GetFileProperty Method

bool **GetFileProperty**(int *fileIndex*, SapAcqDevice.FileProperty *propertyType*,
out SapAcqDevice.FilePropertyVal *propertyValue*)
bool **GetFileProperty**(int *fileIndex*, SapAcqDevice.FileProperty *propertyType*, out ulong *propertyValue*)
bool **GetFileProperty**(string *fileName*, SapAcqDevice.FileProperty *propertyType*,
out SapAcqDevice.FilePropertyVal *propertyValue*)
bool **GetFileProperty**(string *fileName*, SapAcqDevice.FileProperty *propertyType*, out ulong *propertyValue*)

Parameters

fileIndex	Index of the device file. All indices in the range from 0 to the value returned by the FileCount property, minus 1, are valid.	
fileName	Name of the device file	
propertyType	Device file property to inquire, can be one of the following:	
	FileProperty.AccessMode	Access mode for the device file.
	FileProperty.Size	Device file size, in bytes
propertyValue	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:

FilePropertyVal.AccessModeNone
FilePropertyVal.AccessModeReadOnly
FilePropertyVal.AccessModeWriteOnly
FilePropertyVal.AccessModeReadWrite

To find out which device files names are available, use the FileCount property together with the FileNames property.

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

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the propertyValue argument.

Demo/Example Usage

Camera Files Example

SapAcqDevice.GetLut Method

SapLut **GetLut**(string *lutSelector*);

Parameters

lutSelector String identifying the lookup table to modify

Return Value

Returns the specified SapLut object.

Remarks

Returns a SapLut object that contains the current format, number of entries and data of the specified lookup table (LUT).

The *lutSelector* argument specifies the LUT to retrieve. Note that camera LUTs are identified as a string instead of an index, matching the SFNC LUTSelector feature. If the camera has only one valid LUT selector value, you can specify a null value for this argument.

Due to the dynamic nature of camera lookup tables, every call to GetLut will return a different SapLut object. For this reason, make sure to call the GetLut method right before modifying the LUT data with the SapLut class methods, and to call the ApplyLut method right after modification to save the changes.

Demo/Example Usage

Grab LUT Example

SapAcqDevice.GetParameter, SapAcqDevice.SetParameter Methods

bool **GetParameter**(SapAcqDevice.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapAcqDevice.Prm *paramId*, out SapAcqDevice.Val *paramValue*);
bool **GetParameter**(SapAcqDevice.Prm *paramId*, out string *paramValue*);

bool **SetParameter**(SapAcqDevice.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapAcqDevice.Prm *paramId*, SapAcqDevice.Val *paramValue*);
bool **SetParameter**(SapAcqDevice.Prm *paramId*, string *paramValue*);

Parameters

paramId Low-level Sapera parameter to read or write

paramValue Parameter value to read or write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the acquisition device module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter).

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORACQDEVICE_PRM_UPDATE_FEATURE_MODE becomes
SapAcqDevice.Prm.UPDATE_FEATURE_MODE

You can also use the versions of GetParameter/SetParameter which take a SapAcqDevice.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_AUTO becomes
SapAcqDevice.Val.UPDATE_FEATURE_MODE_AUTO

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.

Demo/Example Usage

Not available

SapAcqDevice.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapAcqDevice.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.Int32Array	32-bit integer array
SapCapPrmType.IntPtr	32-bit integer pointer
SapCapPrmType.String	Text string

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapAcqDevice.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapAcqDevice.Cap *capId*);

Parameters

capId Low-level Sapera 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 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. Also see the GetCapability method for more information about the syntax to use for *capId*.

Demo/Example Usage

Not available

SapAcqDevice.IsEventAvailable Method

bool **IsEventAvailable**(string *eventName*);

Parameters

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

Return Value

Returns **true** if the event is available, **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.IsEventEnabled Method

bool IsEventEnabled(string *eventName*);
bool IsEventEnabled(int *eventIndex*);

Parameters

eventName	Name of the event. See the acquisition device User's Manual for the list of supported events.
eventIndex	Index of the event. Valid values for this index are from 0 to the value returned by the EventCount property, minus 1.

Remarks

Checks if the event associated with a specified name or index is enabled. You can only call this method after the Create method.

See the AcqDeviceNotify event for more details.

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice.IsFeatureAvailable

bool **IsFeatureAvailable**(string *featureName*);

Parameters

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

Return Value

Returns **true** if the feature is available, **false** otherwise

Remarks

Checks 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.IsFeatureRecordingStarted Method

bool **IsFeatureRecordingStarted**();

Return Value

Returns **true** if successful, **false** otherwise.

Description

Returns true if feature recording is active. To start recording decive feature setting changes to file use SapAcqDevice.StartFeatureRecording. To pause recording use SapAcqDevice.PauseFeatureRecording. To stop recording use SapAcqDevice.StopFeatureRecording.

Demo/Example Usage

Not available

SapAcqDevice.IsParameterAvailable Method

bool **IsParameterAvailable**(SapAcqDevice.Prm *paramId*);

Parameters

paramId Low-level Sapera 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 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. Also see the GetParameter method for more information about the syntax to use for paramId.

Demo/Example Usage

Not available

SapAcqDevice.Label Property

string **Label** (read-only)

Description

Text description of the acquisition device resource. This property 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 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 gets a SapManager.EventType.ResourceInfoChanged event (if registered using the SapManager.EventType property).

Demo/Example Usage

Not available

SapAcqDevice.LoadFeatures Method

bool **LoadFeatures**(string *configFileName*);

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 contructed with read-only access. See the SapAcqDevice constructor for details.

Demo/Example Usage

Not available

SapAcqDevice.LoadFeatureRecording Method

bool **LoadFeatureRecording**(string *recordFileName*);

Parameters

recordFileName Name of the file containing recorded device feature changes.

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Loads device feature setting changes from a text file and applies the feature setting changes. To record device feature setting changes to file use SapAcqDevice.StartFeatureRecording.

Demo/Example Usage

Camera Files Example

SapAcqDevice.LutEnable Property

bool **LutEnable** (read/write)

Description

Enables or disables the camera lookup table (LUT) module, that is, all available LUT selectors are affected together.

A LUT needs to be enabled in order to affect acquired images.

The initial value for this property is **false**. It is then set according to the current value of the acquisition device lutMode feature when calling the Create method.

When a LUT is disabled, its content is not lost; it may be re-enabled without reprogramming it. Note that some acquisition devices do not support this feature.

Demo/Example Usage

Grab LUT Example

SapAcqDevice.ModeName Property

string **ModeName** (read/write)

Description

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 property is automatically updated.

Demo/Example Usage

Not available

SapAcqDevice.PauseFeatureRecording Method

bool **PauseFeatureRecording**();

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Pauses recording device feature setting changes. To resume recording use SapAcqDevice.ResumeFeatureRecording. Recording is started using SapAcqDevice.StartFeatureRecording. To stop recording use SapAcqDevice.StopFeatureRecording.To check if recording is currently active use SapAcqDevice.IsFeatureRecordingStarted.

Demo/Example Usage

Not available

SapAcqDevice.RawBayerOutput Property

bool **RawBayerOutput** (read only)

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.ReadFile Method

bool **ReadFile**(string *deviceFileName*, string *localFilePath*);
bool **ReadFile**(int *deviceFileIndex*, string *localFilePath*);

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 FileCount property, minus 1, are valid.
localFilePath	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 FileCount property together with the FileNames property.

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

Demo/Example Usage

Camera Files Example

SapAcqDevice.ReadOnly Property

bool **ReadOnly** (read/write)

Description

Checks if the class has read-only access to the device. See the SapAcqDevice contructor for more detail on this option. You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapAcqDevice.ResumeFeatureRecording Method

bool **ResumeFeatureRecording**();

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Resumes recording device feature setting changes following a call to SapAcqDevice.PauseFeatureRecording. To stop recording use SapAcqDevice.StopFeatureRecording. To check if recording is currently active use SapAcqDevice.IsFeatureRecordingStarted.

Demo/Example Usage

Not available

SapAcqDevice.SaveFeatures Method

bool **SaveFeatures**(string *configFileName*);

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.SavedToConfigFile property 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 Method

bool **SetFeatureValue**(string *featureName*, int *featureValue*);
bool **SetFeatureValue**(string *featureName*, uint *featureValue*);
bool **SetFeatureValue**(string *featureName*, long *featureValue*);
bool **SetFeatureValue**(string *featureName*, ulong *featureValue*);
bool **SetFeatureValue**(string *featureName*, float *featureValue*);
bool **SetFeatureValue**(string *featureName*, double *featureValue*);
bool **SetFeatureValue**(string *featureName*, bool *featureValue*);
bool **SetFeatureValue**(string *featureName*, string *featureString*);
bool **SetFeatureValue**(string *featureName*, SapBuffer *featureBuffer*);
bool **SetFeatureValue**(string *featureName*, SapLut *featureLut*);

bool **SetFeatureValue**(int *featureIndex*, int *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, uint *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, long *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, ulong *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, float *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, double *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, bool *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, string *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 FeatureCount property, 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 reading the SapFeature.DataType property. 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 the SapFeature.DataAccessMode property.

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.SiUnit and SapFeature.SiToNativeExp10 properties.

Note that you cannot call this method if the current object was contructed 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.StartFeatureRecording Method

bool **StartFeatureRecording**(string *recordFileName*);

Parameters

deviceFileName	Name of the file to record device features.

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Starts saving device feature setting changes to a text file. This records calls to SapAcqDevice.SetFeatureValue. To load the file and apply the recorded feature setting changes use SapAcqDevice.LoadFeatureRecording. To pause feature recording, use SapAcqDevice.PauseFeatureRecording; to resume feature recording use SapAcqDevice.ResumeFeatureRecording. To stop recording use SapAcqDevice.StopFeatureRecording.To check if recording is currently active use SapAcqDevice.IsFeatureRecordingStarted.

Demo/Example Usage

Camera Files Example

SapAcqDevice.StopFeatureRecording Method

bool **StopFeatureRecording**();

Return Value

Returns **true** if successful, **false** otherwise.

Description

Stops recording device feature setting changes. Recording is started using SapAcqDevice.StartFeatureRecording. To check if recording is currently active use SapAcqDevice.IsFeatureRecordingStarted. To load the file and apply the recorded feature setting changes use SapAcqDevice.LoadFeatureRecording.

Demo/Example Usage

Not available

SapAcqDevice.UpdateFeaturesFromDevice Method

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 using the UpdateMode property. 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 contructed with read-only access. See the SapAcqDevice constructor for details.

This method is only implemented for acquisition devices which are supported through the Network Imaging Package (GigE Vision Framework).

Demo/Example Usage

Not available

SapAcqDevice.UpdateFeaturesToDevice Method

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 using the UpdateMode property. In this mode, writing individual features using the SetFeatureValue method is done to an internal cache. After all the required features have been written to, call this method to update the acquisition device.

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

This method is only implemented for acquisition devices which are supported through the Network Imaging Package (GigE Vision Framework).

Demo/Example Usage

Not available

SapAcqDevice.UpdateMode Property

SapAcqDevice.UpdateFeatureMode **UpdateMode** (read/write)

Description

Mode by which features are written to the acquisition device, which can be one of the following values:

UpdateFeatureMode.Auto	New feature values are immediately sent to the acquisition device
UpdateFeatureMode.Manual	New feature values are temporarily cached before being sent to the acquisition 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 Network Imaging Package (GigE Vision Framework), only UpdateFeature.Auto is implemented; setting the property value to UpdateFeature.Manual has no effect. Consequently, the SapAcqDevice.UpdateFeaturesFromDevice Method and SapAcqDevice.UpdateFeaturesToDevice Method functions are not implemented.

Demo/Example Usage

Not available

SapAcqDevice.UpdateLabel Method

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

Not available

SapAcqDevice.WriteFile Method

bool **WriteFile**(string *localFilePath*, string *deviceFileName*);
bool **WriteFile**(string *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 FileCount property, 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 FileCount property together with the FileNames property.

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

Demo/Example Usage

Camera Files Example

SapAcqDeviceNotifyEventArgs Class

The SapAcqDeviceNotifyEventArgs class contains the arguments to the application handler method for the SapAcqDevice.AcqDeviceNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapAcqDeviceNotifyEventArgs Class Members

Properties

AuxTimeStamp	Time stamp corresponding to the moment when the event occurred on the acquisition device
Context	Application context associated with acquisition device events
CustomData	Data associated with a custom event
CustomSize	Size of the custom data returned by CustomData property
EventCount	Current count of acquisition device events
EventIndex	Index of the event that triggered the invocation of the application event handler
FeatureIndex	Index of the feature associated with the event
GenericParamValue0	Generic properties supported by some events
GenericParamValue1	
GenericParamValue2	
GenericParamValue3	
HostTimeStamp	Time stamp corresponding to the moment when the event occurred on the host

SapAcqDeviceNotifyEventArgs Member Properties

The following are the members of the SapAcqDeviceNotifyEventArgs Class.

SapAcqDeviceNotifyEventArgs.AuxTimeStamp Property

long **AuxTimeStamp** (read-only)

Description

Time stamp 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

SapAcqDeviceNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with acquisition device events. See the AcqDeviceNotifyContext property of the SapAcqDevice class for more details.

Demo/Example Usage

Sequential Grab Demo

SapAcqDeviceNotifyEventArgs.CustomData Property

System.IntPtr **Context** (read-only)

Description

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.

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.CustomSize Property

int **CustomSize (**read-only)

Description

Size of the custom data returned by the CustomData property.

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.EventCount Property

int **EventCount (**read-only)

Description

Current count of acquisition device events. The initial value is 1 and increments every time the event handler method is invoked.

Demo/Example Usage

Sequential Grab Demo

SapAcqDeviceNotifyEventArgs.EventIndex Property

int **EventIndex (**read-only)

Description

Index of the current event. Use this index to retrieve the name of the event using the EventNames property of the SapAcqDevice class.

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDeviceNotifyEventArgs.FeatureIndex Property

int **FeatureIndex** (read-only)

Description

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.FeatureCount property, minus 1.

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDeviceNotifyEventArgs.GenericParamValue0 SapAcqDeviceNotifyEventArgs.GenericParamValue1 SapAcqDeviceNotifyEventArgs.GenericParamValue2 SapAcqDeviceNotifyEventArgs.GenericParamValue3 Properties

int **GenericParamValue0**
int **GenericParamValue1**
int **GenericParamValue2**
int **GenericParamValue3** (read-only)

Description

Any of the four generic properties 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 FeatureIndex property as an alias to GenericParam0. See the acquisition device User's Manual for a list of events using generic properties.

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.HostTimeStamp Property

long **HostTimeStamp** (read-only)

Description

Host CPU 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.

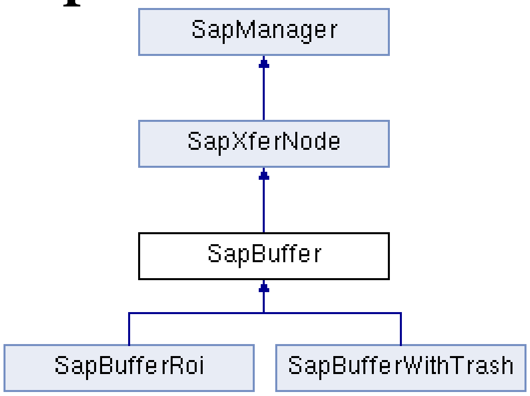
Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

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 Sapera LT User's Manual.

Namespace: DALSA.SaperaLT.SapClassBasic

SapBuffer Class Members

Construction	
SapBuffer	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AllPage	Active page of all the buffer resources for planar or multiformat buffer types
AllState	Empty/full state of all the buffer resources
BufName	Name of a buffer object that is shared between multiple processes
BytesPerPixel	Number of bytes required to store a single buffer element
Count	Number of buffer resources
CounterStamp	Unique identifier associated with a buffer resource
DeviceTimeStamp	Device buffer timestamp
Format	Data format of all the buffer resources
FrameRate	Frame rate of all the buffer resources
Height	Height (in lines) of all the buffer resources
HostCounterStamp	Gets the host counter timestamp at which a specific event occurred.

HostPerformanceCounterStamp	Gets the host performance counter timestamp at which a specific event occurred.
Index	Index of the current buffer resource
Mapped	Indicates if there currently exists a valid virtual data address for a buffer resource
MultiFormat	Checks if the buffer resources are of multiformat type.
NumPages	Gets the number of pages in a buffer resource
Page	Active page of a buffer resource for planar or multiformat buffer types
PageFormats	Gets an array of formats used by the pages of the current buffer resource
Pitch	Number of bytes between two consecutive lines of all the buffer resources
PixelDepth	Number of significant bits of all the buffer resources
SpaceUsed	Number of data bytes actually stored in a buffer resource
State	Empty/full state of the current buffer resource
Type	Type of all the buffer resources
Width	Width (in pixels) of all the buffer resources
Methods	
Clear	Clears the content of all the buffers
ColorConvert	Converts a color image (for example, Bayer format) to RGB format
ColorWhiteBalance	Calculates RGB white balance coefficients for a color image (for example, Bayer format) to be used when converting to RGB format.
Copy	Copies contents of a single buffer resource from another SapBuffer object
CopyAll	Copies contents of all the buffer resources from another SapBuffer object
CopyRect	Copies a rectangular area from a single buffer resource to another buffer resource
GetAddress	Initiates direct address to buffer resource data by a pointer
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetParameter, SetParameter	Gets/sets the count, width, height, format, and type of all the buffer resources
GetParameterType	Gets the data type of a low-level Sapera parameter
IsBufferTypeSupported	Checks if an acquisition resource supports data transfers to a specific buffer type
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
Load	Loads an image file into the current buffer resource
MergeComponents	Merges individual monochrome components into a color buffer or individual multiformat components into a multiformat buffer
Next	Increments the current buffer index
Read	Reads a consecutive series of pixel values in the current buffer resource

ReadElement	Reads the pixel value at a specified position in the current buffer resource
ReadLine	Reads a series of linearly positioned pixel values in the current buffer resource
ReadRect	Reads a series of pixel values from a rectangular area in the current buffer resource
ReleaseAddress	End direct buffer resource data access
ResetIndex	Initializes the current buffer index
Save	Saves the current buffer resource to an image file
SetParametersFromFile	Sets the properties all the buffer resources from an existing file storing a Sapera buffer
SetPhysicalAddress	Sets the physical addresses to use for creating buffer resources
SetVirtualAddress	Sets the virtual addresses to use for creating buffer resources
SplitComponents	Splits a color or multiformat buffer into its individual monochrome components
Write	Writes a consecutive series of pixel values in the current buffer resource
WriteElement	Writes the pixel value at a specified position in the current buffer resource
WriteLine	Writes a series of linearly positioned pixel values to the current buffer resource
WriteRect	Writes a series of pixel values to a rectangular area in the current buffer resource

SapBuffer Member Functions

The following are members of the SapBuffer Class.

SapBuffer.SapBuffer (constructor)

SapBuffer();

SapBuffer(
  int *count*,
  int *width*
  int *height*,
  SapFormat *format*,
  SapBuffer.MemoryType *type*
);

SapBuffer(
  int *count*,
  System.IntPtr[] *physAddress*,
  int *width*,
  int *height*,
  SapFormat *format*
);

SapBuffer(
  int *count*,
  System.IntPtr[] *virtAddress*,
  int *width*,
  int *height*,
  SapFormat *format*,
  SapBuffer.MemoryType *type*
);

```
SapBuffer(
   int count,
   SapXferNode srcNode,
   SapBuffer.MemoryType type
);
SapBuffer(
   string fileName,
   SapBuffer.MemoryType type
);
SapBuffer(
   int count,
   string bufName,
   int width,
   int height,
   SapFormat format,
   SapBuffer.MemoryType type
);
SapBuffer(
   int count,
   string bufName,
   SapXferNode srcNode,
   SapBuffer.MemoryType type
);
SapBuffer(
   string bufName,
   int startIndex,
   int count,
   SapBuffer.MemoryType type
);
```

Parameters

count	Number of buffer resources
width	Width (in pixels) of all the buffer resources
height	Height (in lines) of all the buffer resources
format	Data format of all the buffer resources, can be one of the following values:

Monochrome (unsigned)

SapFormat.Mono1	1-bit
SapFormat.Mono8	8-bit
SapFormat.Mono16	16-bit
SapFormat.Mono32	32-bit

Planar Monochrome

SapFormat.Mono8P2	8-bit Monochrome Planar(2 Planes)
SapFormat.Mono8P3	8-bit Monochrome Planar(3 Planes)
SapFormat.Mono16P2	16-bit Monochrome Planar(2 Planes)
SapFormat.Mono16P3	16-bit Monochrome Planar(3 Planes)

Monochrome (signed)

SapFormat.Int8	8-bit
SapFormat.Int16	16-bit
SapFormat.Int32	32-bit

RGB Color

SapFormat.RGB5551	16-bit (5 for each of red/green/blue, 1for alpha)
SapFormat.RGB565	16-bit (5 for red, 6 for green, 5 for blue)
SapFormat.RGB888	24-bit (8 for red, 8 for green, 8 for blue), blue component is stored first
SapFormat.RGB8888	32-bit (8 for each of red/green/blue, 8 for alpha)
SapFormat.RGB101010	32-bit (10 for each of red/green/blue, 2 unused)
SapFormat.RGB161616	48-bit (16 for each of red/green/blue)

SapFormat.RGB16161616	64-bit (16 for each of red/green/blue/alpha)
SapFormat.RGBP8	8-bit planar
SapFormat.RGBP16	16-bit planar
SapFormat.RGBR888	24-bit (8 for red, 8 for green, 8 for blue), red component is stored first.
SapFormat.RGBAP8	8-bit RGB + Alpha Planar
SapFormat.RGBAP16	16-bit RGB + Alpha Planar
Bi Color	
SapFormat.BICOLOR88	8-bits per component, 32 total.
SapFormat.BICOLOR121212	12-bits per component, 192 total
SapFormat.BICOLOR1616	16-bits per component, 64 total
	For bicolor formats, 1 pixel is generated for 2 components (RG or BG) therefore the buffer width is twice the size of the resulting image.
YUV Color	
SapFormat.UYVY	16-bit, 4:2:2 subsampled
SapFormat.YUY2	16-bit, 4:2:2 subsampled
SapFormat.YVYU	16-bit, 4:2:2 subsampled
SapFormat.YUYV	16-bit, 4:2:2 subsampled
SapFormat.Y411	12-bit, 4:1:1 subsampled
SapFormat.YUV	32-bit (8 for each of Y/U/V, 8 for alpha)
LAB Color	
SapFormat.LAB	32-bit (8 for each component, 8 unused)
SapFormat.LABP8	8-bit Planar(8 for each component, 8 unused)
SapFormat.LABP16	16-bit Planar (16 for each component, 8 unused)
SapFormat.LAB101010	32-bit (10 for each of red/green/blue, 2 unused)
SapFormat.LAB161616	48-bit (16 for each component, 16 unused)
Other Formats	
SapFormat.HSV	32-bit HSV (8 for each component, 8 unused)
SapFormat.HSI	32-bit HSI (8 for each component, 8 unused)
SapFormat.HSIP8	8-bit HSI planar
SapFormat.Float	32-bit signed floating point
SapFormat.Point	64-bit (32-bit signed integer for both X and Y components)
SapFormat.FPoint	64-bit (32-bit signed floating-point for both X and Y components)
Multiformat	
SapFormat.RGB888_ MONO8	32-bit (8 for each of red/green/blue, IR)
SapFormat.RGB161616_MONO16	64-bit (16 for each of red/green/blue/IR)
	Note: Multiformat buffer types do not support color conversion, however the RGB component can be extracted into a supported RGB format using the SapBuffer.Copy or SapBuffer.SplitComponents methods. For load and save operations, multiformat buffers only support the CRC and RAW formats.
3D Formats	
SapFormat.Coord3D_C16	one 16-bit component
SapFormat.Coord3D_AC16	two 16-bit components
SapFormat.Coord3D_ACRW16	four 16-bit components (one reserved)
	3D format buffers are for line profile data. C (z) denotes the height element, A (x) the non-rectified x value for each point, and R the reflectance (intensity) of the peak at the point.
SapFormat.Coord3D_PC_XYZ	Point cloud XYZ floating point coordinates

See also the SapData classes for Sapera data elements described in this document

Type Type of all buffer resources can be one of the following values:

MemoryType. Contiguous	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.

MemoryType. ScatterGather	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. For 32-bit Windows only, if the amount of system memory exceeds 4 GBytes, Sapera automatically uses TypeScatterGatherUnmapped instead.
MemoryType. Virtual	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.
MemoryType. Dummy	Dummy buffers do not have any data memory. They may be used as placeholders by transfer resources when there is no physical data transfer.
MemoryType. Unmapped	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.
MemoryType. ScatterGather Unmapped	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.
MemoryType. ScatterGather Physical	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 (MemoryType.ScatterGather), because they require all physical addresses used during DMA transfers to be limited to 32-bit values.

loc	SapLocation object specifying the server on which the buffer resources are to be created. The resource index of the location object is ignored.
physAddress	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.
virtAddress	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.
srcNode	Source node object. The *width*, *height,* and *format* 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.
fileName	Name of a Sapera image file from which to extract the *count, width*, *height,* and *format* parameters
bufName	Name identifying the buffer object so that it may be shared between multiple processes. The only valid buffer types for this mechanism are MemoryType.ScatterGather and Virtual.
startIndex	Starting index of buffer resource when using a shared buffer object created in another process
display	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. The following examples show how to declare the required arrays for .NET languages:

```
System.IntPtr[] bufAddress = new System.IntPtr[numBuffers];
```

Constructing the object using *srcNode* 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.

Note, for Bayer acquisition the buffer format is either SapFormatMono8 or SapFormatMono16. For bi-color acquisition it is either SapFormat.BICOLOR88, SapFormat.BICOLOR1212 or SapFormat.BICOLOR1616. Refer to the SapColorConversion class for more information on manipulating these buffers.

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

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example

SapBuffer.AllPage Property

int **AllPage** (write-only)

Description

Active page of all the buffer resources for planar buffer types. See also the Page property.

You can only change the value of th property before calling the Create method.

Demo/Example Usage

Not available

SapBuffer.AllState Property

SapBuffer.DataState **AllState** (write-only)

Description

Empty/full state for all buffer resources in the current object. See the State property for a list of possible values.

Demo/Example Usage

Not available

SapBuffer.BufName Property

string **BufName (**read-only)

Description

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, then the value of this property is an empty string.

Demo/Example Usage

Not available

SapBuffer.BytesPerPixel Property

int **BytesPerPixel (**read-only)

Description

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

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

Demo/Example Usage

Not available

SapBuffer.Clear Method

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, SapFormat.RGB888_MONO8 or RGB161616_MONO16) use a SapDataRGBA object.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo

SapBuffer.ColorConvert Method

BOOL **ColorConvert** (SapBuffer *srcBuf*, ColorAlign *align*, ColorMethod *method*, SapDataFRGB *wbCoef*);
BOOL **ColorConvert** (SapBuffer *srcBuf*, ColorAlign *align*, ColorMethod *method*, SapDataFRGB *wbCoef*, SapLut *lut*);
BOOL **ColorConvert** (SapBuffer *srcBuf*, int *srcIndex*, int *dstIndex*, ColorAlign *align*, ColorMethod *method*, SapDataFRGB *wbCoef*);
BOOL **ColorConvert** (SapBuffer *srcBuf*, int *srcIndex*, int *dstIndex*, ColorAlign *align*, ColorMethod *method*, SapDataFRGB *wbCoef*, SapLut *lut*);

Parameters

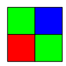
srcBuf Buffer object to convert. The input buffer format must be one of the following:

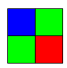
SapFormatUint8
SapFormatUint16

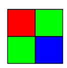
srcIndex Source buffer resource index

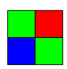
dstIndex Destination buffer resource index in the current object

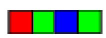
align Specifies the pixel alignment for the color filter. The alignment mode must correspond to the upper left 2x2 square of your camera's color scheme for Bayer conversion; 1x4 line for Bicolor conversion. If the input buffer is a child, the alignment mode is internally recalculated with respect to the upper left corner. Possible values are:

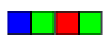
SapBuffer.ColorAlignGBGR 

SapBuffer.ColorAlignBGGR 

SapBuffer.ColorAlignRGGB 

SapBuffer.ColorAlignGRBG 

SapBuffer.ColorAlignRGBG 

SapBuffer.ColorAlignBGRG 

method Specifies the conversion method. Possible values are:

SapBuffer.ColorMethod.Method1	This technique, based on 3x3 bi linear interpolation, is fast but tends to smooth image edges.
SapBuffer.ColorMethod.Method2	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.ColorMethod.Method3	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.ColorMethod.Method4	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.ColorMethod.Method5	This technique (published by IEEE, authors Malvar, He and Cutler), based on a set of linear

	filters, works under the main assumption that edges have much stronger luminance than chrominance component.
SapBuffer.ColorMethod.Method6	Reserved.
SapBuffer.ColorMethod.Method7	Support for bi-color conversion for use with the Teledyne DALSA Piranha 4 camera.

wbCoef White balance coefficients. Can be calculated by 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.

lut Sapera LT LUT object. 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.

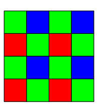
Return Value

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 (SapBuffer.PixelDepth Property) must be set to the number of significant bits.

The white balance coefficients (*wb*Coef) 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 Method of the LUT class).

Demo/Example Usage

Not available

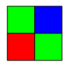
SapBuffer.ColorWhiteBalance Method

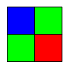
BOOL **ColorWhiteBalance** (ColorAlign *align*, SapDataFRGB *WbCoef*)
BOOL **ColorWhiteBalance** (int *index*, ColorAlign *align*, SapDataFRGB **pWbCoef*);

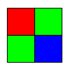
Parameters

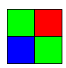
index Index of buffer object to convert. The input buffer format must be one of the following:
SapFormatUint8
SapFormatUint16

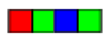
align Specifies the pixel alignment for the color filter. The alignment mode must correspond to the upper left 2x2 square of your camera's color scheme for Bayer conversion; 1x4 line for Bicolor conversion. If the input buffer is a child, the alignment mode is internally recalculated with respect to the upper left corner. Possible values are:

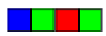
SapBuffer.ColorAlign.GBGR 

SapBuffer.ColorAlign.BGGR 

SapBuffer.ColorAlign.RGGB 

SapBuffer.ColorAlign.GRBG 

SapBuffer.ColorAlignRGBG 

SapBuffer.ColorAlignBGRG 

pWbCoef 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:

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

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

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

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 Method

bool **Copy**(SapBuffer *srcBuf*);
bool **Copy**(SapBuffer *srcBuf*, int *srcIndex*, int *destIndex*);

Parameters

srcBuf	Buffer object to copy from
srcIndex	Source buffer resource index
destIndex	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 supported conversions, see the Image Data Format Conversions section.

For multiformat buffer types (for example, SapFormat.RGB888_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.

For monochrome planar buffer types (for example, SapFormat.Mono8P2, SapFormatMono8P3, SapFormatMono16P2 or SapFormatMono16P3) the copy function can be used to extract the current page to a Mono8 or Mono16 buffer.

For 3D buffer types (SapFormatCoord3D_C16, SapFormatCoord3D_AC16, and SapFormatCoord3D_ACRW16) the copy function can be used to extract the current page to a MONO16 buffer.

Demo/Example Usage

GigE Camera Compression Demo

SapBuffer.CopyAll Method

bool **CopyAll**(SapBuffer *srcBuf*);

Parameters

srcBuf	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. For supported conversions, see the Image Data Format Conversions section.

Demo/Example Usage

Not available

SapBuffer.CopyRect Method

bool **CopyRect**(SapBuffer *srcBuf*, int *srcIndex*, int *xSrc*, int *ySrc*, int *width*, int *height*, int *destIndex*, int *xDest*, int *yDest*);

Parameters

srcBuf	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
destIndex	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.Count Property

int **Count** (read/write)

Description

Number of buffer resources. The initial value for this property is 1, unless you specify another value in the constructor.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

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

SapBuffer.CounterStamp Property

unsigned int **CounterStamp**
unsigned int **get_CounterStamp**(int *index*) (read-only)

Parameters

index Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

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.

The counter stamp value may also be expressed in other units. For framegrabbers, see the SapXferPair.CounterStampTimeBase property for details; for other acquisition devices, refer to the device documentation.

Even though the property 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.

There are two versions of this property: one without an index, and one with an index. The former automatically uses the current buffer resource index. The latter allow this index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
int stamp = buffer.get_CounterStamp(index);
```

Note that some transfer devices do not support this feature.

Demo/Example Usage

Sequential Grab Demo

SapBuffer.Create Method

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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example

SapBuffer.Destroy Method

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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example

SapBuffer.DeviceTimeStamp Property

unsigned long **DeviceTimeStamp**(); (read-only)
unsigned long **get_DeviceTimeStamp**(int *index*)

Parameters

index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Device timestamp at which a specific event occurred, such as the end or start of frame. This value is determined by the timebase of the device. Subtracting timestamp values for two buffers gives the amount of time elapsed between a common reference point.

This read-only parameter is only available from GigE acquisition devices which set their values after an image has been acquired in the buffer, otherwise it will return false if not supported.

Demo/Example Usage

Not available

SapBuffer.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapBuffer .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapBuffer object anymore. If you do, you get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console Example, GigE Auto-White Balance Example, GigE Camera LUT Example, GigE CameraLink Example, Grab LUT Example

SapBuffer.Format Property

SapFormat **Format** (read/write)

Description

Data format of all the buffer resources.

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

> If using the constructor with a SapXferNode object, then this value is SapFormat.Unknown, 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 SapFormat.Unknown. It is then set correctly from the shared buffer object after calling the Create method.
>
> Otherwise, the initial value is equal to SapFormat.Mono8.

You can only change the value of this property before calling the Create method. See the SapBuffer constructor for possible values for *format* (other than SapFormat.Unknown).

Demo/Example Usage

Not available

SapBuffer.FrameRate Property

float **FrameRate** (read/write)

Description

Frame rate for all buffer resources. 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 by reading the value of this property.

When saving a buffer sequence you may optionally save the frame rate. To do so you must specify a new value for this property 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.GetAddress Method

bool **GetAddress**(out System.IntPtr *dataAddress*);
bool **GetAddress**(System.IntPtr *virtualBaseAddress*, out System.IntPtr *dataAddress*);
bool **GetAddress**(int *index*, out System.IntPtr *dataAddress*);
bool **GetAddress**(int *index*, System.IntPtr *virtualBaseAddress*, out System.IntPtr *dataAddress*);
bool **GetAddress**(long *offset*, System.IntPtr *size*, out System.IntPtr *dataAddress*);
bool **GetAddress**(long *offset*, System.IntPtr *size*, System.IntPtr *virtualBaseAddress*,
out System.IntPtr *dataAddress*);
bool **GetAddress**(int *index*, long *offset*, System.IntPtr *size*, out System.IntPtr *dataAddress*);
bool **GetAddress**(int *index*, long *offset*, System.IntPtr *size*, System.IntPtr *virtualBaseAddress*,
out System.IntPtr *dataAddress*);

Parameters

dataAddress	Buffer data address to retrieve
virtualBaseAddress	Starting address of a memory area already reserved by the application
index	Buffer resource index
offset	Byte offset from the 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 properties: Width, Height, Format, PixelDepth, BytesPerPixel, and Pitch.

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 MemoryType.Unmapped or MemoryType.ScatterGatherUnmapped, 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.

Here are examples of how to get the current buffer address for .NET languages:

```
System.IntPtr address;
result = buffer.GetAddress(out address);
```

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *eventIndex* argument.

Demo/Example Usage

Color Split Example

SapBuffer.GetCapability Method

bool **GetCapability**(SapBuffer.Cap *capId*, out int *capValue*);
bool **GetCapability**(int *index*, SapBuffer.Cap *cap*, out int *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	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 capabilities for the buffer module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapBuffer class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORBUFFER_CAP_PIXEL_DEPTH becomes SapBuffer.Cap.PIXEL_DEPTH

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

Demo/Example Usage

Not available

SapBuffer.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapBuffer.Cap *capID*);

Parameters

capID	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapBuffer.GetParameter, SapBuffer.SetParameter Methods

bool **GetParameter**(SapBuffer.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(int *index*, SapBuffer.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapBuffer.Prm *paramId*, out long *paramValue*);
bool **GetParameter**(int index, SapBuffer.Prm *paramId*, out long *paramValue*);
bool **GetParameter**(SapBuffer.Prm *paramId*, out double paramValue);
bool **GetParameter**(int index, SapBuffer.Prm *paramId*, out double *paramValue*);
bool **GetParameter**(SapBuffer.Prm *paramId*, int[] *paramValue*);
bool **GetParameter**(int index, SapBuffer.Prm *paramId*, int[] *paramValue*);

bool **GetParameter**(SapBuffer.Prm *paramId*, out SapBuffer.Val *paramValue*);
bool **GetParameter**(int *index*, SapBuffer.Prm *paramId*, out SapBuffer.Val *paramValue*);
bool **GetParameter**(SapBuffer.Prm *paramId*, out System.IntPtr *paramValue*);
bool **GetParameter**(int *index*, SapBuffer.Prm *paramId*, out System.IntPtr *paramValue*);

bool **SetParameter**(SapBuffer.Prm *paramId*, int *paramValue*);
bool **SetParameter**(int *index*, SapBuffer.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapBuffer.Prm *paramId*, double *paramValue*);
bool **SetParameter**(int *index*, SapBuffer.Prm *paramId*, double *paramValue*);
bool **SetParameter**(SapBuffer.Prm *paramId*, SapBuffer.Val *paramValue*);
bool **SetParameter**(int *index*, SapBuffer.Prm *paramId*, SapBuffer.Val *paramValue*);

Parameters

paramId	Low-level Sapera C library parameter to read or write
paramValue	Parameter value to read or write
index	Buffer resource index

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the buffer module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.IntPtr, then *paramValue* is an address (uninitialized for GetParameter).

The following examples show how to declare a variable to hold an address and call GetParameter:

```
System.IntPtr paramValue;
result = buf.GetParameter(paramId, out paramValue)
```

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, refer to the *Sapera LT Basic Modules Reference Module* (available upon request) for a description of all available parameters and their possible values. for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORBUFFER_PRM_TYPE becomes SapBuffer.Prm.TYPE

You can also use the versions of GetParameter/SetParameter which take a SapBuffer.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORBUFFER_VAL_TYPE_SCATTER_GATHER becomes SapBuffer.Val.TYPE_SCATTER_GATHER

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 SapBuffer class. Also, directly setting parameter values may interfere with the correct operation of the class.

For example, low-level C parameters for 3D buffers include:

CORBUFFER_PRM_SCAN3D_COORD_SCALE_A
CORBUFFER_PRM_SCAN3D_COORD_SCALE_B
CORBUFFER_PRM_SCAN3D_COORD_SCALE_C
Scale factor (data type = double) when transforming a pixel from relative coordinates to world coordinates. A negative scale mirrors the axis. For rectified image axes it is the distance between samples in the rectified image along this axis.

Note: Coordinate A is considered the X or Theta coordinate, B the Y or Phi coordinate and C is the Z or Rho coordinate.

CORBUFFER_PRM_SCAN3D_COORD_OFFSET_A
CORBUFFER_PRM_SCAN3D_COORD_OFFSET_B
CORBUFFER_PRM_SCAN3D_COORD_OFFSET_C
Offset (data type = double) when transforming a pixel from relative coordinates to world coordinates.

CORBUFFER_PRM_SCAN3D_INVALID_DATA_FLAG_C
Enables (data type = boolean) the definition of a non-valid flag value in the data stream. Using an invalid data value may increase processing time due to special handling.

CORBUFFER_PRM_SCAN3D_INVALID_DATA_VALUE_C
Value (data type = double) which identifies a non-valid pixel if CORBUFFER_PRM_SCAN3D_INVALID_DATA_FLAG_C is enabled. The invalid data is flagged in coordinate C (Z/Rho). If the pixel format is an integer the invalid value must be mapped to (rounded to) an available integer in the device pixel range.

CORBUFFER_PRM_SCAN3D_DISTANCE_UNIT
Specifies the unit used when delivering (calibrated) distance data. Possible values are acquisition device dependent. Device specific values can be used to indicate other meaning to distance data.

CORBUFFER_PRM_DEVICE_SCAN_TYPE
Buffer device scan type (data type = UINT32).The value represents the device scan type. Possible values are:

CORBUFFER_VAL_DEVICE_SCAN_TYPE_UNKNOWN (0x00000000)
CORBUFFER_VAL_DEVICE_SCAN_TYPE_AREASCAN (0x00000001)
CORBUFFER_VAL_DEVICE_SCAN_TYPE_LINESCAN (0x00000002)

CORBUFFER_VAL_DEVICE_SCAN_TYPE_AREASCAN3D (0x00000004)
CORBUFFER_VAL_DEVICE_SCAN_TYPE_LINESCAN3D (0x00000008)

The device scan type is only available after an image has been acquired in the buffer, otherwise CORSTATUS_PRM_NOT_AVAILABLE is returned when trying to read the value.

CORBUFFER_PRM_SCAN3D_OUTPUT_MODE
Saves the 3d output mode of the acquisition device when an image is acquired in the buffer. Possible values (data type = UINT32):

CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_UNKNOWN (0x00000000)
CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_UNCALIBRATED_C (0x00000001)
CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_CALIBRATED_C (0x00000002)
CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_RECTIFIED_C (0x00000008)
CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_CALIBRATED_AC (0x00000020)
CORBUFFER_VAL_SCAN3D_OUTPUT_MODE_CALIBRATED_ACRW (0x00000080)

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

Demo/Example Usage

Sequential Grab Demo, GigE Sequential Grab Demo

SapBuffer.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapBuffer.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.IntPtr	Address value
SapCapPrmType.Int64	64-bit integer
SapCapPrmType.Double	floating-point double
SapCapPrmType.Int32Array	32-bit integer array

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapBuffer.Height Property

int **Height** {read/write}

Description

Height (in lines) 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, changing the value of this property 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 change the value of the Height property before calling the Create method.

Demo/Example Usage

Multi-Board Sync Grab Demo, Color Split Example, GigE Auto-White Balance Example

SapBuffer.HostCounterStamp Property

unsigned long **HostCounterStamp**
unsigned long **get_HostCounterStamp**(int *index*) (read-only)

Parameters

index Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

Host counter timestamp at which a specific event occurred. For GigE cameras, this timestamp is at the reception of the first image packet of the buffer; for frame grabbers, it is at the end of frame event.

Subtracting counter stamp values for two buffers gives the amount of time elapsed between a common reference point for their respective data transfers.

For Teledyne DALSA GigE cameras this value is in microseconds. This value is converted from the high –resolution performance counter. To get the high-performance counter value in CPU clock ticks use the SapBuffer.HostPerformanceCounterStamp property.

Under Windows, refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on retreiving the host high-performance counter.

The following static function is used by the Teledyne DALSA GigE-Vision driver to convert from counter ticks to microseconds:

```
static UINT64 convertTickInMicroSecond(LARGE_INTEGER tickCount, LARGE_INTEGER tickPerSecond)
{
   LARGE_INTEGER currentTickCount;
   LONGLONG seconds;
   currentTickCount.QuadPart = tickCount.QuadPart;
   // Arithmetic must be done as follows so that will not overflow when perfFreq is very high
   seconds = currentTickCount.QuadPart / (ULONGLONG)tickPerSecond.QuadPart;
   currentTickCount.QuadPart %= tickPerSecond.QuadPart;
   currentTickCount.QuadPart *= 1000000;
   currentTickCount.QuadPart /= (ULONGLONG)tickPerSecond.QuadPart;
   currentTickCount.QuadPart += 1000000 * seconds;
   return currentTickCount.QuadPart;
}
```

For Teledyne DALSA frame grabbers, this value is determined by the timebase of the CPU clock.

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.

Here are examples of how to use the indexed property for .NET languages:

```
int stamp = buffer.get_HostCounterStamp(index);
```

Note that most transfer devices do not support this feature.

Demo/Example Usage

Not available

SapBuffer.HostPerformanceCounterStamp Property

unsigned long **HostPerformanceCounterStamp**
unsigned long **get_HostPerformanceCounterStamp**(int *index*) (read-only)

Parameters

index	Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

Host performance counter timestamp at which a specific event occurred; for GigE cameras, this timestamp is at the reception of the first image packet of the buffer.

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. For Teledyne DALSA GigE cameras, the SapBuffer.HostCounterStamp property returns the timestamp in microseconds.

Note, this function is not supported by Teledyne DALSA frame grabbers; use the SapBuffer.HostCounterStamp property to return the high performance counter value in ticks.

Here are examples of how to use the indexed property for .NET languages:

```
int stamp = buffer.get_HostPerformanceCounterStamp(index);
```

Note that most transfer devices do not support this feature.

Demo/Example Usage

Not available

SapBuffer.Index Property

int **Index** (read/write)

Description

Index of the current buffer. The value of this property 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.IsBufferTypeSupported Method

static bool **IsBufferTypeSupported**(int *serverIndex*, SapBuffer.MemoryType *bufType*);
static bool **IsBufferTypeSupported**(string *serverName*, SapBuffer.Memory ype *bufType*);
static bool **IsBufferTypeSupported**(SapLocation *location*, SapBuffer.MemoryType *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
location	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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigESequential Grab Demo, Grab Demo, Sequential Grab Demo

SapBuffer.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapBuffer.Cap *cap*);

Parameters

cap	Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.IsParameterAvailable Method

bool **IsParameterAvailable**(SapBuffer.Prm *param*);

bool **IsParameterAvailable**(int *index*, SapBuffer.Prm *param*);

Parameters

index	Buffer resource index
param	Low-level Sapera parameter to check

Return Value

Returns **true** if the parameter is supported, **false** otherwise

Remarks

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

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

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 Method

bool **Load**(string *fileName*, int *bufIndex*);
bool **Load**(string *fileName*, int *bufIndex,* string *options*);
bool **Load**(string *fileName*, int *bufIndex,* int *numBuffers*, int *frameIndex*);
bool **Load**(string *fileName*, int *bufIndex,* int *numBuffers*, int *frameIndex*, string *options*);

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 ccor"	Color calibration coeffiecients file format (*.ccor)
"-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.

This function also supports loading color calibration coefficient files (*.ccor) to devices that support hardware color correction. For *.ccor files, the buffer size is 4x3 and type SapFormatFloat.

SapBuffer.Mapped Property

bool **Mapped**
bool **get_Mapped**(int *index*) (read-only)

Parameters

index Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

Indicates if there currently exists a valid virtual data address for a buffer resource.

This property is only relevant for buffers that are MemoryType.Unmapped or MemoryType.ScatterGatherUnmapped. In this case, the GetAddress method sets up a valid virtual address mapping, and ReleaseAddress ends it. For all other buffer types, this property is always **true**.

There are two versions of this property: one without an index, and one with an index. The former automatically uses the current buffer resource index. The latter allow this index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
bool mapped = buffer.get_Mapped(index);
```

Demo/Example Usage

Not available

SapBuffer.MergeComponents Method

bool **MergeComponents**(SapBuffer *srcBuf*);
bool **MergeComponents**(SapBuffer *srcBuf*, int *destIndex*);
bool **MergeComponents**(SapBuffer *firstSrc*, SapBuffer *secondSrc*, SapBuffer *thirdSrc*)
bool **MergeComponents**(SapBuffer *firstSrc*, SapBuffer *secondSrc*, SapBuffer *thirdSrc*, nt *destIndex*)
bool **MergeComponents**(SapBuffer[] *src*, int *srcCount*);
bool **MergeComponents**(SapBuffer[] *src*, int *srcCount*, int *destIndex*);

Parameters

srcBuf	Source monochrome buffer object
src	Array of source monochrome buffer objects
srcCount	Number of source monochrome buffer objects in *src* array
firstSrc	First source buffer object.
secondSrc	Second source buffer object.
thirdSrc	Third source buffer object.
destIndex	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 *destIndex* 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 SapFormat.Mono8. If color components have more than 8-bits, then the input format must be SapFormat.Mono16.

For multiformat buffers (for example, SapFormat.RGB888_ MONO8 or SapFormat.RGB161616_MONO16), the function prototype with 3 source 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.

For monochrome planar buffer types (for example, SapFormat.Mono8P2, SapFormatMono8P3, SapFormatMono16P2 or SapFormatMono16P3) the function prototype with 3 source buffers or a buffer array is used to merge the source buffers (MONO8 or MONO16) into the current buffer object (for 2 plane formats the 3rd destination buffer is ignored).

For monochrome planar buffer types with more than 3 planes (for example, SapFormatMono8P4 or SapFormatMono16P4) the function prototype with a source buffer array is used to merge the source buffers (MONO8 or MONO16) into the current buffer object.

For 3D buffer types (for example, SapFormat.Coord3D_AC16 or SapFormat.Coord3D_ACRW16) the function prototype with 3 source buffers or a buffer array is used to merge the source buffers (MONO16) into the current buffer object (for 2 plane formats the 3rd destination buffer is ignored). For SapFormatCoord.3D_PC_XYZ, SapFormat.Float buffers for each component can be merged.

Demo/Example Usage

Color Split Example

SapBuffer.MultiFormat

BOOL **MultiFormat**();

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.

Monochrome planar buffer types (for example, SapFormat.Mono8P2, SapFormatMono8P3, SapFormatMono16P2 or SapFormatMono16P3) also use this method.

The SapBuffer.Copy and SapBuffer.SplitComponents methods can extract the RGB or IR (monochrome) components into separate buffers.

Use the Page and AllPage properties to manage the current page of the buffer. This only applies when choosing what format to display when calling the SapView.Show Method.

Demo/Example Usage

Not available

SapBuffer.Next Method

void **Next**();

Remarks

Increments the current buffer index. The SapTransfer class calls the Next method 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.NumPages Property

int **NumPages**();

Return Value

Returns the number of pages in the current buffer resource.

Remarks

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.Page Property

int **Page**
int **get_Page**(int *index*)
void **set_Page**(int *index*, int *page*) (read/write)

Parameters

page	New page number for the buffer resource
index	Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

Active page (or plane) of a 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 monochrome 2-plane planar (SapFormat.MONO8P2), 8-bit RGB planar (SapFormat.RGBP8), or multiformat buffer types such as SapFormat.RGB888_MONO8 or RGB161616_MONO16.

The active page usually 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.

For planar monochrome and RGB buffer formats (for example, SapFormat.Mono8P2 and SapFormat.RGBP8), the following functions from the SapBuffer class also use the current page: Read, Write, ReadElement, and WriteElement.

There are two versions of this property: one without an index, and one with an index. The former automatically uses the current buffer resource index. The latter allow this index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
int page = buffer.get_Page(index);
buffer.set_Page(index, page);
```

Demo/Example Usage

Not available

SapBuffer.PageFormats Property

SapFormat[] **PageFormats**()

Description

Gets the individual formats included in the current buffer resource as a list of SapFormat entries. The list terminates upon reaching a format with a value of 0, and should contain a number of formats equals to the value returned by the NumPages property.

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 (SapFormat.HSIP8), as well as multi-format buffer types such as SapFormat.RGB888_MONO8 and SapFormat.RGB161616_MONO16. Currently supported individual formats for multi-format buffer types are SapFormat.RGB888, SapFormat.Mono8, SapFormat.RGB161616, and SapFormat.Mono16.

Demo/Example Usage

Not available

SapBuffer.Pitch Property

int **Pitch** (read-only)

Description

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 property after calling the Create method.

Demo/Example Usage

Not available

SapBuffer.PixelDepth Property

int **PixelDepth** (read/write)

Description

Number of significant bits of all the buffer resources. The range of possible values is given by the SapManager.GetPixelDepthMin and SapManagerGetPixelDepth.Max methods.

The value of this property 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.Read Method

bool **Read**(long *pixelOffset*, int *numPixels*, System.IntPtr *bufData*);
bool **Read**(int *index*, long *pixelOffset*, int *numPixels*, System.IntPtr *bufData*);

Parameters

offset	Starting position within the buffer (in pixels)
numPixels	Number of pixels to read
bufData	Memory address to receive 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 BytesPerPixel property.

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

For multiformat or planar buffer types the current page is used.

Note, reading elements from video memory buffers may be very slow.

Demo/Example Usage

Not available

SapBuffer.ReadElement Method

bool **ReadElement**(int *x*, int *y*, System.IntPtr *value*);
bool **ReadElement**(int *index*, int *x*, int *y*, System.IntPtr *value*);
bool **ReadElement**(int *x*, int *y*, SapData *value*);
bool **ReadElement**(int *index*, int *x*, int *y*, SapData *value*);

Parameters

x	Horizontal position
y	Vertical position
value	Memory address to receive the pixel value, or 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 BytesPerPixel property.

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, SapFormat.RGB888_ MONO8 or RGB161616_MONO16) use a SapDataRGBA object. Function prototypes using a 'IntPtr' data argument use values formatted as B/G/R/Mono. For RGB888_ MONO8 buffers, this a 32-bit value. For RGB161616_MONO16 buffers, this is a 64-bit value.

Demo/Example Usage

Not available

SapBuffer.ReadLine Method

bool **ReadLine**(int *x1*, int *y1*, int *x2*, int *y2*, System.IntPtr *bufData,* out int *numRead*);
bool **ReadLine**(int *index*, int *x1*, int *y1*, int *x2*, int *y2*, System.IntPtr *bufData,* out int *numRead*);

Parameters

x1	Starting horizontal position
y1	Starting vertical position
x2	Ending horizontal position
y2	Ending vertical position
bufData	Memory address to receive 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 BytesPerPixel property.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *numRead* argument.

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 Method

bool **ReadRect**(int *x*, int *y*, int *width*, int *height*, System.IntPtr *bufData*);
bool **ReadRect**(int *index*, int *x*, int *y*, int *width*, int *height*, System.IntPtr *bufData*);

Parameters

x	Left coordinate of rectangle origin
y	Top coordinate of rectangle origin
width	Rectangle width
height	Rectangle height
bufData	Memory address to receive pixel values
index	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) >> 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 BytesPerPixel property.

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

For multiformat or planar buffer types the current page is used.

Reading elements from video memory buffers may be very slow.

Demo/Example Usage

Not available

SapBuffer.ReleaseAddress Method

bool **ReleaseAddress**(System.IntPtr *dataAddress*);
bool **ReleaseAddress**(int *index*);

Parameters

dataAddress	Buffer data address to release
index	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 MemoryType.Unmapped or MemoryType.ScatterGatherUnmapped, 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 Method

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 Method

bool **Save**(string *fileName*, string *options*);
bool **Save**(string *fileName*, string *options*, int *bufIndex*, int *numBuffers*);

Parameters

fileName	Name of the image file to save
options	String containing the saving options. The following are supported:

"-format bmp"	Window bitmap format
"-format tiff -compression [none/rle/lzw/jpeg] -quality [value]"	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.
"-format jpeg -quality [value]"	JPEG format. The quality level may vary between 1 and 100.
"-format jpeg_2000 -quality [value]"	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
'-format stl''	STL format
'-format ply'	PLY format
'-format pcd'	Format defined inthe Point Cloud Library (PCL)
'-format vtu'	Format for the VTK open source library
'-format vtp'	Format for the VTK open source library

bufIndex	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.
numBuffers	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 SapFormat.RGB888_MONO8 and RGB161616_MONO16, only support saving in CRC or RAW format.

Note, the Save method does not currently support saving color calibration coefficient files (*.ccor). The Sapera Color Calibration tool can be used to save color calibration coefficient files to disk.

Demo/Example Usage

Not available

SapBuffer.SetParametersFromFile Method

bool **SetParametersFromFile**(string *fileName*, SapBuffer.MemoryType *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 Method

bool **SetPhysicalAddress**(System.IntPtr[] *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.SetVirtualAddress Method

bool **SetVirtualAddress**(System.IntPtr[] *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

Not available

SapBuffer.SpaceUsed Property

int **SpaceUsed**
int **get_SpacedUsed**(int *index*) (read-only)

Parameters

index	Buffer resource index (from 0 to the value returned by the Count property, minus 1)

Description

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 this value is 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.

There are two versions of this property: one without an index, and one with an index. The former automatically uses the current buffer resource index. The latter allow this index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
int spaceUsed = buffer.get_ SpaceUsed (index);
```

Demo/Example Usage

Not available

SapBuffer.SplitComponents Method

bool **SplitComponents**(SapBuffer *srcBuf*);
bool **SplitComponents**(SapBuffer *srcBuf*, int *srcIndex*);
bool **SplitComponents**(SapBuffer *firstDst*, SapBuffer *secondDst*, SapBuffer *thirdDst*);
bool **SplitComponents**(SapBuffer *firstDst*, SapBuffer *secondDst*, SapBuffer *thirdDst*, int *srcIndex*);
bool **SplitComponents**(SapBuffer[] *dst*, int *dstCount*);
bool **SplitComponents**(SapBuffer[] *dst*, int *dstCount*, int *srcIndex*);

Parameters

srcBuf	Source color buffer object
firstDst	First destination buffer object
secondDst	Second destination buffer object
thirdDst	Third destination buffer object
dst	Array of destination monochrome buffer objects
dstCount	Number of destination monochrome buffer objects in dst array
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. 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 SapFormat.Mono8. If color components have more than 8-bits, then the output format must be SapFormat.Mono16.

For multiformat buffers (for example, SapFormat.RGB888_ MONO8 or tRGB161616_MONO16), 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 buffers; the 3rd destination buffer is ignored.

For monochrome planar buffer types (for example, SapFormat.Mono8P2, SapFormatMono8P3, SapFormatMono16P2 or SapFormatMono16P3) the function prototype with 3 destination buffers (for 2 plane formats the 3rd destination buffer is ignored) or a buffer array is used to extract the mono components.

For monochrome planar buffer types with more than 3 planes(for example, SapFormatMono8P4 or SapFormatMono16P4) the source buffer is the current object; the function prototype with a destination buffers array is used to extract the mono components.

For 3D buffer types (for example, SapFormat.Coord3D_AC16 or SapFormat.Coord3D_ACRW16) the source buffer is the current object; the function prototype with 3 destination buffers (for 2 plane formats the 3rd destination buffer is ignored) or a buffer array is used to extract the components. For SapFormat.Coord3D_PC_XYZ, SapFormat.Float buffers for each component can be extracted.

Demo/Example Usage

Color Split Example

SapBuffer.State Property

SapBuffer.DataState **State**
SapBuffer.DataState **get_State**(int *index*)
void **set_State**(int *index*, SapBuffer.DataState *state*) (read/write)

Parameters

state	New buffer state may be one of the following:	
	DataState.Empty	The buffer is ready to receive new data
	DataState.Full	The buffer contains unprocessed data
	DataState.Overflow	The buffer contains incorrect data due to insufficient hardware bandwidth. This state can only occur when DataState.Full is active (the two values are combined using a bitwise OR).
index	Buffer resource index (from 0 to the value returned by the Count property, minus 1)	

Description

Empty/full state which indicates whether the specified buffer is ready to accept a new image, or currently contains unprocessed data.

There are two versions of this property: one without an index, and one with an index. The former automatically uses the current buffer resource index. The latter allow this index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
SapBuffer.DataState state = buffer.get_State(index);
buffer.set_State(index, state);
```

Note that Sapera LT automatically manages the buffer state by default, so that you rarely have change the value of the State property directly. If you wish to perform this management yourself, you must set the value of the SapTransfer.AutoEmpty property to **false**.

Demo/Example Usage

Not available

SapBuffer.Type Property

SapBuffer.MemoryType **Type** (read/write)

Description

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 should be MemoryType.Contiguous.
>
> If using the constructor with virtual addresses, then this value should be MemoryType.ScatterGather.
>
> If using the constructor with a shared buffer object with width/height/format, then this value should also be MemoryType.ScatterGather.
>
> If using the constructor with a shared buffer object with a starting index and count, then this value should be MemoryType.Virtual.
>
> Otherwise, the initial value is equal to MemoryType.Default. This is not a real buffer type, but rather a placeholder which specifies that the most appropriate type should be automatically determined and used when calling the Create method. If MemoryType.ScatterGather is supported by the current acquisition hardware (by far the most common case), then it is used. Otherwise MemoryType.ScatterGatherPhysical is used if supported, otherwise MemoryType.Contiguous is used.

You can only change the value of this property before calling the Create method. See the SapBuffer constructor for possible values.

Demo/Example Usage

Color Split Example

SapBuffer.Width Property

int **Width (**read/write)

Description

Width (in pixels) of all the buffer resources.

There are many possible initial values for this propertiy, 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, changing the value of this property 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 change the value of the Width property before calling the Create method.

Demo/Example Usage

Color Split Example, GigE Auto-White Balance Example

SapBuffer.Write Method

bool **Write**(long *offset*, int *numPixels*, System.IntPtr *bufData*);
bool **Write**(int *index*, long *offset*, int *numPixels*, System.IntPtr *bufData*);

Parameters

offset	Starting position within the buffer (in pixels)
numPixels	Number of pixels to write
bufData	Source memory address 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 ((*numPixels* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes of at least *numPixels* times the value returned by the BytesPerPixel property.

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

For multiformat or planar buffer types the current page is used.

Writing elements to video memory buffers may be very slow.

Demo/Example Usage

Not available

SapBuffer.WriteElement Method

bool **WriteElement**(int *x*, int *y*, System.IntPtr *value*);
bool **WriteElement**(int *index*, int *x*, int *y*, System.IntPtr *value*);
bool **WriteElement**(int *x*, int *y*, SapData *value*);
bool **WriteElement**(int *index*, int *x*, int *y*, SapData *value*);

Parameters

X	Horizontal position
Y	Vertical position
Value	Source memory address for the pixel value, 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 BytesPerPixel property.

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

Writing elements to video memory buffers may be very slow.

For multiformat or planar buffer types the current page is used.

Multiformat buffers (for example, SapFormat.RGB888_ MONO8 or RGB161616_MONO16) use a SapDataRGBA object. Function prototypes using an 'IntPtr' data argument use values formatted as B/G/R/Mono. For 8-bit buffers, this a 32-bit value. For 16-bit buffers, this is a 64-bit value.

Demo/Example Usage

Not available

SapBuffer.WriteLine Method

bool **WriteLine**(int *x1*, int *y1*, int *x2*, int *y2*, System.IntPtr *bufData,* out int *numWritten*);
bool **WriteLine**(int *index*, int *x1*, int *y1*, int *x2*, int *y2*, System. IntPtr *bufData,* out int *numWritten*);

Parameters

x1	Starting horizontal position
y1	Starting vertical position
x2	Ending horizontal position
y2	Ending vertical position
bufData	Source memory address 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 BytesPerPixel property.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *numWritten* argument.

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

WriteLine does not support 1-bit buffers.

For multiformat or planar buffer types the current page is used.

Writing elements to video memory buffers may be very slow.

Demo/Example Usage

Not available

SapBuffer.WriteRect Method

bool **WriteRect**(int *x*, int *y*, int *width*, int *height*, System.IntPtr *bufData*);
bool **WriteRect**(int *index*, int *x*, int *y*, int *width*, int *height*, System.IntPtr *bufData*);

Parameters

x	Left coordinate of rectangle origin
y	Top coordinate of rectangle origin
width	Rectangle width
height	Rectangle height
bufData	Source memory address for pixel values
index	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 ((*numPixelss* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numPixels* times the value returned by the BytesPerPixel property.

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

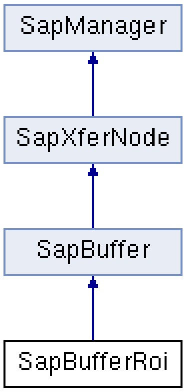
For multiformat or planar buffer types the current page is used.

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapBufferRoi Class Members

Construction	
SapBufferRoi	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
Parent	Parent SapBuffer object for the ROI
Root	Topmost SapBuffer object for the ROI
X	Width (in pixels) for the ROI
Y	Height (in pixels) for the ROI
Methods	
ResetRoi	Sets the ROI origin and dimensions to default values
SetRoi	Sets the ROI origin and dimensions in one step

SapBufferRoi Member Functions

The following are members of the SapBufferRoi Class.

SapBufferRoi.SapBufferRoi (constructor)

SapBufferRoi(SapBuffer *parent*);
SapBufferRoi(SapBuffer *Parent*, int *x*, int *y*, int *width*, int *height*);

Parameters

parent	SapBuffer object that represents the parent for the current SapBufferRoi object
x	Left origin for the ROI, relative to the parent object
y	Top origin for the ROI, relative to the parent object
width	Width (in pixels) of the ROI
height	Height (in lines) of the ROI

Remarks

The SapBufferRoi constructor sets up a rectangular Region of Interest (ROI) inside the SapBuffer object identified by *parent*. 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

Multi-Board Sync Grab Demo

SapBufferRoi.Create Method

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 Method.

Demo/Example Usage

Multi-Board Sync Grab Demo

SapBufferRoi.Destroy Method

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

Multi-Board Sync Grab Demo

SapBufferRoi.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapBufferRoi .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapBufferRoi object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Multi-Board Sync Grab Demo

SapBufferRoi.Parent Property

SapBuffer **Parent (**read/write)

Description

Parent SapBuffer object for the ROI. Note that you can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapBufferRoi.ResetRoi Method

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.Root Property

SapBuffer **Root** (read-only)

Description

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 the Parent property repeatedly until we reach the top.

Demo/Example Usage

Not available

SapBufferRoi.SetRoi Method

bool **SetRoi**(int *xmin*, int *ymin,* int *width*, int *height*);

Parameters

xmin	Left origin for the ROI, relative to the parent object
ymin	Top origin for the ROI, relative to the parent object
width	Width (in pixels) of the ROI
height	Height (in lines) 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

Multi-Board Sync Grab Demo

SapBufferRoi.X Property

int **X** (read/write)

Description

Width (in pixels) for the ROI. If it has not been specified in the constructor, the value of this property is set to the parent buffer width when calling the Create method.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapBufferRoi.Y Property

int **Y** (read/write)

Description

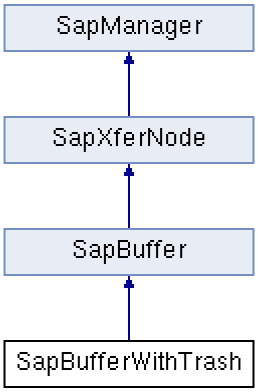
Height (in lines) for the ROI. If it has not been specified in the constructor, the value of this attribute is set to the parent buffer height when calling the Create method.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapBufferWithTrash Class Members

Construction	
SapBufferWithTrash	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
TrashType	Buffer type for the trash buffer resource only

SapBufferWithTrash Member Functions

The following are members of the SapBufferWithTrash Class.

SapBufferWithTrash.SapBufferWithTrash (constructor)

SapBufferWithTrash();

SapBufferWithTrash(
  int *count*
);

SapBufferWithTrash(
  int *count*,
  int *width*,
  int *heigh*,
  SapFormat *format*,
  SapBuffer.MemoryType *type*,
);

SapBufferWithTrash(
  int *count*,
  System.IntPtr[] *physicalAddress*

```
   int width,
   int height,
   SapFormat format
 );
SapBufferWithTrash (
   int count,
   System.IntPtr[] virtAddress
   int width,
   int height,
   SapFormat format,
   SapBuffer.MemoryType type
);
SapBufferWithTrash (
   int count,
   SapXferNode srcNode,
   SapBuffer.MemoryType,
);
SapBufferWithTrash (
   int count,
   string bufName
   int width,
   int height,
   SapFormat format,
   SapBuffer.MemoryType type
);
SapBufferWithTrash (
   int count,
   string bufName
   SapXferNode srcNode,
   SapBuffer.MemoryType type
);
SapBufferWithTrash (
   string bufName
   int startIndex
   int count,
   SapBuffer.MemoryType type
);
```

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 parameters (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, Multi-Board Sync Grab 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

SapBufferWithTrash.Create Method

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 Method.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab 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

SapBufferWithTrash.Destroy Method

bool **Destroy**();

Return Value

Returns **true** if successful, **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, Multi-Board Sync Grab 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

SapBufferWithTrash.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapBufferWithTrash .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapBufferWithTrash object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Bayer Demo, Multi-Board Sync Grab 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

SapBufferWithTrash.TrashType Property

SapBuffer.MemoryType **TrashType** (read/write)

Remarks

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.MemoryType.Dummy) to reduce bandwidth requirements associated with trash buffer transfers.

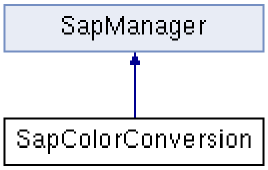
If you do not set a value for this property, then it is set to the same value as the other buffer resources when calling the Create method

You can only change the value of the TrashType property before calling the Create method. See the SapBuffer constructor for possible values for the buffer 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 input buffer and get a resulting RGB888 or RGB8888 output buffer (Methods 1-5)
- Apply color conversion on a raw Bi-Color88 input buffer and get a resulting RGB888 or RGB8888 output buffer (Method 7)
- Apply color conversion on a raw Bayer Mono16 input buffer and get a resulting RGB101010 or RGB16161616 output buffer (Methods 1-5)
- Apply color conversion on a raw Bi-Color1616 input buffer and get a resulting RGB101010 or RGB16161616 output buffer (Method 7)
- Apply white-balance gain on a RGB/Bayer/Bi-Color input buffer

Namespace: DALSA.SaperaLT.SapClassBasic

SapColorConversion Class Members

Construction	
SapColorConversion	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Dispose	Frees unmanaged memory resources
Properties	
AcqDevice	AcqDevic~e object for acquiring color images
Acquisition	Acquisition object for acquiring color images
Align	Color alignment mode
AvailAlign	Available alignment modes
AvailMethod	Available pixel value calculation methods
OutputBuffer	Buffer object used as the destination for software conversion
OutputBufferCount	Number of buffer resources used for software conversion
SoftwareEnabled	Checks if color conversion in software is enabled
SoftwareSupported	Checks if the input buffer supports color conversion
HardwareEnabled	Checks if color conversion in hardware is enabled
HardwareSupported	Checks if the input buffer supports color conversion
InputBuffer	Buffer object in which images are acquired or loaded
Enabled	Checks if color conversion is enabled

Gamma	Gamma correction factor for the color lookup table
IsAcqLut	Checks if the color lookup table corresponds to the acquisition LUT
Lut	Current color lookup table
LutEnabled	Color lookup table enable value
Method	Color pixel value calculation method
OutputFormat	Data output format of color conversion
SoftwareConversion	Checks if color conversion is performed in software or using the hardware
WBGain	Color white balance gain coefficients
WBOffset	Color white balance offset coefficients
Methods	
Convert	Converts a color-encoded image to an RGB image using software
Enable	Enables/disables color conversion
WhiteBalance	Calculates the white balance gain coefficients for color conversion

SapColorConversion Member Functions

The following are members of the SapColorConversion Class.

SapColorConversion.SapColorConversion (constructor)

SapColorConversion();
SapColorConversion(SapAcquisition *acquisition,* SapBuffer *buffer*);
SapColorConversion(SapAcqDevice *acqDevice*, SapBuffer *buffer*);
SapColorConversion(SapBuffer *buffer*);

Parameters

acquisition	SapAcquisition object to use for image acquisition and color conversion (if available in hardware)
acqDevice	SapAcqDevice object to use for image acquisition and color conversion (if available in hardware)
buffer	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 *buffer*. When using software conversion, the buffer object for the result of the conversion is automatically created using relevant attributes from *buffer*.

In both cases, the resulting SapBuffer object will be available through the OutputBuffer property.

Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

SapColorConversion.AcqDevice Property

SapAcqDevice **AcqDevice** (read/write)

Description

SapAcqDevice object to be used for image acquisition and for color conversion. You can only set the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapColorConversion.Acquisition Property

SapAcquisition **Acquisition** (read/write)

Description

SapAcquisition object to be used for image acquisition and for color conversion. You can only set the value of this property before calling the Create method.

Demo/Example Usage

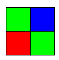
Not available

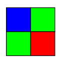
SapColorConversion.Align Property

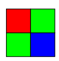
SapColorConversion.AlignMode **Align** (read/write)

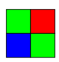
Description

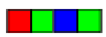
Color alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of the camera, or 1x4 line for a Bicolor camera. This mode may be one of the following values

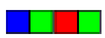
AlignMode.GBRG 

AlignMode.BGGR 

AlignMode.RGGB 

AlignMode.GRBG 

AlignMode.RGBG 

AlignMode.BGRG 

The initial value for this property is GRBG. It is then set to the acquisition color alignment value when calling the Create method (except when no acquisition is used).

Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

SapColorConversion.AvailAlign Property

SapColorConversion.AlignMode **AvailAlign** (read-only)

Description

Available color alignment modes, combined together using bitwise OR.

The initial value for this property includes all available modes. It is then set to the valid acquisition alignment modes when calling the Create method (except when no acquisition is used).

See the Align property for a list of possible modes.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.AvailMethod Property

SapColorConversion.CalculationMethod **AvailMethod** (read-only)

Description

Available color pixel value calculation methods, combined together using bitwise OR.

The initial value for this property includes all available methods. It is then set to the valid acquisition calculation methods when calling the Create method (except when no acquisition is used).

See the Method property for a list of possible methods.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.Convert Method

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

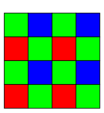
Converts a color-encoded image to an RGB image using software.

The source buffer for the conversion is the current buffer resource in the main buffer object, unless you specify a source index. The Buffer property 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 OutputBuffer property 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 Method

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 Acquisition property), 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 Buffer property).

Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

SapColorConversion.Destroy Method

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.Dispose Method

void **Dispose();**

Remarks

Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

SapColorConversion.Enable Method

bool **Enable**(bool *enable*, bool *useHardware*);

Parameters

enable	Set to **true** to enable color conversion, **false** to disable it
useHardware	Set to **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.ColorConversionAvailable property to find out if hardware correction is available in the acquisition device.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.Enabled Property

bool **Enabled** (read-only)

Description

Checks if color conversion is enabled. The initial value for this property depends on the acquisition device.

Use the Enable method if you need to enable or disable color conversion.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.Gamma Property

float **Gamma** (read/write)

Description

Color gamma correction factor. If color conversion is enabled, and the color lookup table is also enabled (using the LutEnableLut property), 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 Conversion Demo

SapColorConversion.HardwareEnabled Property

bool **HardwareEnabled**(read-only)

Description

Returns TRUE if hardware conversion is enabled.

Demo/Example Usage

Not available

SapColorConversion.HardwareSupported Property

bool **HardwareSupported**(read-only);

Description

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.InputBuffer Property

SapBuffer **Buffer** (read/write)

Description

SapBuffer object in which images are be acquired or loaded.

For software conversion, the buffer format must be either SapFormat.Mono8, SapFormat.Mono16, SapFormat.BICOLOR88, SapFormat.BICOLOR1212 or SapFormat.BICOLOR1616. The buffer object with the result of the conversion is then available by reading the value of the OutputBuffer property.

For hardware conversion, the buffer format may be SapFormat.RGB888, SapFormat.RGB8888, or SapFormat.RGB101010 (16-bit input image only). In this case, the buffer object returned by this property is the same as the one returned by reading the value of the OutputBuffer property.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.IsAcqLut Property

bool **IsAcqLut** (read-only)

Description

Checks if the color lookup table corresponds to the acquisition LUT. If the value of this property is **false**, then a software lookup table is used instead.

The initial value for this property is **false**. It is then set according to the current acquisition lookup table availability when calling the Create method.

Demo/Example Usage

Not available

SapColorConversion.Lut Property

SapLut **Lut** (read-only)

Description

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 LutEnable property.

For hardware conversion, this is actually the acquisition lookup table, which you may also obtain through the SapAcquisition.Luts property. If the acquisition hardware has no lookup table, then the value of this property 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.LutEnabled Property

bool **LutEnable** (read/write)

Description

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

Color Conversion Demo

SapColorConversion.Method Property

SapColorConversion.CalculationMethod **Method** (read/write)

Description

Color pixel value calculation methodwhich may be one of the following values:

CalculationMethod.Method1	Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image. Based on a 3x3 neighborhood operation.
CalculationMethod.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.
CalculationMethod.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.
CalculationMethod.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.
CalculationMethod.Method5	Technique based on a set of linear filters. It assumes that edges have a much stronger luminance than chrominance component.
CalculationMethod.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.

The initial value for this property is Method1. It is then set to the acquisition color method when calling the Create method (except when no acquisition is used).

For CalculationMethod.Method1, four cases are possible according to window position:

G	R	G
B	G	B
G	R	G

R = (R[up] + R[down]) / 2;

G = G

B = (B[left] + B[right]) / 2

R	G	R
G	B	G
R	G	R

R = (R[left,up] + R[right,up] + R[left,down] + R[right,down]) / 4

G = (G[left] + G[right] + G[up] + G[down]) / 4

B = B

B	G	B
G	R	G
B	G	B

R = R

G = (G[left] + G[right] + G[up] + G[down]) / 4

B = (B[left,up] + B[right,up] + B[left,down] + B[right,down]) / 4

G	B	G
R	G	R
G	B	G

R = (R[left] + R[right]) / 2;

G = G

B = (B[up] + B[down]) / 2

Demo/Example Usage

Color Conversion Demo

SapColorConversion.OutputBuffer Property

SapBuffer **OutputBuffer** (read-only)

Description

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 Buffer property.

You cannot read this property before calling the Create method.

Demo/Example Usage

Color Conversion Demo

SapColorConversion.OutputBufferCount Property

int **OutputBufferCount** (read/write)

Description

Number of buffer resources used for software conversion. The initial value for this property is 2.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapColorConversion.OutputFormat Property

SapFormat **OutputFormat** (read/write)

Description

Data output format of color conversion. The possible values for this attribute are SapFormat.RGB888, SapFormat.RGB8888, SapFormat.RGB101010, and SapFormat.RGB16161616.

The initial value for this property is SapFormat.Unknown. It is then set to the appropriate value when calling the Create method, or through this property.

You can only change the value of this property after calling the Create method

Demo/Example Usage

Color Conversion Demo

SapColorConversion.SoftwareConversion Property

bool **SoftwareConversion** (read-only)

Description

Checks if color conversion is performed in software or using the hardware

The value of this property is **true** if color conversion is not available in the acquisition, or if software conversion has been explicitly chosen by calling the Enable method.

The value of this property is **false** if color conversion is available in the acquisition, and software conversion has not been explicitly chosen by calling the Enable method.

Demo/Example Usage

Not available

SapColorConversion.SoftwareEnabled Property

bool **SoftwareEnabled**(read-only)

Description

Returns TRUE if software conversion is enabled.

Demo/Example Usage

Not available

SapColorConversion.SoftwareSupported Property

bool **SoftwareSupported**(read-only);

Description

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.WBGain Property

SapDataFRGB **GetWBGain** (read/write)

Description

Color 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.WBOffset Property

SapDataFRGB **WBOffset** (read/write)

Description

Color white balance offset coefficients. These apply only for hardware conversion, that is, when the value of the SoftwareConversion property is **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

Not available

SapColorConversion.WhiteBalance Method

bool **WhiteBalance**(int *x*, int *y*, int *width*, int *height*);
bool **WhiteBalance**(SapBuffer *buffer*, 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
buffer	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:

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

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

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

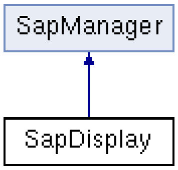
where $\overline{R}$, $\overline{G}$ and $\overline{B}$ are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormat.Mono8 or SapFormat.Mono16. 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 *buffer* 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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapDisplay Class Members

Construction	
SapDisplay	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
Height	Height (in lines) for the current display mode
Interlaced	Checks if the current display mode is interlaced or progressive
Location	Location where the display resource is located
PixelDepth	Number of significant bits per pixel for the current display mode
PrimaryVGABoard	Checks if the current display belongs to the primary VGA board in the system
RefreshRate	Refresh rate for the current display mode
Type	Type of the display (primary or secondary)
Width	Width (in pixels) for the current display mode
Methods	
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
GetGraphics	Gets the .NETgraphics object corresponding to the entire screen
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
ReleaseGraphics	Releases the .NETgraphics object corresponding to the entire screen

SapDisplay Member Functions

The following are members of the SapDisplay Class.

SapDisplay.SapDisplay (constructor)

SapDisplay();

Remarks

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

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 Method

bool **Create**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Creates all the low-level Sapera 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 Method

bool **Destroy**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Destroys all the low-level Sapera 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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapDisplay .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapDisplay object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Not available

SapDisplay.GetCapability Method

bool **GetCapability**(SapDisplay.Cap *capId*, out int *capValue*);
bool **GetCapability**(SapDisplay.Cap *capId*, SapDisplay.Val *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the display module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapDisplay class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORDISPLAY_CAP_WIDTH_MIN becomes SapDisplay.Cap.WIDTH_MIN

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

Demo/Example Usage

Not available

SapDisplay.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapDisplay.Cap *capId*);

Parameters

capId	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapDisplay.GetParameter, SetParameter Methods

bool **GetParameter**(SapDisplay.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapDisplay.Prm *paramId*, out SapDisplay.Val *paramValue*);
bool **GetParameter**(SapDisplay.Prm *paramId*, out System.IntPtr *paramValue*);
bool **GetParameter**(SapDisplay.Prm *paramId*, out string *paramValue*);
bool **GetParameter**(SapDisplay.Prm *paramId*, out int[] *paramValue*);

bool **SetParameter**(SapDisplay.Prm *paramId*, int *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read or write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the display module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.Int32Array, then *paramValue* is an integer array (uninitialized for GetParameter) with an unknown number of elements. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter). If this value is SapCapPrmType.IntPtr, then *paramValue* is an address (uninitialized for GetParameter).

The following examples show how to declare an uninitialized array and call GetParameter:

```
int[] paramValue;
result = disp.GetParameter(paramId, out paramValue);
```

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORDISPLAY_PRM_TYPE becomes SapDisplay.Prm.TYPE

You can also use the versions of GetParameter/SetParameter which take a SapDisplay.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORDISPLAY_VAL_TYPE_SYSTEM becomes SapDisplay.Val.TYPE_SYSTEM

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 SapDisplay class. Also, directly setting parameter values may interfere with the correct operation of the class.

Demo/Example Usage

Not available

SapDisplay.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapDisplay.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.Int32Array	Array of 32-bit integers
SapCapPrmType.String	Text string
SapCapPrmType.IntPtr	Address value

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapDisplay.GetGraphics Method

System.Drawing.Graphics **GetGraphics**();

Return Value

.NET graphics object

Remarks

Gets the .NETgraphics object corresponding to the entire screen for the current SapDisplay object. Use the ReleaseGraphics method you are finished using this object.

Demo/Example Usage

Not available

SapDisplay.Height Property

int **Height** (read-only)

Description

Height (in lines) for the current display mode.

The initial value for this property is 0. It is then set according to the current display value when calling the Create method.

Demo/Example Usage

Not available

SapDisplay.Interlaced Property

bool **Interlaced** (read-only)

Description

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.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapDisplay.Cap *capId*);

Parameters

capId	Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.

SapDisplay.IsParameterAvailable Method

bool **IsParameterAvailable**(SapDisplay.Prm *prmId*);

Parameters

prmId	Low-level Sapera parameter to check

Return Value

Returns **true** if the parameter is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.

SapDisplay.Location Property

SapLocation **Location (**read/write)

Description

Location where the display resource is located. This usually corresponds to the system server.

You can only change the value of this property before calling the Create method.

SapDisplay.PixelDepth Property

int **PixelDepth** (read-only)

Description

Number of significant bits per pixel for the current display mode.

The initial value for this property is 0. It is then set according to the current display value when calling the Create method.

SapDisplay.PrimaryVGABoard Property

bool **PrimaryVGABoard** (read-only)

Description

Checks if the current display belongs to the primary VGA board in the system. You can only read the value of this property after calling the Create method.

SapDisplay.RefreshRate Property

int **RefreshRate** (read-only)

Description

Refresh rate (in Hz) for the current display mode

The initial value for this property is 0. It is then set according to the current display value when calling the Create method.

SapDisplay.ReleaseGraphics Method

bool **ReleaseGraphics**(System.Drawing.Graphics *graphic*);

Parameters

graphic	.NET graphics object returned by the GetGraphics method

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Releases the .NETgraphics object corresponding to the entire screen for the current display object.

SapDisplay.Type Property

SapDisplay.ConfigType **Type** (read-only)

Description

Type of the display, which can be one of the following values:

ConfigType.Unknown	Undetermined display type
ConfigType.System	A display under the control of the primary Windows display driver. It normally displays the Windows Desktop.
ConfigType.Duplicate	A secondary display that shows the same contents as the primary Windows VGA display
ConfigType.Extended	A secondary display that extends the desktop from the primary Windows VGA display
ConfigType.Independent	A secondary display that is completely independent from the primary Windows VGA display

The initial value for this property is ConfigType.Unknown. It is then set according to the current display value when calling the Create method.

Demo/Example Usage

Not available

SapDisplay.Width Property

int **Width** (read-only)

Description

Width (in pixels) for the current display mode.

The initial value for this property is 0. It is then set according to the current display value when calling the Create method.

Demo/Example Usage

Not available

SapDisplayDoneEventArgs

The SapDisplayDoneEventArgs class contains the arguments to the application handler method for the SapView.DisplayDone event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapDisplayDoneEventArgs Class Members

Properties	
Context	Application context associated with the display event

SapDisplayDoneEventArgs Member Properties

The following are members of the SapDisplayDoneEventArgs Class.

SapDisplayDoneEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with display events. See the DisplayDoneContext property of the SapView class for more details.

Demo/Example Usage

Color Conversion Demo

SapErrorEventArgs

The SapErrorEventArgs class contains the arguments to the application handler method for the SapManager.Error event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapErrorEventArgs Class Members

Properties

Context	Application context associated with the error event
Message	Error message associated with the error event

SapErrorEventArgs Member Properties

The following are members of the SapErrorEventArgs Class.

SapErrorEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with error events. See the ErrorContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapErrorEventArgs.Message Property

string **Message** (read-only)

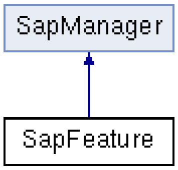
Description

Error message associated with the call to the application event handler method.

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapFeature Class Members

Construction	
SapFeature	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
ArrayLength	Number of bytes required for an array type feature
Category	Category to which the current feature belongs
DataAccessMode	Current data access mode for a feature
DataRepresentation	Mathematical representation of a integer or float feature
DataSign	Checks if an integer/float feature is signed or not
DataType	Data type of the feature
DataWriteMode	Checks if a feature can be modified when the transfer object is connected and/or acquiring
Description	Text which represents the full description of the feature
DisplayName	Descriptive name of the feature
EnumCount	Number of possible values for a feature which belongs to an enumerated type
EnumEnabled	Checks if the enumeration value corresponding to a specified index is enabled
EnumText	String values for the enumerated type corresponding to the current feature
EnumValues	Integer values for the enumerated type corresponding to the current feature
FloatDisplayNotation	Gets the notation type to use to display a float type feature
FloatDisplayPrecision	Gets the number of decimal places to display for a float type feature
IsSelector	Determines if the value of a feature directly affects other features
Location	Location where the feature resource is located
Name	Short name of the feature
PollingTime	Interval of time between two consecutive feature updates
SavedToConfigFile	Checks if a feature is saved to a CCF configuration file

SelectedFeatureCount	Number of features associated with a selector
SelectedFeatureIndexes	Indexes of all features associated with a selector
SelectedFeatureNames	Names of all features associated with a selector
SelectingFeatureCount	Number of selectors associated with a feature
SelectingFeatureIndexes	Indexes of all selectors associated with a feature
SelectingFeatureNames	Names of all selectors associated with a feature
SiToNativeExp10	Feature conversion factor from international system (SI) units to native units
SiUnit	Physical units representing the feature in the international system (SI)
Standard	Checks if a feature is standard or custom
ToolTip	Text which represents the explanation of the feature
UserVisibility	Level of visibility assigned to a feature
ValueIncrementType	Type of increment for an integer or floating-point feature
ValidValueCount	Number of valid values for an integer or floating-point feature which defines them as a list
Methods	
GetEnumTextFromValue	Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature
GetEnumValueFromText	Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature
GetValidValue	Gets one of a predefined set of valid values for a feature
GetValueIncrement	Gets the minimum acceptable increment for an integer or a float feature
GetValueMin	Gets the minimum acceptable value for a feature
GetValueMax	Gets the maximum acceptable value for a feature

SapFeature Member Functions

The following are members of the SapFeature Class.

SapFeature.SapFeature (constructor)

SapFeature();
SapFeature(SapLocation *location*);

Parameters

location	SapLocation object specifying where the feature is located. The 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 function.

Demo/Example Usage

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

SapFeature.ArrayLength Property

int **ArrayLength (**read-only)

Description

Number of bytes required to store the value of an array type feature, that is, when the value returned by the DataType property is SapFeature.Type.Array. 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.Category Property

string **Category** (read-only)

Description

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.Create Method

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.DataAccessMode Property

SapFeature.AccessMode **DataAccessMode** (read-only)

Description

Current data access mode for a feature, which can be one of the following values:

AccessMode.Undefined	Undefined access mode
AccessMode.RW	The feature may be read and written. Most features are of this type.
AccessMode.RO	The feature can only be read.
AccessMode.WO	The feature can only be written. This is the case for some features which represent commands (or actions) such as 'TimestampReset'.
AccessMode.NP	The feature is not present. The feature is visible in the interface but is not implemented for this device.
AccessMode.NE	The feature is present but currently not enabled. Often used when a feature depends on another feature's value.

Demo/Example Usage

Camera Features Example, Camera Files Example

SapFeature.DataRepresentation Property

SapFeature.Representation **DataRepresentation** (read-only)

Description

Mathematical representation of a integer or float feature, which can be one of the following values:

Representation.Undefined	Undefined representation
Representation.Linear	The feature follows a linear scale
Representation.Logarithmic	The feature follows a logarithmic scale
Representation.Boolean	The feature can have two values: zero or non-zero

Demo/Example Usage

Not available

SapFeature.DataSign Property

SapFeature.Sign **DataSign** (read-only)

Description

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. It can be one of the following values:

Sign.Undefined	Sign is undefined
Sign.Signed	The feature is a signed integer of float
Sign.Unsigned	The feature is an unsigned integer of float

Demo/Example Usage

Not available

SapFeature.DataType Property

SapFeature.Type **DataType** (read-only)

Description

Data type of a feature, which can be one of the following values:

Sapera Type	**Description**
Type.Undefined	Undefined type
Type.Int32	32-bit integer
Type.Int64	64-bit integer
Type.Float	32-bit floating-point
Type.Double	64-bit floating-point
Type.Bool	Boolean
Type.Enum	Enumeration
Type.String	ASCII character string
Type.Buffer	Sapera LT buffer object (SapBuffer)
Type.Lut	Sapera LT look-up table object (SapLut)
Type.Array	Sapera LT buffer object (SapBuffer)

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 ArrayLength property.

Demo/Example Usage

Camera Features Example

SapFeature.DataWriteMode Property

SapFeature.WriteMode **DataWriteMode (**read-only)

Description

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 transfered to the buffer. Use this information to prevent an application from changing certain features when the transfer object is connected and/or acquiring.

Note that this property is only relevant for features which are writable, that is, when the DataAccessMode property identifies the feature as read/write or read-only.

The data write mode can be one of the following values:

WriteMode.Undefined	Undefined write mode
WriteMode.Always	The feature can always be written
WriteMode.NotAcquiring	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.
WriteMode.NotConnected	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.

Demo/Example Usage

Not available

SapFeature.Description Property

string **Description** (read-only)

Description

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.Destroy Method

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.DisplayName Property

string **DisplayName** (read-only)

Description

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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapFeature .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapFeature object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

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

SapFeature.EnumCount Property

int **EnumCount** (read-only)

Description

Number of possible values for a feature which belongs to an enumerated type. Use this property along with the EnumText and EnumValues properties to enumerate all the items contained within an enumeration feature.

Demo/Example Usage

Camera Features Example

SapFeature.EnumEnabled Property

bool **get_EnumEnabled**(int *index*) (read-only)

Parameters

enumIndex	Index of the enumeration item (from 0 to the value returned by the EnumCount property, minus 1)

Description

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 property to find out the enable state of an item at any given time.

There is only an indexed version of this property. Here are examples of how to use it for .NET languages:

```
bool enabled = feature.get_EnumEnabled(index);
```

Demo/Example Usage

Not available

SapFeature.EnumText Property

string[] **EnumText** (read-only)

Description

String values for the enumerated type corresponding to the current feature. Use this property with the EnumCount and EnumValues properties to enumerate all the items contained within an enumeration feature.

Here are examples of how to retrieve this property:

```
string[] enumText = feature.EnumText;
```

Demo/Example Usage

Camera Features Example

SapFeature.EnumValues Property

int[] **EnumValues** (read-only)

Description

Integer values for the enumerated type corresponding to the current feature. Use this property along with EnumCount and EnumText properties to enumerate all the items contained within an enumeration feature.

Here are examples of how to retrieve this property:

```
int[] enumValues = feature.EnumValues;
```

Demo/Example Usage

Not available

SapFeature.FloatDisplayNotation Property

SapFeature.FloatNotation **FloatDisplayNotation** (read-only)

Description

Gets the notation type to use to display a float type feature. Possible values are:

FloatNotation.Fixed	Display variable using fixed notation. For exampe, 123.4
FloatNotation.Scientific	Display variable using scientific notation. For example, 1.234e-2.
FloatNotation.Undefined	Undefined.

Demo/Example Usage

Not available

SapFeature.FloatDisplayPrecision Property

long **FloatDisplayPrecision** (read-only)

Description

Gets the number of decimal places to display for a float type feature.

Demo/Example Usage

Not available

SapFeature.GetEnumTextFromValue Method

bool **GetEnumTextFromValue**(int *enumValue*, out string *enumText*);

Parameters

enumValue	Value to look for in the enumeration items
enumText	Returned text string

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.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *enumText* argument.

Demo/Example Usage

Camera Features Example

SapFeature.GetEnumValueFromText Method

bool **GetEnumValueFromText**(string *enumText*, out int *enumValue*);

Parameters

enumText	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.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *enumValue* argument.

Demo/Example Usage

Not available

SapFeature.GetValidValue Method

bool **GetValidValue**(int *validValueIndex*, out int *validValue*);
bool **GetValidValue**(int *validValueIndex*, out uint *validValue*);
bool **GetValidValue**(int *validValueIndex*, out long *validValue*);
bool **GetValidValue**(int *validValueIndex*, out ulong *validValue*);
bool **GetValidValue**(int *validValueIndex*, out float *validValue*);
bool **GetValidValue**(int *validValueIndex*, out double *validValue*);

Parameters

validValueIndex	Index of the valid value, can be any value from 0 to the value returned by the ValidValueCount property, 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 ValueIncrementType property returns IncrementType.List.

Demo/Example Usage

Camera Features Example

SapFeature.GetValueIncrement Method

bool **GetValueIncrement**(out int *valueIncrement*);
bool **GetValueIncrement**(out uint *valueIncrement*);
bool **GetValueIncrement**(out long *valueIncrement*);
bool **GetValueIncrement**(out ulong *valueIncrement*);
bool **GetValueIncrement**(out float *valueIncrement*);
bool **GetValueIncrement**(out double *valueIncrement*);

Parameters

valueIncrement	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.GetValueMax Method

bool **GetValueMax**(out int *valueMax*);
bool **GetValueMax**(out uint *valueMax*);
bool **GetValueMax**(out long *valueMax*);
bool **GetValueMax**(out ulong *valueMax*);
bool **GetValueMax**(out float *valueMax*);
bool **GetValueMax**(out double *valueMax*);

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 integer version to get the maximum length of the string (excluding the trailing null character).

Demo/Example Usage

GigE Camera Compression Demo, Camera Events Example, Camera Features Example, GigE Auto White Balance Example, Grab Camera Link Example

SapFeature.GetValueMin Method

bool **GetValueMin**(out int *valueMin*);
bool **GetValueMin**(out uint *valueMin*);
bool **GetValueMin**(out long *valueMin*);
bool **GetValueMin**(out ulong *valueMin*);
bool **GetValueMin**(out float *valueMin*);
bool **GetValueMin**(out double *valueMin*);

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 integer version to get the minimum length of the string (excluding the trailing null character).

Demo/Example Usage

GigE Camera Compression Demo, Camera Events Example, Camera Features Example, GigE Auto White Balance Example, Grab Camera Link Example

SapFeature.IsSelector Property

bool **IsSelector** (read-only)

Description

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 properties to find out which features are associated: SelectedFeatureCount, SelectedFeatureIndexes, and SelectedFeatureNames.

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

Demo/Example Usage

Not available

SapFeature.Location Property

SapLocation **Location (**read/write)

Description

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 change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapFeature.Name Property

string **Name** (read-only)

Description

Short name of the feature. This name can be used with SapAcqDevice methods which expect a feature name. This string should not be used for display in a graphical user interface. Use the DisplayName property instead to provide a more descriptive name.

Demo/Example Usage

Not available

SapFeature.PollingTime Property

int **PollingTime (**read-only)

Description

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 property is only relevant for acquisition devices which are supported through the Network Imaging Package (GigE Vision 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.SavedToConfigFile Property

bool **SavedToConfigFile** (read/write)

Description

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 this property 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 Network Imaging Package (GigE Vision Framework), the features saved to the configuration file are hardcoded and cannot be changed. Therefore writing this property has no effect.

Demo/Example Usage

Not available

SapFeature.SelectedFeatureCount Property

int **SelectedFeatureCount (**read-only)

Description

Number of features associated with a selector (value of IsSelector property is **true**), or 0 if the current feature is not a selector. These selected features can be considered as children of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature.SelectedFeatureIndexes Property

int[] **SelectedFeatureIndexes** (read-only)

Description

Indexes of all features associated with a selector (value of IsSelector property is **true**). The indexes are relative to the acquisition device, with possible values from 0 to the value returned by the SapAcqDevice.FeatureCount property, minus 1. These features can be considered as the children of the current SapFeature object.

The SelectedFeatureCount property returns the number of features associated with the selector.

The indexes returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
int[] selectedFeatureIndexes = feature.SelectedFeatureIndexes;
```

Demo/Example Usage

Not available

SapFeature.SelectedFeatureNames Property

string[] **SelectedFeatureNames** (read-only)

Description

Names of all features associated with a selector (value of IsSelector property is **true**). These features can be considered as the children of the current SapFeature object.

The SelectedFeatureCount property returns the number of features associated with the selector.

The names returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
string[] selectedFeatureNames = feature.SelectedFeatureNames;
```

Demo/Example Usage

Not available

SapFeature.SelectingFeatureCount Property

int **SelectingFeatureCount (**read-only)

Description

Number of selectors (value of IsSelector property is **true**) associated with a feature, or 0 if there are no associated selectors. These selectors can be considered as parents of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature.SelectingFeatureIndexes Property

int[] **SelectingFeatureIndexes** (read-only)

Description

Indexes of all selectors (value of IsSelector property is **true**) associated with a feature. The indexes are relative to the acquisition device, with possible values from 0 to the value returned by the SapAcqDevice.FeatureCount property, minus 1. These selectors can be considered as the parents of the current SapFeature object.

The SelectingFeatureCount property returns the number of selectors associated with the feature.

The indexes returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
int[] selectingFeatureIndexes = feature.SelectingFeatureIndexes;
```

Demo/Example Usage

Not available

SapFeature.SelectingFeatureNames Property

string[] **SelectingFeatureNames** (read-only)

Description

Names of all selectors (value of IsSelector property is **true**) associated with a feature. These selectors can be considered as the parents of the current SapFeature object.

The SelectingFeatureCount property returns the number of selectors associated with the feature.

The names returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
string[] selectingFeatureNames = feature.SelectingFeatureNames;
```

Demo/Example Usage

Not available

SapFeature.SiToNativeExp10 Property

int **SiToNativeExp10** (read-only)

Description

Gets the base 10 exponent (positive or negative) 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:

VNATIVE = VSI * 10E

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:
VNATIVE = 0.5*106 = 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 SiUnit property 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.SiUnit Property

string **SiUnit (**read-only)

Description

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 SiToNativeExp10 property.

Demo/Example Usage

Not available

SapFeature.Standard Property

bool **Standard (**read-only)

Description

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

SapFeature.ToolTip Property

string **ToolTip** (read-only)

Description

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.UserVisibility Property

SapFeature.Visibility **UserVisibility** (read-only)

Description

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. It can be one of the following values:

Visibility.Undefined	Undefined visibility level
Visibility.Beginner	The feature should be made visible to any user
Visibility.Expert	The feature should be made visible to users with a certain level of expertise
Visibility.Guru	Specifies that the feature should be made visible to users with a high level of expertise
Visibility.Invisible	The feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features

Demo/Example Usage

Not available

SapFeature.ValueIncrementType Property

SapFeature.IncrementType **ValueIncrementType** (read-only)

Description

Type of increment for an integer or floating-point feature. This is useful for finding out which values are valid for this feature. It can be one of the following values:

IncrementType.Undefined	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.
IncrementType.None	The feature has no increment. Use the GetValueMin and GetValueMax functions to find out the feature value limits.
IncrementType.Linear	The feature has a fixed increment. Use the GetValueMin and GetValueMax functions to find the feature value limits, and GetValueIncrement to find the increment.
IncrementType.List	The feature has a fixed set of valid values. Use the ValidValueCount property to find the number of values, and the GetValidValue function to enumerate them.

Demo/Example Usage

Camera Features Example

SapFeature.ValidValueCount Property

int **ValidValueCount** (read-only)

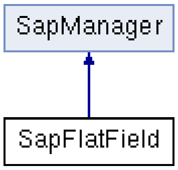
Description

Number of valid values for an integer or floating-point feature which defines them as a list, that is, the ValueIncrementType property returns IncrementType.List. In this case, use the GetValidValue function to enumerate these values.

Demo/Example Usage

Camera Features Example

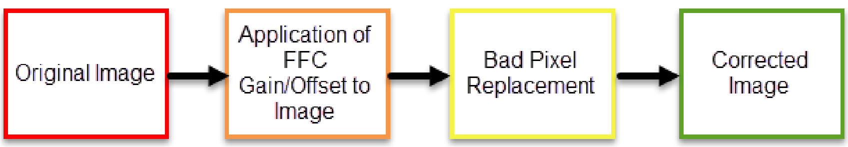
SapFlatField



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.

The flat field correction processing consists of 2 stages. The 1st stage applies the FFC Gain/Offset to all of the good pixels. The 2nd stage replaces the bad pixels.



The gain and offset is applied to each good pixel. The maximum gain and offset values are dependent on the pixel depth of the acquired image.

- A Gain of 0 identifies a bad pixel. In this case, no gain or offset will be applied to this pixel.
- Offsets range from 0 to Max Pixel Value.
- Gains range from 1 .. 2.
- Corrected Value = (Original Value – Offset) * ((Gain) /Gain Divisor)
- If the Corrected Value < 0 then Corrected Value = 0;
- If the Corrected Value > Max Pixel Value then Corrected Value = Max Pixel Value

Pixel Depth	Offset Range	Gain Range	Gain Divisor
8	0 .. 255	1 .. 255	128
10	0 .. 1023	1 .. 1023	512
12	0 .. 4095	1 .. 4095	2048
14	0 .. 16383	1 .. 16383	16384

Namespace: DALSA.SaperaLT.SapClassBasic

SapFlatField Class Members

Construction	
SapFlatField	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Dispose	Frees unmanaged memory resources
Properties	
AcqDevice	Acquisition device object for acquiring images and flat-field correction
Acquisition	Acquisition object for acquiring images and flat-field correction
AcqVideoType	Gets the acquisition video type (monochrome or color)
BlackPixelPercentage	Allowed percentage of black pixels (value 0) in an image when flat field calibration is done
Buffer	Buffer object for operating the flat-field correction without an acquisition object
BufferOffset	Buffer objects for the flat-field correction offset and gain coefficients
BufferGain	
ClippedGainOffsetDefects	Checks whether to consider pixels as defective when calculated gain or offset coefficients reach the hardware limitations
ClusterMapPixelCount	Gets the number of defective pixels in the pixel replacement cluster map
CorrectionType	Line scan vs area scan correction type
DeviationMaxBlack	Maximum deviation of the calculated coefficients towards black
DeviationMaxWhite	Maximum deviation of the calculated coefficients towards white
Enable	Checks if flat-field correction is enabled
GainBase	Gain base used when calculating the gain coefficients
GainDivisor	Factor by which a gain coefficient has to be divided for getting a unitary scale factor.
GainMin,	Minimum and maximum values for computed gain values
GainMax	
NumLinesAverage	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.
NumFramesAverage	Number of frames to average for the calibration before computing the gain and offset coefficients for areascan video source.
OffsetFactor	Multiplication factor applied to the offset coefficients
OffsetMin	Minimum and maximum values for computed offset values
OffsetMax	
PixelReplacement	Checks if replacement of defective pixels is enabled
SetRegionOfInterest	Specifies the ROI of coefficients to use for software flat-field correction.
SoftwareCorrection	Checks if flat-field correction is performed in software or using the hardware
VerticalOffset	Vertical line scan averaging offset in a full frame
Methods	
Clear	Clears the gain and offset buffers

ComputeGain	Calculates the flat-field correction gain coefficients
ComputeOffset	Calculates the flat-field correction offset coefficients
Enable	Enables/disables flat-field correction
Execute	Performs the software implementation of flat-field correction
GetAverage	Gets average pixel value and standard deviation for a buffer
GetBufferGain GetBufferGain	Copy the buffer objects for the flat-field correction gain and offset coefficients to application supplied buffer objects
GetStats	Gets statistics for a buffer subtracted from the offset buffer
Load	Loads gain and offset buffer data from disk files or from existing buffer objects
ReadGainOffsetFromDevice	Gets the current flat-field correction coefficients from the acquisition hardware
Save	Saves gain and offset buffer data to disk files
SetClusterMap	Sets the cluster map file to use.
SetVideoType	Sets the acquisition video type (monochrome or color)

SapFlatField Member Functions

The following are members of the SapFlatField Class.

SapFlatField.SapFlatField (constructor)

SapFlatField();
SapFlatField(SapAcquisition *acquisition*);
SapFlatField(SapAcqDevice *acqDevice*,);
SapFlatField(SapBuffer *buffer*);

Parameters

acquisition	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.
acqDevice	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.
buffer	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.

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.AcqDevice Property

SapAcqDevice **AcqDevice** (read/write)

Description

Acquisition device 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 change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapFlatField.Acquisition Property

SapAcquisition **Acquisition** (read/write)

Description

Acquisition object to be used for image acquisition and for flat-field correction. This object typically corresponds to a frame grabber.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapFlatField.AcqVideoType Property

SapFlatField.VideoType **AcqVideoType** (read-only)

Parameters

videoType	New acquisition video type (SapAcquisition.VideoType.Mono or SapAcquisition.VideoType.Color)

Remarks

Acquisition video type. The initial value for this property 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 the SetVideoType method (since this property is read-only) before the Create method.

Demo/Example Usage

Not available

SapFlatField.BlackPixelPercentage Property

float **BlackPixelPercentage** (read/write)

Description

Allowed percentage of black pixels (with value 0) in an image when flat field calibration is done. You must set the value of this property 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 property is 2.0.

Demo/Example Usage

Not available

SapFlatField.Buffer Property

SapBuffer **Buffer** (read/write)

Description

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 change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapFlatField.BufferOffset, SapFlatField.BufferGain Properties

SapBuffer **BufferOffset** (read-only)
SapBuffer **BufferGain** (read-only)

Description

Buffer objects for the flat-field correction offset and gain coefficients.

Demo/Example Usage

Not available

SapFlatField.Clear Method

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.ClippedGainOffsetDefects Property

bool **ClippedGainOffsetDefects** (read/write)

Description

Controls 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 property 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.ClusterMapPixelCount Property

int **ClusterMapPixelCount** (read-only)

Description

Gets the number of defective pixels in the defective pixel cluster map currently loaded in the device.

Demo/Example Usage

Not available

SapFlatField.ComputeGain Method

bool **ComputeGain**(SapBuffer *buffer*, SapFlatFieldDefects *defects*);
bool **ComputeGain**(SapBuffer *buffer*, SapFlatFieldDefects *defects*, SapData *target*);
bool **ComputeGain**(SapBuffer *buffer*, SapFlatFieldDefects *defects*, bool *useImageMaxValue*);

Parameters

Buffer	Buffer object containing a calibration image
Defects	SapFlatFieldDefects object
useImageMaxValue	Indicates how to calculate the range of the calibrated output images
Target	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 *useImageMaxValue* 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 SaBuffer.PixelDepth property).

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 *defects* contains statistics about the defects found in the gain image. It has the following properties:

int NumDefects	Number of defective pixels
int NumClusters	Number of defective pixels that are adjacent
float DefectRatio	Ratio between defective pixels and good pixels in percent

Note

The version of this method with a *numImages* argument is now obsolete, it has been replaced by the NumFramesAverage property. However, for backwards compatibility, if this obsolete method is called with a *numImages* argument set to any other value than 0, it will override the value set by the property.

Demo/Example Usage

Not available

SapFlatField.ComputeOffset Method

bool **ComputeOffset**(SapBuffer *buffer*);

Parameters

buffer	Buffer object containing a calibration image

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 version of this method with a *numImages* argument is now obsolete, it has been replaced by the NumFramesAverage property. However, for backwards compatibility, if this obsolete method is called with a *numImages* argument set to any other value than 0, it will override the value set by the property.

Demo/Example Usage

Not available

SapFlatField.CorrectionType Property

SapFlatField.ScanCorrectionType **CorrectionType** (read/write)

Description

Flat-field correction type, which can be one of the following values

ScanCorrectionType.Field	Correction is performed on full frames
ScanCorrectionType.Line	Correction is performed on individual lines
ScanCorrectionType.Invalid	Invalid correction type

The initial value for this property is Invalid. It is then set according to the acquisition scan type when calling the Create method. This means that changing the value of this property is only relevant when no acquisition 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.Create Method

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 Method

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.DeviationMaxBlack Property

int **DeviationMaxBlack** (read/write)

Description

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 property 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.DeviationMaxWhite Property

int **DeviationMaxWhite** (read/write)

Description

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 property 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.Dispose Method

void **Dispose();**

Remarks

Frees unmanaged memory used internally by a SapFlatField .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapFlatField object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

FlatField Demo, GigE FlatField Demo

SapFlatField.Enable Method

bool **Enable**(bool *enable*, bool *useHardware*);

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.Enabled Property

int **Enabled** (read-only)

Description

Checks if flat-field correction is enabled. The initial value for this property 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.Execute Method

bool **Execute**(SapBuffer *buffer*);
bool **Execute**(SapBuffer *buffer*, int *bufIndex*);

Parameters

buffer 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:

correctedValue = (originalValue – offset) * (gain / gainDivisor)

For 8-bit gain coefficients, the gain division is typically equal to 128, so that a gain value between 0 and 255 becomes a value between 0 and 2. Use the GainDivisor property to change its value.

Demo/Example Usage

FlatField Demo, GigE FlatField Demo

SapFlatField.GainBase Property

int **GainBase** (read/write);

Remarks

Gain base used when calculating the gain coefficients.

When using a Teledyne DALSA acquisition device which support hardware-based gain base (e.g., Genie TS), then the initial value for this property is only meaningful after calling the Create method, since it is retrieved from the acquisition hardware itself. In this case, application code should not change this property at all.

For all other acquisition devices, and also for software based flat-field correction, the initial value for this property is 0, and application code can modify it if required.

Demo/Example Usage

Not available

SapFlatField.GainDivisor Property

int **GainDivisor** (read/write)

Description

Factor by which the gain coefficients have to be divided for getting a unitary scale factor.

The initial value for this property is 128. It is then set to the acquisition gain divisor value when calling the Create method.

This property can only be used when operating without hardware support.

Demo/Example Usage

FlatField Demo, GigE FlatField Demo

SapFlatField.GainMin, SapFlatField.GainMax Properties

int **GainMin** (read/write)
int **GainMax** (read/write)

Description

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 properties are 0 and 255.

Demo/Example Usage

Not available

SapFlatField.GetAverage Method

bool **GetAverage**(SapBuffer *buffer*, SapFlatFieldStats *stats*);

Parameters

Buffer	Buffer object from which to compute the average
Stats	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.GetBufferOffset, SapFlatField.GetBufferGain Methods

bool **GetBufferOffset**(SapBuffer *buffer*);
bool **GetBufferOffset**(SapBuffer *buffer*, int *bufIndex*, int *offsetIndex*);
bool **GetBufferGain**(SapBuffer *buffer*);
bool **GetBufferGain**(SapBuffer *buffer*, int *bufIndex*, int *gainIndex*);

Parameters

buffer	Buffer object from which to compute the average
stats	SapFlatFieldStats object for returned statistics

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Copy the buffer objects for the flat-field correction gain and offset coefficients to application supplied buffers 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.GetStats Method

bool **GetStats**(SapBuffer *buffer*, SapFlatFieldStats *stats*);

Parameters

Buffer	Buffer object from which to compute the average
Stats	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 *stats* contains statistics about the image. It has the following read-only properties:

int NumComponents	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 SapColorConversion.Align property).
int Average int get_Average(int *index*)	Buffer average
int StdDeviation int get_StdDeviation(int *index*)	Buffer standard deviation
int PeakPosition int get_PeakPosition(int *index*)	Peak value position in the histogram used to calculate the gain coefficients
int Low int get_Low(int *index*)	Lower bound of the histogram. Pixels below the lower bound will be assigned a gain of 2.
int High int get_High(int *index*)	Higher bound of the histogram. Pixels above the higher bound will be assigned a gain of 1.
int NumPixels int get_NumPixels(int *index*)	Number of pixels in the histogram between the lower and the higher bounds
float PixelRatio float get_PixelRatio(int *index*)	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

There are two versions of each property (except NumComponents): one without an index, and one with an index. The former automatically uses an index of 0, corresponding to the first component The latter allow the component index to be specified.

Here are examples of how to use the indexed property for .NET languages:

```
int low = stats.get_Low(index);
```

Note that only the NumComponents, Average, and StdDeviation properties are relevant when the SapFlatFieldStats object is used in a call to the Average property.

Demo/Example Usage

Not available

SapFlatField.Load Method

bool **Load**(string *fileName*);
bool **Load**(SapBuffer *bufferGain*, SapBuffer *bufferOffset*);

Parameters

fileName	Name of the image file with the gain and offset parameters
bufferGain	Buffer object containing the gain values
bufferOffset	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.NumFramesAverage Property

int **NumFramesAverage** (read/write)

Description

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 property is 10. This property must be set before calling SapFlatField.ComputeOffset Method and SapFlatField.ComputeGain Method.

Demo/Example Usage

Not available

SapFlatField.NumLinesAverage Property

int **NumLinesAverage** (read/write)

Description

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 property is 128. This property must be set before calling SapFlatField.ComputeOffset Method and SapFlatField.ComputeGain Methods.

Demo/Example Usage

Not available

SapFlatField.OffsetFactor Property

double **OffsetFactor** (read/write)

Description

Multiplication factor used when calculating flat field offset coefficients.

When using a Teledyne DALSA acquisition device which support a hardware-based offset factor (for example, Genie TS), then the initial value for this property is only meaningful after calling the Create method, since it is retrieved from the acquisition hardware itself. In this case, application code should not change this property at all.

For all other acquisition devices, and also for software based flat-field correction, the initial value for this property is 1, and application code can modify it if required.

Demo/Example Usage

Not available

SapFlatField.OffsetMin, SapFlatField.OffsetMax Properties

int **OffsetMin** (read/write)
int **OffsetMax** (read/write)

Description

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.PixelReplacement Property

bool **PixelReplacement** (read/write)

Description

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 **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.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 Method

bool **Save**(string *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.SetClusterMap Method

bool **SetClusterMap**(string *clusterMapFileName*);

Parameters

clusterMapFileName	Name of cluster map file
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 applies a flat field correction (FFC) algorithm for clusters of defective pixels during FFC calibration. Defective pixel clusters are defined as two or more consecutive bad pixels. The cluster replacement method can compensate for up to 3 consecutive bad pixels in a row (other FFC methods rely on interpolation methods that are effective on single pixels only).

The cluster map file must be loaded before performing the FFC calibration process. The cluster map file format is a simple CSV file, the content of which consists of a list of row, column #n, column #m, .. locations of every bad pixels of the image sensor. Rows and columns are 0 based.

Example: 3 bad pixels on row 0, located at offset 10, 11, 12, and 2 bad pixels on row 2 located at offset 15 and 20:

0, 10, 11, 12

2, 15, 20

CSV Binary file content of the file is as follows:

30 2C 31 30 2C 31 31 2C 31 32 0D 0A 32 2C 31 35 2C 32 30 0D 0A

Note, the hardware FFC must support the cluster correction algorithm, otherwise the correction may not be applied properly.

Demo/Example Usage

Not available

SapFlatField.SetRegionOfInterest Method

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 value of the SoftwareCorrection property is **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

SapFlatField.SetVideoType Method

bool **SetVideoType**(SapAcquisition.VideoType *videoType*, SapColorConversion.ColorAlignMode *alignMode*);

Parameters

videoType	New acquisition video type (SapAcquisition.VideoType.Mono or SapAcquisition.VideoType.Color)
alignMode	Bayer alignment. Only used when *videoType* is set to Color. Possible values are: SapColorConversion.AlignMode.GBRG SapColorConversion.AlignMode.BGGR SapColorConversion.AlignMode.RGGB SapColorConversion.AlignMode.GRBG SapColorConversion.AlignMode.RGBG SapColorConversion.AlignMode.BGRG

Remarks

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.SoftwareCorrection Property

bool **SoftwareCorrection** (read-only)

Description

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.FlatFieldAvailable or SapAcqDevice.FlatFieldAvailable. The SapFlatField.Enable *useHardware* parameter determines if hardware correction is used.

Demo/Example Usage

FlatField Demo, GigE FlatField Demo

SapFlatField.VerticalOffset Property

int **VerticalOffset** (read/write)

Remarks

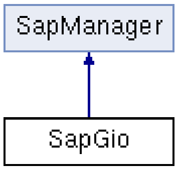
Vertical line scan averaging offset in a full frame.

The initial value for this property 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

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).

Note that acquisition devices do not all support general I/Os.
Namespace: DALSA.SaperaLT.SapClassBasic

SapGio Class Members

Construction	
SapGio	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AvailPinConfig	Possible configurations for all I/O pins
GioNotifyContext	Application specific data for I/O events
Location	Location where the I/O resource is located
NumPins	Number of pins present on the I/O resource
PinConfig	Current configuration for all I/O pins or a specific pin
PinState	Low/high state for all I/O pins or a specific pin
Methods	
DisableEvent	Disables notification of I/O events
EnableEvent	Enables notification of I/O events
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
Events	
GioNotify	Notification of I/O hardware events

SapGio Member Functions

The following are members of the SapGio Class.

SapGio.SapGio (constructor)

SapGio();
SapGio(SapLocation *location***);**

Parameters

location	SapLocation object specifying the server where the I/O resource is located and the index of the resource on this server.

Remarks

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

Demo/Example Usage

Not available

SapGio.AvailPinConfig Property

SapGio.IOPinConfig[] **AvailPinConfig (**read only)

Description

Possible configurations for all I/O pins.

Individual entries in the array (number of entries = value of NumPins property) are set to one or more of the following values (combined using bitwise OR). The entry at index 0 in the array corresponds to the first pin, the entry at index 1 corresponds to the second pin, and so on.

IOPinConfig.Input	I/O pin may be configured as an input
IOPinConfig.Output	I/O pin may be configured as an output
IOPinConfig.Tristate	I/O pin may be tri-stated

Here are examples of how to retrieve this property:

```
SapGio.IOPinConfig[] availConfigs = gio.AvailPinConfig;
```

Note that you can only access this property after calling the Create method.

Demo/Example Usage

Not available

SapGio.Create Method

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

Not available

SapGio.Destroy Method

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

Not available

SapGio.DisableEvent Method

bool DisableEvent();
bool DisableEvent(int *pinNumber*);

Parameters

pinNumber	Pin number on the current I/O resource (from 0 to the value returned by the NumPins property, minus 1)

Remarks

Disables notification of I/O events. You can only call this method after the Create method.

See the GioNotify event for more details.

Demo/Example Usage

Not available

SapGio.Dispose Method

void **Dispose();**

Remarks

Frees unmanaged memory used internally by a SapGio .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapGio object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Not available

SapGio.EnableEvent Method

bool **EnableEvent**(int *pinNumber*, SapGio.EventType *eventType*);
bool **EnableEvent**(int *pinMask*, SapGio.EventType[] *eventType*);
bool **EnableEvent**(SapGio.EventType *eventType*);

Parameters

pinNumber	Pin number on the current input I/O resource
eventType	I/O event type (or array of event types) for which the GioNotify event will occur. One or more of the following values may be combined together using a bitwise OR operation:
	EventType.RisingEdge — Rising edge of I/O pin state transition (low to high)
	EventType.FallingEdge — 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.

Remarks

Enables notification of I/O events.

The first form of EnableEvent may be used using a single input pin number and a corresponding event type. Use this method together with the version of DisableEvent which takes a pin number argument.

The second form takes a bit mask of affected input I/O pins, and an array to specify the event associated with each pin. Entries in the *eventType* array corresponding to bits set to 1 in the *pinMask* argument enable events for the corresponding pins. Bits set to 0 in *pinMask* disable events for the corresponding pins.

The third form enables events for all input pins using the same *eventType*. 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 event handler is invoked. Use this method together with the version of DisableEvent with no arguments.

You can only call this method after the Create method.

See the GioNotify event for more details.

Demo/Example Usage

Not available

SapGio.GetCapability Method

bool **GetCapability**(SapGio.Cap *capId*,out int *capValue*);
bool **GetCapability**(SapGio.Cap *capId*,out SapGio.Val *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the I/O module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapGio class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for *capId*, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORGIO_CAP_DIR_OUTPUT becomes SapGio.Cap.DIR_OUTPUT

You can also use the versions of GetCapability which take a SapGio.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORGIO_VAL_EVENT_TYPE_RISING_EDGE becomes SapGio.Val. EVENT_TYPE_RISING_EDGE

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

Demo/Example Usage

Not available

SapGio.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapGio.Cap *capId*);

Parameters

capId	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapGio.GetParameter, SapGio.SetParameter Methods

bool **GetParameter**(SapGio.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapGio.Prm *paramId*, out SapGio.Val *paramValue*);
bool **GetParameter**(SapGio.Prm *paramId*, out string *paramValue*);

bool **SetParameter**(SapGio.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapGio.Prm *paramId*, SapGio.Val *paramValue*);
bool **SetParameter**(SapGio.Prm *paramId*, string *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read back or to write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the I/O module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter).

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORGIO_PRM_DIR_OUTPUT becomes SapGio.Prm.DIR_OUTPUT

You can also use the versions of GetParameter/SetParameter which take a SapGio.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORGIO_VAL_EVENT_TYPE_RISING_EDGE becomes SapGio.Val. EVENT_TYPE_RISING_EDGE

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 SapGio class. Also, directly setting parameter values may interfere with the correct operation of the class.

Demo/Example Usage

Not available

SapGio.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapGio.Prm *paramId*);

Parameters

paramId	Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.String	Text string

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapGio.GioNotify Event

SapGioNotifyHandler Gio**Notify**

Description

Notifies the application of I/O related hardware events. Use the EnableEvent method to set the hardware events that the application needs to be notified of. Use the GioNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void Gio_GioNotify(Object sender, SapGioNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapGio object for which the event has been registered.

Demo/Example Usage

Not available

SapGio.GioNotifyContext Property

System.Object **GioNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the GioNotify event is invoked. This can be any object instance derived from the System.Object base type. See the GioNotify event description for more details.

Demo/Example Usage

Not available

SapGio.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapGio.Cap *capId*);

Parameters

capId Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera capability for the 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

Not available

SapGio.IsParameterAvailable Method

bool **IsParameterAvailable**(SapGio.Prm *prmId*);

Parameters

prmID Low-level Sapera parameter to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera parameter for the 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.Location Property

SapLocation **Location (**read/write)

Description

Location where the I/O resource is located. A specific location can also be specified through the SapGio constructor.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapGio.NumPins Property

int **NumPins** (read-only)

Description

Number of pins present on the I/O resource. The returned value is only meaningful after you call the Create method.

Demo/Example Usage

Not available

SapGio.PinConfig Property

SapGio.IOPinConfig[] **PinConfig**
SapGio.IOPinConfig **get_PinConfig**(int *pinNumber*)
void **set_PinConfig**(int *pinNumber*, SapGio.IOPinConfig *pinConfig*) (read/write)

Parameters

pinConfig	New pin configuration. See the AvailPinConfig property for possible values.
pinNumber	Pin number on the current I/O resource (from 0 to the value returned by the NumPins property, minus 1)

Description

Current configuration for all I/O pins or a specific pin.

There are two versions of this property: one as an array (number of entries = value of NumPins property), and one with an index.

Here are examples of how to use the indexed property for .NET languages:

```
SapGio.IOPinConfig pinConfig = gio.get_PinConfig(pinNumber);
gio.set_PinConfig(pinNumber, pinConfig);
```

Note that you can only access this property after calling the Create method.

Demo/Example Usage

Not available

SapGio.PinState Property

SapGio.IOPinState [] **PinState**
SapGio.IOPinState **get_PinState**(int *pinNumber*)
void **set_PinConfig**(int *pinNumber*, SapGio.IOPinState *pinState*) (read/write)

Parameters

pinState	New pin state, can be one of the following values:	
	IOPinState.Low	The I/O pin is low
	IOPinState.High	The I/O pin is high
pinNumber	Pin number on the current I/O resource (from 0 to the value returned by the NumPins property, minus 1)	

Description

Low/high state for all I/O pins or a specific pin.

There are two versions of this property: one as an array (number of entries = value of NumPins property), and one with an index.

Here are examples of how to use the indexed property for .NET languages:

```
SapGio.IOPinState pinState = gio.get_PinState(pinNumber);
gio.set_PinState(pinNumber, pinState);
```

Note that you can only access this property after calling the Create method.

Demo/Example Usage

Not available

SapGioNotifyEventArgs

The SapGioNotifyEventArgs class contains the arguments to the application handler method for the SapGio.GioNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapGioNotifyEventArgs Class Members

Properties

AuxTimeStamp	Gets the auxiliary timestamp associated with I/O events
Context	Application context associated with I/O device events
CustomData	Gets the data associated with a custom I/O event
CustomSize	Gets the size of the custom data returned by GetCustomData
EventCount	Current count of I/O events
EventType	I/O events that triggered the invocation of the application event handler
GenericParam0	Gets generic parameters supported by some events
GenericParam1	
GenericParam2	
GenericParam3	
PinNumber	I/O pin number that generated an I/O event
HostTimestamp	Gets the host timestamp associated with I/O events.

SapGioNotifyEventArg Member Properties

The following are members of the SapGioNotifyEventArgs Class.

SapGioNotifyEventArgs.AuxTimeStamp Property

long **AuxTimeStamp** (read-only)

Description

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

SapGioNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with I/O events. See the GioNotifyContext property of the SapGio class for more details.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.CustomData Property

System.IntPtr **Context** (read-only)

Description

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.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.CustomSize Property

int **CustomSize (**read-only)

Description

Size of the custom data returned by the CustomData property.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.EventCount Property

int **EventCount (**read-only)

Description

Current count of I/O events. The initial value is 1 and increments every time the event handler method is invoked.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.EventType Property

SapGio.EventType **EventType (**read-only)

Description

Combination of I/O events that triggered the invocation of the application event handler. Since it is possible for multiple events to trigger one such invocation, this property may actually return a combination of many events, using a bitwise OR operator. See the SapGio.EventType property for the list of possible values.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.GenericParamValue0 SapGioNotifyEventArgs.GenericParamValue1 SapGioNotifyEventArgs.GenericParamValue2 SapGioNotifyEventArgs.GenericParamValue3 Properties

int **GenericParamValue0**
int **GenericParamValue1**
int **GenericParamValue2**
int **GenericParamValue3** (read-only)

Description

Any of the four generic properties supported by some events. See the acquisition device User's Manual for a list of events using generic properties.

Demo/Example Usage

Not available

SapGioNotifyEventArgs.HostTimeStamp Property

long **HostTimeStamp** (read-only)

Description

Host CPU timestamp corresponding to the moment when the event occurred on the host. 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

SapGioNotifyEventArgs.PinNumber Property

int **PinNumber** (read-only)

Description

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.PinState property 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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapLocation Class Members

Construction	
SapLocation	Class constructor
Dispose	Frees unmanaged memory resources
Properties	
ResourceIndex	Resource index
ServerIndex	Server index
ServerName	Server name

SapLocation Member Functions

The following are members of the SapLocation Class.

SapLocation.SapLocation (constructor)

SapLocation();
SapLocation(int *serverIndex*, int *resourceIndex*);
SapLocation(string *serverName*, 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

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.

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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapLocation .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapLocation object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

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.ResourceIndex Property

int **ResourceIndex** (read-only)

Remarks

Resource index.

Demo/Example Usage

Not available

SapLocation.ServerIndex Property

int **ServerIndex** (read-only)

Remarks

Server index. If the returned value is equal to SapLocation.ServerIndexUnknown, it does not necessarily mean that the object is invalid. In this case, use the ServerName property instead.

Demo/Example Usage

Not available

SapLocation.ServerName Property

string **ServerName (**read-only)

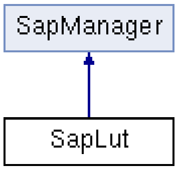
Remarks

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 ServerIndex property instead.

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.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.ApplyLut method.

If you need to manipulate lookup tables on an acquisition device controlled through the SapAcqDevice class (for example, a GigE Vision camera), use theSapAcqDevice.GetLut and SapAcqDevice.ApplyLut methods. Keep in mind that for some devices, the LUT settings do not necessarily match the current output format. For instance, the LUT may actually be 1024 entries with Mono10 format, whereas the camera output format is Mono8. So it is important to remember that the LUT must be programmed according to the number of entries and data format reported by the SapLut object returned by the GetLut function. Any necessary conversions between the LUT data and the actual output format are done internally in SapAcqDevice.

If you need to manipulate display lookup tables, you may use the same technique, but using the SapView.Lut property and SapView.ApplyLut method instead.

Namespace: DALSA.SaperaLT.SapClassBasic

SapLut Class Members

Construction	
SapLut	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
ElementSize	Number of bytes required to store a single LUT element
Format	LUT data format
Location	Location where the LUT resource is located
NumEntries	Number of LUT entries
NumPages	Number of color planes in the LUT
Signed	Checks if the LUT contains signed or unsigned data
TotalSize	Total number of bytes required to store LUT data
Methods	
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
Copy	Copies all LUT entries to another LUT resource
Gamma	Modifies all LUT entries using Gamma correction
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera C library parameter

GetParameterType	Gets the data type of a low-level Sapera parameter
Load	Loads LUT entries from a file
Normal	Modifies all LUT entries using a linear mapping with a positive slope
Read	Reads one or more elements from LUT storage to user-allocated memory
Reverse	Modifies all LUT entries using a linear mapping with a negative slope
Roll	Relocates LUT entries upwards or downwards as one block
Save	Saves LUT entries to a file
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
Write	Writes one or more elements from user-allocated memory to LUT storage

SapLut Member Functions

The following are members of the SapLut Class.

SapLut.SapLut (constructor)

SapLut();
SapLut(int *numEntries*, SapFormat *format*);
SapLut(string *filename*);

Parameters

numEntries Number of LUT entries

format Data format for the LUT resource, can be one of the following values.

Monochrome (unsigned)

SapFormat.Mono8	8-bit
SapFormat.Mono9	9-bit
...	...
SapFormat.Mono15	15-bit
SapFormat.Mono16	16-bit

Monochrome (unsigned)

SapFormat.Int8	8-bit
SapFormat.Int9	9-bit
...	...
SapFormat.Int15	15-bit
SapFormat.Int16	16-bit

Color (non-interlaced)

SapFormat.ColorNI8	8-bit
SapFormat.ColorNI9	9-bit
...	...
SapFormat.ColorNI15	15-bit
SapFormat.ColorNI16	16-bit

Color (interlaced)

SapFormat.ColorI8	8-bit
SapFormat.ColorI9	9-bit
...	...
SapFormat.ColorI15	15-bit
SapFormat.ColorI16	16-bit

loc SapLocation object specifying the server where the LUT resource is located and the index of the resource on this server

filename String containing the name of a Sapera LUT file from which to extract the *numEntries* and *format* 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 use the SapAcquisition.Luts Property and SapAcquisition.ApplyLut method, 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 Method

bool **Arithmetic**(SapLut.ArithmeticOp *operation*, SapData *value*);

Parameters

operation Specifies how to modify LUT data elements. The following operations are available:

ArithmeticOp.Add	Addition with saturation: data[index] = min(maxValue, data[index] + *value*)
ArithmeticOp.Asub	Absolute subtraction: data[index] = abs(data[index] – *value*)
ArithmeticOp.Max	Maximum value: data[index] = max(data[index], *value*)
ArithmeticOp.Min	Minimum value: data[index] = min(data[index], *value*)
ArithmeticOp.Scale	Scale to smaller maximum value: data[index] = (data[index] * *value*) / maxValue
ArithmeticOp.Sub	Subtraction with saturation: data[index] = max(minValue, data[index] – *value)*

value 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 Method

bool **BinaryPattern**(int *bitNumber*, SapData *newValue*);

Parameters

bitNumber	Bit number that identifies the indices of the LUT data elements to modify
newValue	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:

data[index] = (index & (1 << *bitNumber*)) ? *newValue* : data[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 Method

bool **Boolean**(SapLut.BooleanOp *operation*, SapData *value*);

Parameters

operation	Specifies how to modify LUT data elements. The following operations are available:	
	BooleanOp.And	Boolean AND: data[index] = data[index] & *value*
	BooleanOp.Or	Boolean OR: data[index] = data[index] | *value*
	BooleanOp.Xor	Boolean XOR: data[index] = data[index] ^ *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 Method

bool **Copy**(SapLut *srcLut*);

Parameters

srcLut	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 TotalSize property.

Demo/Example Usage

Not available

SapLut.Create Method

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 Method

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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapLut .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapLut object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

SapLut.ElementSize Property

int **ElementSize** (read-only)

Description

Number of bytes required to store a single LUT element.

The initial value for this property is 1. It is then set to the LUT element size value when calling the Create method.

Demo/Example Usage

Not available

SapLut.Format Property

SapFormat **Format** (read/write)

Description

LUT data format. The initial value for this property is SapFormat.Mono8, unless a specific value was specified in the constructor.

You can only change the value of this property before calling the Create method. See the SapLut constructor for possible values for *format*.

Demo/Example Usage

Not available

SapLut.Gamma Method

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.GetParameter, SapLut.SetParameter Methods

bool **GetParameter**(SapLut.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapLut.Prm *paramId*, out SapLut.Val *paramValue*);
bool **GetParameter**(SapLut.Prm *paramId*, out System.IntPtr *paramValue*);

bool **SetParameter**(SapLut.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapLut.Prm *paramId*, SapLut.Val *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read or write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the LUT module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.IntPtr, then *paramValue* is an address (uninitialized for GetParameter).

The following examples show how to declare a variable to hold an address and call GetParameter:

```
System.IntPtr paramValue;
result = lut.GetParameter(paramId, out paramValue)
```

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORLUT_PRM_ADDRESS becomes SapLut.Prm.ADDRESS

You can also use the versions of GetParameter/SetParameter which take a SapLut.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORLUT_VAL_FORMAT_MONO8 becomes SapLut.Val.FORMAT_MONO8

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 SapLut class. Also, directly setting parameter values may interfere with the correct operation of the class.

Demo/Example Usage

Not available

SapLut.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapLut.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter to read or write

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.IntPtr	Address value

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapLut.Load Method

bool **Load**(string *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 NumEntries and Format properties to get their updated values.

Demo/Example Usage

GigE Camera LUT Example, Grab LUT Example

SapLut.Location Property

SapLocation **Location** (read/write)

Remarks

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.Normal Method

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.NumEntries Property

int **NumEntries** (read/write)

Description

Number of LUT entries. The initial value for this property is 256, unless a specific value was specified in the constructor.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapLut.NumPages Property

int **NumPages** (read-only)

Description

Number of color planes in the LUT. The initial value for this property 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.Read Method

bool **Read**(int *elementIndex*, SapData *Value*);
bool **Read**(int *byteOffset*, System.IntPtr *lutData*, int *numBytes*);

Parameters

lutIndex	Index of LUT element to read, starting at 0
Value	Destination value object
byteOffset	Byte offset to start reading from in the LUT.
lutData	Memory area to receive LUT data
numBytes	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 Format, ElementSize, and NumPages properties to find out the LUT data organization.

Demo/Example Usage

Not available

SapLut.Reverse Method

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 Method

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 Method

bool **Save**(string *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 Method

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.Signed Property

bool **Signed** (read-only)

Description

Checks if the LUT contains signed or unsigned data.

The initial value for this property is **false**. It is then set to the LUT signed value when calling the Create method.

Demo/Example Usage

Not available

SapLut.Slope Method

bool **Slope**(int *startIndex*, int *endIndex*, SapData *minValue*, SapData *maxValue*, bool *modifyOutside*);

Parameters

startIndex	Starting LUT index for linear mapping
endIndex	Ending LUT index for linear mapping
minValue	LUT element value at starting index
maxValue	LUT element value at ending index
modifyOutside	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 *modifyOutside* 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 Method

bool **Threshold**(SapData *threshValue*);
bool **Threshold**(SapData *lowValue*, SapData *highValue*);

Parameters

threshValue	Reference value for single threshold
lowValue	Lower reference value for double threshold
highValue	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.TotalSize Property

int **TotalSize()** (read-only)

Description

Total number 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.Write Method

bool **Write**(int *elementIndex*, SapData *value*);
bool **Write** (int *byteOffset*, System.IntPtr *lutData*, int *numBytes*);

Parameters

elementIndex	Index of LUT element to write, starting at 0
value	Source value object
byteOffset	Byte offset to start writing to in the LUT.
lutData	Source memory address for LUT data
numBytes	Number of bytes to write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

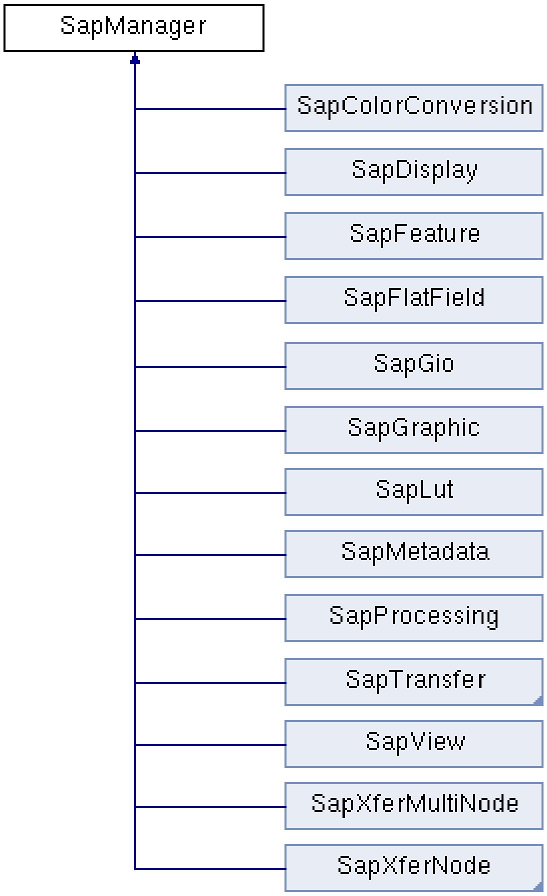
Use the first form of Write to write a single LUT element from either a SapDataMono or SapDataRGB object. You do not have to know how the LUT is stored in memory.

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 Format, ElementSize, and NumPages properties to find out the LUT data organization.

Demo/Example Usage

Not available

SapManager



The SapManager Class includes methods for describing the Sapera resources present on the system. It also includes error management capabilities.

With .NET, this class is declared abstract, which means it cannot be explicitly instantiated.

Namespace: DALSA.SaperaLT.SapClassBasic

SapManager Class Members

Properties

CommandTimeout	Timeout value used when waiting for completion of Sapera LT commands
DisplayStatusMode	Global reporting mode for messages and errors
EndResetContext	Application specific data for end of reset events
ErrorContext	Application specific data for error events
Initialized	Checks whether the Create method has succeeded for a derived object
LastStatusCode	Numeric value of the latest Sapera low-level error
LastStatusMessage	Description of the latest Sapera low-level error
ResetTimeout	Timeout value used when resetting a hardware device
ServerEventType	Currently registered event type for server related events
ServerNotifyContext	Application specific data for server related events
VersionInfo	Sapera LT version and licensing information

Methods

Open	Initializes access to the Sapera LT libraries
Close	Terminates access to the Sapera LT libraries
DetectAllServers	Detects GenCP cameras after a Sapera application has been started
DisplayMessage	Reports a custom message using the current reporting mode
Dispose	Frees unmanaged memory used internally by a Sapera .NET object
GetFormatType	Gets the data type corresponding to a Sapera data format
GetInstallDirectory	Gets the directory where a Sapera product is installed
GetPixelDepthMin, GetPixelDepthMax	Gets the minimum and maximum number of significant bits for a given data format
GetResourceCount	Gets the number of Sapera resources of a specific type on a server
GetResourceIndex	Gets the index of a Sapera resource
GetResourceName	Gets the name of a Sapera resource
GetSerialNumber	Gets the serial number corresponding to a Sapera server
GetSerialPortCount	Gets the number of serial ports on a Sapera server
GetSerialPortComIndex SetSerialPortComIndex	Gets/sets the mapping between a frame grabber serial port to a Windows COM port.
GetServerCount	Gets the number of available Sapera servers
GetServerIndex	Gets the index of a Sapera server
GetServerName	Gets the name of a Sapera server
GetServerType	Gets the type of a Sapera server
IsResourceAvailable	Checks whether a resource is available for use
IsSameLocation	Checks whether two SapLocation objects are the same
IsSameServer	Checks whether two SapLocation objects are located on the same server
IsServerAccessible	Checks if the resources for a server are accessible
IsSystemLocation	Checks whether a SapLocation object is located on the system server
ReadDeviceInfoValue	Returns the value of the specified infoValueName parameter of the device.
ResetServer	Resets the hardware device associated with a specific server
WriteFile	Writes a file to non-volatile memory on the device
Events	
EndReset	Notification that a server reset operation is finished
Error	Notification of Sapera LT .NET library errors
ServerNotify	Notification of server related events (except reset)
ServerFileNotify	Notification of file transfer event

SapManager Member Functions

The following are members of the SapManager Class.

SapManager.Close Method

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

Not available

SapManager.CommandTimeout Property

static int **CommandTimeout** (read/write)

Remarks

Timeout value (in milliseconds) used when waiting for completion of Sapera LT commands. The initial value for this property is 20000 (20 seconds).

If you need to control the timeout value used by the ResetServer method, use the ResetTimeout property instead.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo, Camera Files Example

SapManager.DetectAllServers Method

static bool **DetectAllServers**(DetectServerType *type*);

Parameters

type Specifies the type of server to detect. Possible values are:

DetectServerType.GenCP	Detect GenCP servers only.
DetectServerType.All	Currently equivalent to GenCP only.

Remarks

Use this function to detect GenCP cameras after a Sapera application has been started. In a typical application device detection (discovery) is initiated during application startup. If a GenCP camera is connected after an application has been launched, it will not be detected automatically. Use this function to trigger the device discovery process.

Note that you must register the EventType.ServerNew event before calling this function. See the SapManager.ServerEventType property for details.

Demo/Example Usage

Find Camera example

SapManager.DisplayMessage Method

static void **DisplayMessage**(string *message*);

Parameters

message	Custom message to report

Remarks

Reports a custom message using the current reporting mode. See the DisplayStatusMode property for a description of the available reporting modes.

Note that, when the reporting mode is set to the Log Viewer, messages are logged as errors. It is possible to log these as informational instead by using the case insensitive '(Sapera app)' prefix at the beginning of each message. This prefix will be automatically stripped from the message before logging.

Demo/Example Usage

Not available

SapManager.DisplayStatusMode Property

static SapManager.StatusMode **DisplayStatusMode** (read/write)

Description

Global reporting mode for messages and errors. This mode is used by the DisplayMessage method, and also internally by the Sapera .NET library. It can be one of the following values:

StatusMode.Popup	Sends messages to a popup window
StatusMode.Log	Sends messages to the Sapera Log Server (can be displayed using the Sapera Log Viewer)
StatusMode.Event	Notifies application code through an event
StatusMode.Exception	Notifies application code through an exception
StatusMode.Custom	Error information is not reported, it is just stored internally

The initial value for this property is Popup.

For Event reporting mode, the SapManager.Error event is invoked whenever an error occurs. Also see the description of this event and of the ErrorContext property.

For Exception reporting mode, you get an exception of type SapLibraryException whenever an error occurs. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

For Custom reporting mode, the only way to retrieve error information is by reading the value of the LastStatusCode and/or LastStatusMessage properties.

Note that, for all reporting modes except Exception, any Sapera .NET method returns **false** to indicate an error.

Demo/Example Usage

Camera Features Example, Find Camera Example

SapManager.Dispose Method

Description

Frees unmanaged memory used internally by a Sapera .NET object.

The following classes contains Dispose methods derived from the SapManager class:

- SapAcquisition.Dispose Method
- SapBuffer.Dispose Method
- SapBufferRoi.Dispose Method
- SapBufferWithTrash.Dispose Method
- SapColorConversion.Dispose Method
- SapDisplay.Dispose Method
- SapFeature.Dispose Method
- SapFlatField.Dispose Method
- SapGio.Dispose Method
- SapGraphic.Dispose Method
- SapLocation.Dispose Method
- SapLut.Dispose Method
- SapMetadata.Dispose Method
- SapPerformance.Dispose Method
- SapProcessing.Dispose Method
- SapTransfer.Dispose Method
- SapView.Dispose Method
- SapXferPair.Dispose Method
- SapXferParams.Dispose Method

Demo/Example Usage

All demos and examples

SapManager.EndReset Event

static SapResetHandler **EndReset**

Description

Notifies the application that a server reset operation is finished. This operation is initiated by calling the ResetServer method. Use the EndResetContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapManager_EndReset(Object sender, SapResetEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapManager object for which the event has been registered.

Demo/Example Usage

Not available

SapManager.EndResetContext Property

System.Object **EndResetContext** (read/write)

Description

Supplies application specific data when the application event handler for the EndReset event is invoked. This can be any object instance derived from the System.Object base type. See the EndReset event description for more details.

Demo/Example Usage

Not available

SapManager.Error Event

static SapErrorHandler **Error**

Description

Notifies the application that Sapera.NET library error has occurred as a result of calling one of its methods. Use the ErrorContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapManager_Error(Object sender, SapErrorEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapManager object for which the event has been registered.

Demo/Example Usage

Not available

SapManager.ErrorContext Property

System.Object **ErrorContext** (read/write)

Description

Supplies application specific data when the application event handler for the Error event is invoked. This can be any object instance derived from the System.Object base type. See the Error event description for more details.

Demo/Example Usage

Not available

SapManager.GetFormatType Method

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:

SapFormatType.Unknown	Unable to determine data type
SapFormatType.Mono	Monochrome
SapFormatType.RGB	RGB color
SapFormatType.YUV	YUV color
SapFormatType.HSI	HSI color
SapFormatType.HSV	HSV color
SapFormatType.Color	Lookup table color data
SapFormatType.RGBA	RGB color with an additional component (alpha channel, infrared component, etc.)

Demo/Example Usage

Not available

SapManager.GetInstallDirectory Method

static string **GetInstallDirectory**(int *serverIndex*);
static string **GetInstallDirectory** (string *serverName*);
static string **GetInstallDirectory** (SapLocation *location*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
location	Valid SapLocation object

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.GetPixelDepthMin, SapManager.GetPixelDepthMax Method

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

Multi-Board Sync Grab Demo, Grab LUT Example

SapManager.GetResourceCount Method

static int **GetResourceCount**(int *serverIndex*, SapManager.ResourceType *resourceType*);
static int **GetResourceCount**(string *serverName*, SapManager.ResourceType *resourceType*);
static int **GetResourceCount**(SapLocation *loc*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
serverName	Sapera server name

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 Method

static int **GetResourceIndex**(int *serverIndex*, SapManager.ResourceType *resourceType*, string *resourceName*);
static int **GetResourceIndex**(string *serverName*, SapManager.ResourceType *resourceType*, string *resourceName*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceName	Sapera resource name
serverName	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 this method 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 Method

static string **GetResourceName**(int *serverIndex*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static string **GetResourceName**(string *serverName*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static string **GetResourceName**(SapLocation *loc*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

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.GetSerialPortCount Method

static int **GetSerialPortCount**(int *serverIndex*);
static int **GetSerialPortCount**(string *serverName*);
static int **GetSerialPortCount**(SapLocation *loc*);

Parameters

serverIndex	Sapera server index.
serverName	Sapera server name.
loc	Valid SapLocation object.
serialCount	Number of serial ports available on the selected server.

Remarks

Gets the number of serial ports on a Sapera server. This count is used to determine the valid range, starting at 0, of the *serialIndex* parameter of the SapManager.GetSerialPortComIndex, SetSerialPortComIndex methods.

This only applies to static resources, that is, those attached to physical devices.

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.GetSerialNumber Method

static string **GetSerialNumber**(int *serverIndex*);
static string **GetSerialNumber**(string *serverName*);
static string **GetSerialNumber**(SapLocation *location*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
Loc	Valid SapLocation object

Remarks

Gets a text representation of the serial number corresponding to the hardware device for the specified Sapera server. This is usually a letter followed by seven digits, for example 'S1234567'. It may also contain digits only, or be longer than 8 characters.
Note that there is no serial number associated with the System server. Also, this function is supported for frame grabbers, and GigE Vision / USB3 Vision cameras. When using a CameraLink GenCP camera server, you need a valid CorAcqDevice object from which the serial number can be retrieved through a named feature, this is normally 'DeviceSerialNumber' from GenICam SFNC.

Demo/Example Usage

Not available

SapManager.GetSerialPortComIndex, SetSerialPortComIndex Methods

static bool **SetSerialPortComIndex**(int *serverIndex*, int *serialIndex*, int *comIndex*);
static bool **SetSerialPortComIndex**(string*serverName*, int *serialIndex*, int *comIndex*);
static bool **SetSerialPortComIndex**(SapLocation *loc*, int *serialIndex*, int *comIndex*);

static int **GetSerialPortComIndex**(int *serverIndex*, int *serialIndex*);
static int **GetSerialPortComIndex**(string *serverName*, int *serialIndex*);
static int **GetSerialPortComIndex**(SapLocation *loc*, int *serialIndex*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
serialIndex	Frame grabber serial port index
comIndex	Windows COM port index

Remarks

Gets/sets the mapping between a frame grabber serial port to a Windows COM port. Use the SapManager.GetSerialPortCount method to get the valid range, starting at 0, of the *serialIndex* parameter.

This only applies to static resources, that is, those attached to physical devices.

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.GetServerCount Method

static int **GetServerCount**();
static int **GetServerCount**(SapManager.ResourceType *resourceType*);

Parameters

resourceType Resource type to inquire, can be one of the following:

ResourceType.Acq	Frame grabber acquisition hardware
ResourceType.AcqDevice	Camera acquisition hardware (for example, Genie)
ResourceType.Display	Physical displays
ResourceType.Gio	General inputs and outputs
ResourceType.Graphic	Graphics engine

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.

Demo/Example Usage

IO Demo, Find Camera Example

SapManager.GetServerIndex Method

static int **GetServerIndex**(string *serverName*);
static int **GetServerIndex**(string *deviceUserId*, int *reserved*);
static int **GetServerIndex**(SapLocation *location*);

Parameters

serverName	Sapera server name
deviceUserId	User-specified device ID
location	Valid SapLocation object
reserved	Reserved for future use; must be set to 0.

Remarks

Gets the index of a Sapera server. Returns SapLocation.ServerIndexUnknown if the specified server cannot be found.

The different method prototypes allow the server index to be inquired using the server name, the device user ID or 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 Method

static string **GetServerName**(int *serverIndex*);
static string **GetServerName**(string *deviceUserId*);
static string **GetServerName**(SapLocation *loc*);
static string **GetServerName**(int *serverIndex*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Sapera server index
deviceUserId	User-specified device ID
loc	Valid SapLocation object
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.

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 the device user ID for a camera object. The third form uses an existing SapLocation object with valid server information. The fourth 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

Multi-Board Sync Grab Demo, IO Demo, Find Camera Example

SapManager.GetServerType Method

static SapManager.Server **GetServerType**(int *serverIndex*);
static SapManager.Server **GetServerType**(string *serverName*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name

Return Value Can be one of the following:

Server.None	Server type cannot be determined
Server.System	System server
Server.Bandit3CV	Bandit-3 CV Express VGA frame grabber
Server.X64CL	X64-CL acquisition board
Server.X64CLiPRO	X64-CL iPro acquisition board
Server.X64CLExpress	X64-CL-Express acquisition board
Server.X64CLLX4	X64 Xcelera-CL LX4 acquisition board
Server.X64CLPX4	X64 Xcelera-CL PX4 acquisition board
Server.X64LVDS	X64-LVDS acquisition board
Server.X64LVDSPX4	X64-LVDSPX4 acquisition board
Server.X64LVDSVX4	X64-LVDSVX4 acquisition board
Server.X64ANQuad	X64-AN Quad acquisition board
Server.X64ANLX1	X64-ANLX1 acquisition board
Server.PC2Vision	PC2-Vision acquisition board
Server.PC2Comp	PC2-Comp Express acquisition board
Server.PC2CamLink	PC2-CamLink acquisition board
Server.Genie	Genie camera
Server.AnacondaCL	Anaconda-CL vision processor
Server.AnacondaLVDS	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.Initialized Property

bool **Initialized** (read-only)

Remarks

Checks whether the Create method has succeeded for an object derived from SapManager.

Calling the Destroy method resets this property to **false**.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, Multi-Board Sync Grab Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example

SapManager.IsResourceAvailable Method

static bool **IsResourceAvailable**(int *serverIndex*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static bool **IsResourceAvailable**(string *serverName*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static bool **IsResourceAvailable**(SapLocation *location*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
location	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 the SapAcquisition.Create Method 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 *firstLocation*, SapLocation *secondLocation*);

Parameters

firstLocation	First valid SapLocation object
secondLocation	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 Method

static bool **IsSameServer**(SapLocation *firstLocation*, SapLocation *secondLocation*);

Parameters

firstLocation	First valid SapLocation object
secondLocation	Second valid SapLocation object

Remarks

Checks if the two specified SapLocation objects have the same server information

Demo/Example Usage

IsSameServer

SapManager.IsServerAccessible Method

static bool **IsServerAccessible**(int *serverIndex*);
static bool **IsServerAccessible**(string *serverName*);
static bool **IsServerAccessible**(SapLocation *location*);

Parameters

serverIndex	Index of Sapera server containing the resource
serverName	Name of Sapera server containing the resource
location	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.

This method is useful when a polling mechanism is required to determine accessibility of servers, so it should be called repeatedly within a retry loop. However, the preferred way to determine accessibility of servers is by registering an event handler for the ServerNotify event, and specifying the event type using the ServerEventType property.

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 Method

static bool **IsSystemLocation**();
static bool **IsSystemLocation**(SapLocation *location*);

Parameters

location	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.LastStatusCode Property

static SapStatus **LastStatusCode** (read-only)

Description

Numeric value of the latest Sapera low-level error. See the DisplayStatusMode property for a description of the available error reporting modes.

Note that each thread in a Sapera LT application has its own latest error code. This means that you cannot read this property to retrieve error information for a Sapera .NET function which has been called in another thread.

Demo/Example Usage

Not available

SapManager.LastStatusMessage Property

static string **LastStatusMessage** (read-only)

Description

Description of the latest Sapera low-level error. See the DisplayStatusMode property for a description of the available error reporting modes.

Note that each thread in a Sapera LT application has its own latest error description. This means that you cannot read this property to retrieve error information for a Sapera .NET function which has been called in another thread.

Demo/Example Usage

Not available

SapManager.Open Method

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 garbage collection of the last Sapera LT object. For example:

```
// No Sapera LT calls before this line
// This loads the libraries
SapBuffer buf1 = new SapBuffer();
SapBuffer buf2 = new SapBuffer();
buf1.Dispose();
buf2.Dispose();
// No Sapera LT calls after this line
// Libraries will eventually be unloaded by the garbage collector
```

There is, however, at least one case for which the default behavior may not be acceptable. If the application code frequently frees 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

Not available

SapManager.ReadDeviceInfoValue Method

static string **ReadDeviceInfoValue**(int *serverIndex*,string *infoValueName)*;
static string **ReadDeviceInfoValue**(string *serverName*, string *infoValueName*);
static string **ReadDeviceInfoValue**(SapLocation *loc*, string *infoValueName*);

Parameters

serverIndex	Sapera server index.
serverName	Sapera server name.
loc	Valid SapLocation object.
infoValueName	Device information value name.

Remarks

Returns the value, as a string, of the specified *infoValueName* parameter of the device. The *infoValueName* is the string as it appears in the Sapera Device Manager Information tab, including spaces (for example, "Hardware Configuration). The support for this functionality is device specific.

Demo/Example Usage

Not available

SapManager.ResetServer Method

static bool **ResetServer**(int *serverIndex*, bool *wait*);
static bool **ResetServer**(string *serverName*, bool *wait*);
static bool **ResetServer**(SapLocation *loc*, bool *wait*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
wait	Specifies whether this methgd should return immediately after resetting the specified server, or if it should wait for the server to be operational again

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Resets the hardware frame grabber 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 two ways to use this method:

wait = **true**	Returns only when the reset is complete, and the server is operational again
wait = **false** (EndReset event is not registered)	Returns immediately after resetting the server. The application program is then responsible for figuring out when the server is operational again.
wait = **false** (EndReset event is registered)	Returns immediately after resetting the server, and notifies the application by invoking the EndReset event when the server is operational again

You can read the values of ServerIndex and Context methods of the SapResetEventArgs class to retrieve the required information from the application event handler method.

Note that all other Sapera .NET 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 .NET objects (SapTransfer, SapBuffer, SapAcquisition, ...)
- Call ResetServer
- Call Create for all needed Sapera .NET 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.

Note: this method is only for use with frame grabbers; for cameras use the *DeviceReset* feature to reset the device.

Demo/Example Usage

Not available

SapManager.ResetTimeout Property

static int **ResetTimeout** (read/write)

Description

Timeout value (in milliseconds) used when resetting a hardware device. This value is used by the ResetServer method.

If you need to control the timeout value used when waiting for completion of Sapera LT commands, use the CommandTimeout property instead.

The initial value for this attribute is 20000 (20 seconds).

Demo/Example Usage

Not available

SapManager.ServerEventType Property

static SapManager.EventType **ServerEventType** (read-write)

Description

Registered event type for the ServerNotify event. One or more of the following values may be combined together using a bitwise OR operation:

EventType.ServerNew	A new device is connected while a Sapera application is already running
EventType.ServerDisconnected	The device corresponding to an existing server is disconnected. (Replaces EventType.ServerNotAccessible which is now deprecated.)
EventType.ServerConnected	The device corresponding to an existing, unaccessible server is reconnected. (Replaces EventType.ServerAccessible, which is now deprecated.)
EventType.ServerDatabaseFull	There is no room in the Sapera server database for a new device that has just been connected
EventType.ResourceInfoChanged	The information describing a resource (typically its label) has changed
EventType.ServerFile	The information about the progress of the file being transferred

In the event handler method, you can get the event type through the EventType property of the *args* argument. For all events except ServerDatabaseFull, you can get the index of the server through the ServerIndex property. For the ResourceInfoChanged event, you can get the index of the affected resource through the ResourceIndex property. For all events, you can access application specific data (originally specified by the SapManager.ServerNotifyContext property) through the Context property.

The ResourceInfoChanged event can only occur as a result of modifying the value of the 'DeviceUserID' feature through the SapAcqDevice class.

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.ServerFileNotify Event

static SapServerFileNotifyHandler **ServerFileNotify**

Description

Notifies the application of server related file transfer events. Use the ServerEventType property to set the events that the application needs to be notified of. Use the ServerNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapManager_ServerFileNotify(Object sender, SapServerNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapManager object for which the event has been registered.

Demo/Example Usage

Not available

SapManager.ServerNotify Event

static SapServerNotifyHandler **ServerNotify**

Description

Notifies the application of server related events (except reset, which occurs through the EndReset event). Use the ServerEventType property to set the events that the application needs to be notified of. Use the ServerNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapManager_ServerNotify(Object sender, SapServerNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapManager object for which the event has been registered.

Demo/Example Usage

Not available

SapManager.ServerNotifyContext Property

System.Object **ServerNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the ServerNotify event is invoked. This can be any object instance derived from the System.Object base type. See the ServerNotify event description for more details.

Demo/Example Usage

Not available

SapManager.VersionInfo Property

static SapManVersionInfo **VersionInfo** (read-only)

Description

Sapera LT version and licensing information. The SapManVersionInfo object contains the following read-only properties:

Major	Major version number. For example, if the version number is 6.30.01.0806, the value of this property is 6.
Minor	Minor version number. For example, if the version number is 6.30.01.0806, the value of this property is 30.
Revision	Revision number. For example, if the version number is 6.30.01.0806, the value of this property is 1.
Build	Build number. For example, if the version number is 6.30.01.0806, the value of this property is 806.
LicenseType	Sapera LT license type for the current installation, which can be one of LicenseType.Runtime, LicenseType.Evaluation, or LicenseType.FullSDK
EvalDaysRemaining	Number of days remaining in the evaluation period when the license type is LicenseType.Runtime, where a value equal to 0 means that the evaluation period has expired.

Demo/Example Usage

Not available

SapManager.WriteFile Method

static bool **WriteFile**(int *serverIndex*, String *localFilePath*, int *deviceFileIndex*);
static bool **WriteFile**(String *serverName*, String *localFilePath*, int *deviceFileIndex*);
static bool **WriteFile**(SapLocation *loc*, String *localFilePath*, int *deviceFileIndex*);

Parameters

serverIndex	*Sapera server index*
serverName	Sapera server name
loc	Valid SapLocation object
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. Setting this argument to -1 will automatically use the index stored in the file instead.
localFilePath	Full directory path and filename on the host computer to save the file.

Return Value

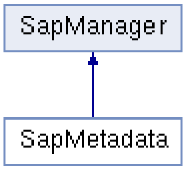
Returns **true** if successful, **false** otherwise

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.

Setting the deviceFileIndex argument to -1 does not work on all devices. If the functions returns an error in this case, you need to explicitly specify the index instead.

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 Method must be called **after** calling the Create function of the SapAcqDevice object and **before** the Create function of the SapBuffer object. In addition, the SapBuffer object must not be constructed with a prototype that uses a SapAcqDevice source node object since the SapMetadata class automatically sizes the buffer to the correct dimensions to support the addition of metadata to the buffer. 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 Method, 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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapMetadata Class Members

Construction	
SapMetadata	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
Enable	Enables metadata in the buffer.
ExtractedResultCount	Gets the number of extracted metadata items.
SelectorCount	Gets the metadata selector count.
Type	Returns the metadata type (frame or line based).
Methods	
Extract	Extracts the selected metadata from the buffer.
GetExtractedResult	Gets the specified metadata.
GetMetadataType	Gets the metadata type for the acquisition device (frame or line based).
GetSelectorName	Gets the specified selector's name.
IsMetadataEnabled	Returns whether metadata is enabled on the acquisition device.
IsMetadataSupported	Returns whether metadata is supported by the acquisition device.
IsSelected	Returns if the specified metadata is selected.
SaveToCVS	Saves the metadata to a comma separated values (CSV) file.
Select	Selects the metadata.

SapMetadata Member Functions

The following are members of the SapMetadata Class.

SapMetadata.SapMetadata (constructor)

SapMetadata(SapAcqDevice *acqDevice*, SapBuffer *buffer*);

Parameters

acqDevice	Acquisition device object
buffer	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 Method

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 method must be called after construction of the SapAcqDevice object and before the construction of the SapBuffer object.

Demo/Example Usage

GigE Metadata Demo GigE Metadata Demo

SapMetadata.Destroy Method

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.Dispose Method

void **Dispose();**

Remarks

Frees unmanaged memory used internally by a SapMetadata .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapMetadata object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Color Conversion Demo, GigE Auto-White Balance Example

SapMetadata.Enable Property

bool **Enable();**

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 Method

bool **Extract**();

bool **Extract**(int *bufferIndex*);

bool **Extract**(int *bufferIndex,* int *lineIndex*);

Parameters

bufferIndex	Buffer index. Possible values are from 0 to (SapBuffer.Count - 1).
lineIndex	Line index. Possible values are from 0 to (SapBuffer.Height - 1).

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Extracts 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(*bufIndex*) prototype.

For line scan acquisition, use the Extract(*bufIndex*, *lineIndex*) prototype.

Use the SapMetadata.Type Property to verify the metadata type (per line or frame).

Demo/Example Usage

GigE Metadata Demo

SapMetadata.ExtractedResultCount Property

int **ExtractedResultCount();**

Return Value

Returns the number of available metadata results. Use the SapMetadata.GetExtractedResult Method to extract the metadata results.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.GetExtractedResult Method

bool **GetExtractedResult**(int *resultIndex*, string *name*, string *value*);

Parameters

resultIndex	Result index. Possible values are from 0 to (SapMetadata.ExtractedResultCount -1).
Name	Metadata item name.
Value	Metadata value.

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Extracts the metadata for the specified result index. Use the SapMetadata.ExtractedResultCount Property to get the number of available metadata results.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.GetMetadataType Method

static MetadataType **GetMetadataType**(SapAcqDevice *acqDevice*);

Parameters

acqDevice	Acquisition device object.

Return Value

Returns the metadata type for the acquisition device. Possible values are:

MedataType.PerFrame	Metadata is inserted per frame.
MedataType.PerLine	Metadata is inserted per line.
MedataType.Unknown	Metadata type is unknown.

Remarks

Gets the metadata type for the acquisition device specified during construction of the SapMetadata object.

This is a static function and can be called without creating a SapMetadata object.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.GetSelectorName Method

bool **GetSelectorName**(int *selectorIndex*, string *name*);

Parameters

selectorIndex	Selector index. Possible values are from 0 to (SapMetadata.SelectorCount -1).
Name	String to hold the metadata 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.SelectorCount Property.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.IsMetadataEnabled Method

static bool **IsMetadataEnabled**(SapAcqDevice *acqDevice*);

Parameters

acqDevice	Acquisition device object.

Return Value

Returns TRUE if metadata is enabled on 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.IsMetadataSupported Method

static bool **IsMetadataSupported**(SapAcqDevice *acqDevice*);

Parameters

acqDevice 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 Method

bool **IsSelected**(int *selectorIndex*);

Parameters

selectorIndex Index of the metadata item. Possible values are from 0 to (SelectorCount -1).

Return Value

Returns TRUE if metadata is enabled for the specified metadata item, FALSE otherwise.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.SaveToCSV Method

bool **SaveToCSV**(string *filename*);

Parameters

filename Name of CSV file to save metadata to.

Return Value

Returns **true** if successful, **false** otherwise.

Remarks

Saves the metadata for the specified buffer as a comma separated values (CSV) file.

Demo/Example Usage

GigE Metadata Demo

SapMetadata.Select Method

bool **Select**(int *selectorIndex*, bool *select*);

Parameters

selectorIndex Index of the metadata item. Possible values are from 0 to (SelectorCount -1).

select Sets the enable stat 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

SapMetadata.SelectorCount Property

int **SelectorCount**();

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.Type Property

MetadataType **Type();**

Return Value

Returns the metadata type. Possible values are:

MedataType.PerFrame	Metadata is inserted per frame.
MedataType.PerLine	Metadata is inserted per line.
MedataType.Unknown	Metadata type is unknown.

Remarks

When the metadata type is Perframe, the SapMetadata.Extract Method (int *bufferIndex*) prototype can be used; for PerLine, use the Extract method (int *bufferIndex*, int *lineIndex*) prototype.

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapPerformance Class Members

Construction	
SapPerformance	Class constructor
Dispose	Frees unmanaged memory resources
Properties	
AutoReset	Controls automatic reset of the internal timer
Time	Number of seconds elapsed since the last timer reset
TimeMicro	Number of microseconds elapsed since the last timer reset
TimeMilli	Number of milliseconds elapsed since the last timer reset
Methods	
Reset	Resets the internal timer

SapPerformace Member Functions

The following are members of the SapPerformance Class.

SapPerformance.SapPerformance (constructor)

SapPerformance();

Remarks

The SapPerformance constructor initializes the internal timer and resets it.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.AutoReset Property

bool **AutoReset (**read/write)

Description

Specifies whether the internal timer should be automatically reset after reading the value of the Time, TimeMilli, or TimeMico properties.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapPerformance .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapPerformance object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.Reset Method

void **Reset**();

Remarks

Resets the internal timer. Reading the value of the Time, TimeMilli, or TimeMicro properties then returns the amount of time elapsed since the reset.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.Time Property

float **Time (**read-only)

Description

Number of seconds elapsed since the last timer reset.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.TimeMicro Property

float **TimeMicro** (read-only)

Description

Number of microseconds elapsed since the last timer reset.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapPerformance.TimeMilli Property

float **TimeMilli** (read-only)

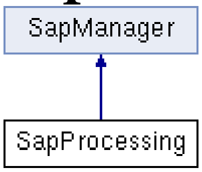
Description

Number of milliseconds elapsed since the last timer 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 event handler method for the SapTransfer.XferNotify event.
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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapProcessing Class Members

Construction	
SapProcessing	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AutoEmpty	Auto-empty mechanism
Buffer	Buffer object with the buffer resources to process
Index	Index of the current or last processed buffer
ProcessingDoneContext	Application context associated with the application callback method
ProcessingDoneEnable	Enables/disables end of processing events
Time	Execution time for the most recently processed buffer
Methods	
Execute	Process the next buffer or a specific one, possibly skipping buffers in the process
ExecuteNext	Process the next buffer, without skipping any buffers in the process
Init	Initializes the processing index
Run	Method overridden in application code to implement custom processing
Events	
ProcessingDone	End of processing notification

SapProcessing Member Functions

The following are members of the SapProcessing Class.

SapProcessing.SapProcessing (constructor)

SapProcessing();
SapProcessing(SapBuffer *buffer*);

Parameters

buffer	Buffer object with the buffer resources to process

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.AutoEmpty Property

bool **AutoEmpty** (read/write)

Description

Auto-empty mechanism, used for synchronizing the transfer and processing tasks in the application program.

By default, the SapTransfer class automatically sets the SapBuffer.State property to SapBuffer.DataState.Empty 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 setting SapTransfer.AutoEmpty to **false**. Then set this property to **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 property is **false**.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.Buffer Property

SapBuffer **Buffer** (read/write)

Description

Buffer object with the buffer resources to process. You can only change the value of this property before calling the Create method.

SapProcessing.Create Method

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 Method

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.Dispose Method

void **Dispose();**

Remarks

Frees unmanaged memory used internally by a SapProcessing .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapProcessing object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.Execute Method

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 from the application event handler method for the SapTransfer.XferNotify event. This will eventually call the Run method in your derived processing class.

The SapProcessing class will then process as many buffers as possible, and possibly skip the processing of some of these in order to avoid the loss of acquired frames through a trash buffer or lost frames from the acquisition device. 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 successfully acquired frames to be processed, call the ExecuteNext method instead.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.ExecuteNext Method

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 from the application event handler method for the SapTransfer.XferNotify event. This will eventually call the Run method in your derived processing class.

The SapProcessing class will then process all successfully acquired frames at the risk of losing some frames through a trash buffer or lost frames from the acquisition device. If the processing task is fast enough to keep-up with the incoming frames, ExecuteNext behaves exactly the same way as Execute. Otherwise, the acquisition rate 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 changing the acquisition rate, use the Execute method instead.

Note that this function does not support the SapXferPair.CycleMode.NextEmpty and SapXferPair.CycleMode.NextWithTrash transfer cycle modes.

Demo/Example Usage

FlatField Demo, GigE FlatField Demo

SapProcessing.Index Property

int **Index** (read-only)

Description

When you read the value of this property from within the Run method of your custom processing class, it returns the index of the current buffer to process. When you read it at any other time, it returns the index of the last processed buffer.

Demo/Example Usage

Not available

SapProcessing.Init Method

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.ProcessingDone Event

SapProcessingDoneHandler **ProcessingDone**

Description

Notifies the application of end of processing events for each buffer. Use the ProcessingDoneEnable property to enable or disable this notification. Use the ProcessingDoneContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void Pro_ProcessingDone(Object sender, SapProcessingDoneEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapProcessing object for which the event has been registered.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.ProcessingDoneContext Property

System.Object **ProcessingDoneContext** (read/write)

Description

Supplies application specific data when the application event handler for the ProcessingDone event is invoked. This can be any object instance derived from the System.Object base type. See the ProcessingDone event description for more details.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.ProcessingDoneEnable Property

bool **ProcessingDoneEnable** (read/write)

Description

Enables/disables end of processing notification events. The initial value for this property is **false**. You can only set the value of this property after calling the Create method.

See the ProcessingDone event for more details.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.Run Method

virtual bool **Run**();

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 Index property to get the index of the buffer to process.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessing.Time Property

float **Time** (read-only)

Remarks

Gets the execution time for the most recently processed buffer (in milliseconds). The initial value for this property is 0.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapProcessingDoneEventArgs

The SapProcessingDoneEventArgs class contains the arguments to the application handler method for the SapProcessing.ProcessingDone event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapProcessingDoneEventArgs Class Members

Properties

Context	Application context associated with end of processing events

SapProcessingDoneEventArgs Member Properties

The following properties are members of the SapProcessingDoneEventArgs Class.

SapProcessingDoneArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with end of processing events. See the ProcessingDoneContext property of the SapProcessing class for more details.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE FlatField Demo

SapResetEventArgs

The SapResetEventArgs class contains the arguments to the application handler method for the SapManager.EndReset event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapResetEventArgs Class Members

Properties

Context	Application context associated with server reset events
ServerIndex	Sapera server index associated with the invocation of the application event handler.

SapResetEventArgs Member Properties

The following properties are members of the SapResetEventArgs Class.

SapResetEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with server reset events. See the EndResetContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapResetEventArgs.ServerIndex Property

int **ServerIndex** (read-only)

Description

Sapera server index associated with the invocation of the application event handler.

Demo/Example Usage

Not available

SapServerFileNotifyEventArgs

The SapServerFileNotifyEventArgs class contains the arguments to the application handler method for the SapManager.ServerFileNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapServerNotifyEventArgs Class Members

Properties

Context	Application context associated with server events
EventType	Server event that triggered the invocation of the application event handler
FilePercentProgress	Returns the file transfer progress, as a percentage of the file size.

SapServerNotifyEventArgs Member Properties

The following properties are members of the SapServerNotifyEventArgs Class.

SapServerFileNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with server events. See the ServerContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapServerFileNotifyEventArgs.EventType Property

SapManager.EventType **EventType** (read-only)

Description

Server event which triggered the invocation of the application event handler. See the SapManager.EventType property for the list of possible values.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.FilePercentProgress Property

int **FilePercentProgress** (read-only)

Description

Demo/Example Usage

Not available

SapServerNotifyEventArgs

The SapServerNotifyEventArgs class contains the arguments to the application handler method for the SapManager.ServerNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapServerNotifyEventArgs Class Members

Properties

Context	Application context associated with server events
EventType	Server event that triggered the invocation of the application event handler
ResourceIndex	Sapera resource index associated with the invocation of the application event handler
ServerIndex	Sapera server index associated with the invocation of the application event handler

SapServerNotifyEventArgs Member Properties

The following properties are members of the SapServerNotifyEventArgs Class.

SapServerNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with server events. See the ServerContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.EventType Property

SapManager.EventType **EventType** (read-only)

Description

Server event which triggered the invocation of the application event handler. See the SapManager.ServerEventType property for the list of possible values.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.ResourceIndex Property

int **ResourceIndex** (read-only)

Description

Sapera resource index associated with the invocation of the application event handler

Demo/Example Usage

Not available

SapServerNotifyEventArgs.ServerIndex Property

int **ServerIndex** (read-only)

Description

Sapera resource index associated with the invocation of the application event handler

Demo/Example Usage

Not available

SapSignalNotifyEventArgs

The SapSignalNotifyEventArgs class contains the arguments to the application handler method for the SapAcquisition.SignalNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapSignalNotifyEventArgs Class Members

Properties

Context	Application context associated with signal status notification events
SignalStatus	Signal status that triggered the invocation of the application event handler

SapSignalNotifyEventArgs Member Properties

The following properties are members of the SapServerNotifyEventArgs Class.

SapSignalNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with signal status events. See the SignalNotifyContext property of the SapAcquisitionr class for more details.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

SapSignalNotifyEventArgs.SignalStatus Property

SapAcquisition.AcqSignalStatus **SignalStatus** (read-only)

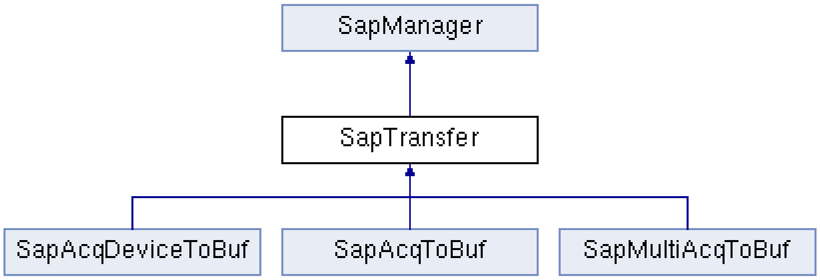
Description

Signal status that triggered the invocation of the application event handler. See the SapAcquisition.IsSignalStatusAvailable method for the list of possible values.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo

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.

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapTransfer Class Members

Construction	
SapTransfer	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AutoConnect	Automatic activation of physical transfer data paths in the Create method
AutoEmpty	Auto-empty mechanism
Connected	Checks whether the physical transfer data paths have been activated
ConnectTimeout	Communication timeout value for the Connect method
CounterStampInfo	Complete list of destination buffer counter stamp capabilities for all transfer pairs
FrameRateStatistics	Returns a pointer to a XferFrameRateInfo object containing frame rate statistics for the associated SapTransfer object
Grabbing	Checks whether continuous data transfer is currently in progress
Location	Location where the transfer resource is located
NumPairs	Number of pairs of source and destination transfer nodes
Pairs	Complete list of transfer pairs
StartMode	Synchronization mode used when starting a data transfer
XferNotifyContext	Application callback context for transfer events
XferTrashNotifyContext	Application callback context for trash buffer transfer events
Methods	
Abort	Stops the data transfer immediately using brute force
AddPair	Adds a new pair of source and destination transfer nodes
Connect	Activates the physical transfer data paths
Disconnect	Deactivates the physical transfer data paths

DisableEvent	Disables all transfer event types
EnableEvent	Enables transfer event types
Freeze	Issues a stop request for continuous data transfer
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
Grab	Starts continuous data transfer
Init	Performs the setup for data transfers
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsCycleModeAvailable	Gets the availability of a specific buffer cycling mode for a specific transfer pair
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
RemoveAllPairs	Removes all transfer pairs
Select	Sets the current source and destination resource indexes
Snap	Transfers a predetermined number of frames
UpdateFrameRateStatistics	Updates the frame rate statistics
Wait	Waits for complete termination of data transfer
Events	
XferNotify	Notification of transfer events
XferTrashNotify	Notification of trash buffer transfer events

SapTransfer Member Functions

The following are members of the SapTransfer Class.

SapTransfer.SapTransfer (constructor)

SapTransfer();

Remarks

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

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 event handler for the XferNotify event is also used for trash buffers, unless you override it using an event handler for the XferTrashNotify event. If you do not use SapBufferWithTrash , then trash buffer settings are ignored.

The event handlers for the XferNotify and XferTrashNotify events apply to all transfer pairs by default, unless you override then for specific pairs by specifying event handlers for their SapXferPair.XferNotify and SapXferPair.XferTrashNotify events.

By default, regular and trash buffer callback event handlers are invoked 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 changing the value of the SapXferPair.EventType property. You cannot change the event type for trash buffers, however.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab 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

SapTransfer.Abort Method

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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab LUT Example

SapTransfer.AddPair Method

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.AutoConnect Property

bool **AutoConnect** (read/write)

Description

Automatic activation of physical transfer data paths. Calling the Create method automatically calls the Connect method when the value of this property is **true**.

Setting this property 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 property is **true**.

Demo/Example Usage

Not available

SapTransfer.AutoEmpty Property

bool **AutoEmpty** (read/write)

Description

Auto-empty mechanism, used for synchronizing the transfer with the processing and/or view tasks in the application program.

By default, this class automatically sets the SapBuffer.State property to SapBuffer.DataState.Empty 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 set the value of this property to **false** to disable auto-empty in this class. You then set the SapProcessing.AutoEmpty or SapView.AutoEmpty property to **true**, depending on which of the 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 manage the value of the SapBuffer.State property 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 property is **true**.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE Camera Compression Demo, GigE FlatField Demo

SapTransfer.Connect Method

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:

```
xfer.Disconnect();
// Modify some transfer parameters
xfer.Connect();
```

This allows the modification of transfer parameters (attributes) through properties 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 use the AutoConnect property to turn off auto-connect, as in the following case:

```
xffer.AutoConnect = false;
xfer.Create();
// Modify some transfer parameters
xfer.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 ConnectTimeout property to increase this value.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE FlatField Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example

SapTransfer.Connected

bool **Connected** (read-only)

Description

Checks whether the physical transfer data paths have been activated. By default, calling the Create method automatically invokes the Connect method, so that the value of this property is **true**. If you set the AutoConnect property to **false** before calling the Create method, then the value of this property is **false**.

If you explicitly call the Connect method, then the value of this property is **true**. If you explicitly call the Disconnect method, then this value is **false**.

The initial value for this property is **false**.

Demo/Example Usage

Not available

SapTransfer.ConnectTimeout Property

int **ConnectTimeout** (read/write)

Description

Communication timeout value (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 (SapManager.CommandTimeout property). In this case, the call to Connect fails with a timeout condition. This property 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 property is 0.

Demo/Example Usage

Not available

SapTransfer.CounterStampInfo Property

SapXferCounterStampInfo[] **CounterStampInfo** (read-only)

Description

Complete list of destination buffer counter stamp capabilities for all transfer pairs.

The returned SapXferCounterStampInfo object has the following read-only properties for each pair:

bool	Supported	Equal to **true** if the current transfer device can report these capabilities
bool	bool Available	Equal to **true** if counter stamp is available
int	MaxValue	Maximum counter stamp value
SapXferPair.XferEventType	EventType	Possible event types (combined using bitwise OR) that identify the reference point for the counter stamp. See the SapXferPair.EventType property for a list of possible values.
SapXferPair.XferCounterStampTimeBase	TimeBase	Possible base units (combined using bitwise OR) used for the counter stamp. See the SapXferPair.CounterStampTimeBase property for a list of possible values.

Here are examples of how to retrieve this property:

```
SapXferCounterStampInfo[] info = xfer.CounterStampInfo;
```

Demo/Example Usage

Not available

SapTransfer.Create Method

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. Setting the AutoConnect property to **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 ConnectTimeout property to increase this value.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab 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

SapTransfer.Destroy Method

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, Multi-Board Sync Grab 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

SapTransfer.DisableEvent Method

bool **DisableEvent**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Disables all registered transfer event types: see the SapXferPair.EventType Property for a description of available events.

Demo/Example Usage

Not available

SapTransfer.Disconnect Method

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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapTransfer .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapTransfer object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab 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

SapTransfer.EnableEvent Method

bool **EnableEvent**(SapXferPair.XferEventType *eventType*);

Parameters

eventType	Low-level command ID

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Enables an transfer event type, or a combination of registered acquisition event types, for which the AcqNotify event will occur. One or more values may be combined together using a bitwise OR operation: see the SapXferPair.EventType Property for a description of available events.

Demo/Example Usage

Not available

SapTransfer.FrameRateStatistics Property

SapXferFrameRateInfo **FrameRateStatistics (read-only)**

Remarks

Gets the SapXferFrameRateInfo object containing the current frame rate statistics. A SapXferFrameRateInfo object is created automatically when a SapTransfer object is constructed. Refer to the SapXferFrameRateInfo class for more information on the available statistics.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo, GigE MetaData Demo

SapTransfer.Freeze Method

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, Multi-Board Sync Grab 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

SapTransfer.GetCapability Method

bool **GetCapability**(SapTransfer.Cap *capId*, out int *capValue*);
bool **GetCapability**(SapTransfer.Cap *capId*, out SapTransfer.Val *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the transfer module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapTransfer class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORXFER_CAP_EVENT_TYPE becomes SapTransfer.Cap.EVENT_TYPE

You can also use the versions of GetCapability which take a SapTransfer.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORXER_VAL_EVENT_TYPE_END_OF_FRAME becomes
SapTransfer.Val.EVENT_TYPE_END_OF_FRAME

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

Demo/Example Usage

GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo

SapTransfer.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapTransfer.Cap *capId*);

Parameters

capId	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapTransfer.GetParameter, SetParameter Method

bool **GetParameter**(SapTransfer.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapTransfer.Prm *paramId*, out SapTransfer.Val *paramValue*);

bool **SetParameter**(SapTransfer.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapTransfer.Prm *paramId*, SapTransfer.Val *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read back or to write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the transfer module.

Use the GetParameterType method to find out which version of GetParameter to use. For the SapTransfer class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORXFER_PRM_EVENT_TYPE becomes SapTransfer.Prm.EVENT_TYPE

You can also use the versions of GetParameter/SetParameter which take a SapTransfer.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORXER_VAL_EVENT_TYPE_END_OF_FRAME becomes
SapTransfer.Val.EVENT_TYPE_END_OF_FRAME

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.

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 SapTransfer class. Also, directly setting parameter values may interfere with the correct operation of the class.

Demo/Example Usage

Not available

SapTransfer.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapTransfer.Prm *paramId*);

Parameters

paramId	Low-level Sapera parameter for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapTransfer.Grab Method

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, Multi-Board Sync Grab 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

SapTransfer.Grabbing Property

bool **Grabbing** (read-only)

Description

Checks whether continuous data transfer is currently in progress. Use the Grab method to initiate continuous transfer.

The value of this property is only relevant after calling the Create method. Otherwise, it is always equal to **false**.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, Grab Demo

SapTransfer.Init Method

bool **Init**(bool *resetIndex*);

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.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapTransfer.Cap *capId*);

Parameters

capId	Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.IsCycleModeAvailable Method

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 NumPairs property minus 1.

See the SapXferPair.Cycle method for a list of valid values for the *cycleMode* argument..

Demo/Example Usage

Not available

SapTransfer.IsParameterAvailable Method

bool **IsParameterAvailable**(SapTransfer.Prm *prmId*);

Parameters

prmId	Low-level Sapera parameter to check

Return Value

Returns **true** if the parameter is supported, **false** otherwise

Remarks

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

Nothe 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.Location Property

SapLocation **Location** (read/write)

Description

Location where the transfer resource is located.

The initial value for this property is SapLocation.ServerIndexUnknown. 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.NumPairs Property

int **NumPairs** (read-only)

Description

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.Pairs Property

SapXferPair[] **Pairs** (read-only)

Description

Complete list of transfer pairs. Here are examples of how to retrieve this property:

```
SapXferPair[] allPairs = xfer.Pairs;
if (allPairs != null) …
```

Note that the examples check for a null value for this property, which is the case if no pairs are currently defined, that is, the value of the NumPairs property returns 0.

See the SapXferPair Class for more details.

Demo/Example Usage

Not available

SapTransfer.RemoveAllPairs Method

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 Method

bool **Select**(int *pairIndex*);
bool **Select**(int *pairIndex*, int *srcIndex*, int *destIndex*);
bool **Select**(SapXferPair *pair*);
bool **Select**(SapXferPair *pair*, int *srcIndex*, int *destIndex*);

Parameters

pairIndex	Index of new transfer pair
srcIndex	New resource index for source transfer node
dstIndex	New resource index for destination transfer node
Pair	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. It is also possible to specify the SapXferPair object to select, instead of its index

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 *destIndex* 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 through properties 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.Snap Method

bool **Snap**();
bool **Snap**(int *count*);

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 if using the version of this method with no arguments) 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 StartMode property 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 the SapXferPair.FramesPerCallback property, the value of *count* should be a multiple of the number of frames per callback, otherwise, the application behavior is undefined. Typically, the application event handler function might not get invoked for any leftover frames. For example, if you acquire 10 frames and the value of FramesPerCallback is 4, then you may not get the application event handler 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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example

SapTransfer.StartMode Property

SapTransfer.XferStartMode **StartMode** (read/write)

Description

Synchronization mode used when starting a data transfer using the Snap method, which can be one of the following values:

XferStartMode.Asynchronous	Return immediately without waiting for the transfer to begin
XferStartMode.Synchronous	For single frame transfers, first wait for any active transfer to end, and return only when the current transfer has been completed.
XferStartMode.HalfAsynchronous	For single frame transfers, first wait for any active transfer to end, then immediately return without waiting for the current transfer to begin.
XferStartMode.Sequential	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.

Note that, when using the Synchronous mode, you should always register transfer end of frame events, otherwise you may get intermittent issues when doing custom processing after the SapTransfer.Snap function returns (for example, invalid buffer index when reading the SapBuffer.Index property).

The default value for this property is Asynchronous. You can only change its value before calling the Create method.

Demo/Example Usage

Not available

SapTransfer.UpdateFrameRateStatistics Method

bool **UpdateFrameRateStatistics**();

Description

Updates the frame rate statistics contained in the associated SapXferFrameRateInfo object. A SapXferFrameRateInfo object is created automatically when a SapTransfer object is constructed.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo, GigE MetaData Demo

SapTransfer.Wait Method

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

SapTransfer.XferNotify Event

SapXferNotifyHandler **XferNotify**

Description

Notifies the application of transfer events. For each transfer pair (SapXferPair) which belongs to the transfer object, use the SapXferPair.EventType property to set the events that the application needs to be notified of. Use the XferNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void Xfer_XferNotify(Object sender, SapXferNotifyEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapTransfer object for which the event has been registered.

If a SapBufferWithTrash object is used as a destination node for a transfer pair, the event handler for the XferNotify event is also used for trash buffers, unless you override it using an event handler for the XferTrashNotify event.

The event handlers for the XferNotify event applies to all transfer pairs by default, unless you override it for specific pairs by specifying event handlers for their SapXferPair.XferNotify and/or SapXferPair.XferTrashNotify events.

Demo/Example Usage

Not available

SapTransfer.XferNotifyContext Property

System.Object **XferNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the XferNotify event is invoked. This can be any object instance derived from the System.Object base type. See the XferNotify event description for more details.

Demo/Example Usage

Not available

SapTransfer.XferTrashNotify Event

SapXferNotifyHandler **XferTrashNotify**

Description

Notifies the application of trash buffer transfer events, thus overriding the XferNotify event. This applies only to transfer pairs (SapXferPair) for which the destination node is a SapBufferWithTrash object. Also, only end of frame events (SapXferPair.XferEventType.EndOfFrame) are available for trash buffers.

Use the XferTrashNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

static void Xfer_XferTrashNotify(Object *sender*, SapXferNotifyEventArgs *args*)

The event handlers for the XferTrashNotify event applies to all transfer pairs by default, unless you override it for specific pairs by specifying event handlers for their SapXferPair.XferTrashNotify events.

Demo/Example Usage

Not available

SapTransfer.XferTrashNotifyContext Property

System.Object **XferTrashNotifyContext** (read/write)

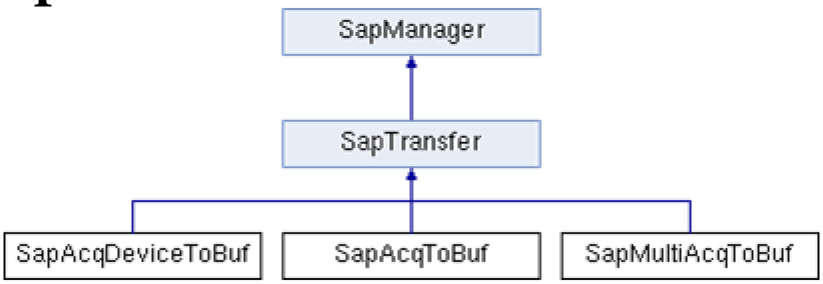
Description

Supplies application specific data when the application event handler for the XferTrashNotify event is invoked. This can be any object instance derived from the System.Object base type. See the XferTrashNotify event description for more details.

Demo/Example Usage

Not available

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.

SapAcqToBuf Class

SapAcqToBuf(SapAcquisition *acq*, SapBuffer *buf*);

Parameters

acq	Source acquisition object
buf	Destination buffer object

Remarks

Implements a transfer from an acquisition object to a buffer object

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, Grab Demo, Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapAcqDeviceToBuf Class

SapAcqDeviceToBuf(SapAcquisition *acqDevice*, SapBuffer *buf*);

Parameters

acqDevice	Source acquisition device object
buf	Destination buffer object

Remarks

Implements a transfer from an acquisition device object (for example, for 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[] *acq*, SapBuffer[] *buf*, int *numPairs*);

Parameters

acq	List of source acquisition objects
buf	List of destination buffer object
numPairs	Number of entries in acquisition and buffer lists

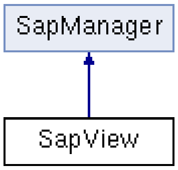
Remarks

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapView Class Members

Construction	
SapView	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AutoEmpty	Auto-empty mechanism
Buffer	Buffer object with the buffer resources to display
BufferAreaHeight	Height and width of the displayed buffer area
BufferAreaWidth	
Display	Display object with the display device associated with the view
DisplayDoneContext	Application context associated with the application callback method
DisplayDoneEnable	Enables/disables end of display events
HasRange	Checks if the view resource can show a subrange of buffer data bits
HorzScrollPosition	Current scrolling position of the viewing area relative to buffer coordinates
VertScrollPosition	
HorzScrollRange	Scrolling range of the viewing area relative to buffer coordinates
VertScrollRange	
ImmediateMode	View thread bypass mode
Index	Index of the last displayed buffer
Lut	Current view lookup table
Range	Gets the number of pages in a planar or multi-format buffer.
RangeMax	Gets/sets the page of a planar or multi-format buffer to display.
RangeMin	Gets the SapFormat of the current page of a planar or multi-format buffer.
ScalingMode	Viewing range value
ScalingDestArea	Maximum and minimum viewing range value
ScalingSrcArea	
ScalingZoomHorz	Gets the mode specifying how buffer content is scaled to the viewing area

ScalingZoomVert	Coordinates and dimensions of the viewing area
ViewAreaHeight	Coordinates and dimensions of the displayed buffer area
ViewAreaWidth	Horizontal and vertical zooming factors
Window	
WindowTitle	Height and width of the viewing area
SapView	
Create	.NET GUI control object used for showing buffers
Destroy	Title of view windows automatically created by SapView
Methods	
ApplyLut	Programs a new view lookup table
FindView	Finds the SapView object corresponding to a .NET form object
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetGraphics	Gets the .NET graphics object corresponding to the view area
GetParameter	Gets/sets the value of a low-level Sapera C library parameter
SetParameter	
GetParameterType	Gets the data type of a low-level Sapera parameter
Init	Initializes the view index
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
OnHScroll	Adjusts the horizontal scrolling position following a form horizontal scroll .NET event
OnMove	Adjusts the position of the viewing window following a form move .NET event
OnPaint	Shows the last displayed buffer again following a form paint .NET event
OnSize	Adjusts the size of the viewing window following a form resize .NET event
OnVScroll	Adjusts the vertical scrolling position following a form vertical scroll .NET event
ReleaseGraphics	Releases the .NET graphics object corresponding to the view window
SetScalingMode	Sets the mode specifying how buffer content is scaled to the viewing area
Sh-w	Shows the next buffer or a specific one, possibly skipping buffers in the process
ShowNext	Shows the next buffer, without skipping any buffers in the process
Events	
DisplayDone	End of image display notification

SapView Member Functions

The following are members of the SapView Class.

SapView.SapView (constructor)

SapView()
SapView(SapBuffer *buffer*)
SapView(SapBuffer *buffer,*System.Windows.Forms.Control *control*)
SapView(SapDisplay *display*, SapBuffer *Buffer*);**SapView**(SapDisplay *display*, SapBuffer *buffer,*System.Windows.Forms.Control *control*)

Parameters

buffer	Buffer object with the buffer resources to display
control	.NET GUI control object used for displaying buffers
display	Display object specifying on which display resource the buffers will be shown

Remarks

The SapView constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.If you use the *control* argument, then the corresponding .NET GUI control will be used for displaying buffer contents. This control is any .NET class derived from the **Control** class in the **System.Windows.Forms** namespace, for example, the **Form** class. If you do not use the *control* argument, then SapView will automatically create a view window (supported on single monitor configurations only). This is especially useful in console applications, where you do not have a full GUI at your disposal.

If you do not specify the *display* 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 *display* argument is to speed up creation of the display object, and to eliminate possible related flicker effects.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapView.ApplyLut Method

bool **ApplyLut**();

Return Value

Returns TRUE if successful, FALSE otherwise.

Remarks

Reprograms the view lookup table. After getting the current LUT using the Lut property, 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 the SapDisplay.PrimaryVGABoard property).

Demo/Example Usage

Not available

SapView.AutoEmpty Property

bool **AutoEmpty (**read/write)

Description

Auto-empty mechanism, used for synchronizing the transfer and view tasks in the application program.

By default, the SapTransfer class automatically sets the SapBuffer.State property to SapBuffer.DataState.Empty 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 setting SapTransfer.AutoEmpty to **false**. Then set this property to **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 property is **false**.

Demo/Example Usage

Not available

SapView.Buffer Property

SapBuffer **Buffer (**read/write)

Description

Buffer object with the buffer resources to display. You set the initial value for this property through the SapView constructor.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Color Conversion Demo, Color Split Example

SapView.BufferAreaHeight, SapView.BufferAreaWidth Property

int **BufferAreaHeight**
int **BufferAreaWidth (**read-only)

Remarks

Height (in lines) and width (in pixels) of the displayed buffer area. The height is equal to the minimum of the buffer height and the viewing area height. The width is equal to the minimum of the buffer width and the viewing area width.

The value returned by these properties are only relevant after calling the Create method.

Demo/Example Usage

Not available

SapView.Create Method

bool **Create**();

Return Value

Returns **true** if successful created, **false** otherwise

Remarks

Creates all the low-level Sapera 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 Method. See the SapView constructor for more details.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapView.Destroy Method

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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapView.Display Property

SapDisplay **Display** (read/write)

Description

Display object specifying where the buffer contents are shown.

If you explicitly specify a SapDisplay object in the SapView constructor, then this property 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, this property returns the internal object.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, GigE Camera Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo

SapView.DisplayDone Event

SapDisplayDoneHandler **DisplayDone**

Description

Notifies the application of end of image display for each buffer. Use the DisplayDoneEnable property to enable or disable this notification. Use the DisplayDoneContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void View_DisplayDone(Object sender, SapDisplayDoneEventArgs args)
```

The sender argument represents the object instance which fired the event. Since all .NET classes are derived from System.Object, this can actually be any class. In this case, this argument can be cast to the SapView object for which the event has been registered.

Demo/Example Usage

Not available

SapView.DisplayDoneContext Property

System.Object **DisplayDoneContext** (read/write)

Description

Supplies application specific data when the application event handler for the DisplayDone event is invoked. This can be any object instance derived from the System.Object base type. See the DisplayDone event description for more details.

Demo/Example Usage

Not available

SapView.DisplayDoneEnable Property

bool **DisplayDoneEnable** (read/write)

Description

Enables/disables end of image display events. The initial value for this property is **false**. You can only set the value of this property before calling the Create method.

See the DisplayDone event for more details.

Demo/Example Usage

Not available

SapView.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapView .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapView object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, Color Split Example, File Load Console, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapView.FindView Method

static SapView **FindView**(System.Windows.Forms.Form *form*);

Parameters

Form	.NET form object

Return Value

Returns an appropriate SapView object, or a null value if none is found

Remarks

Finds the SapView object corresponding to a .NET **Form** object in the **System.Windows.Forms** namespace.

If you supplied a **Form** object (derived from **Control**) in the SapView contructor, then this method returns this object.

If no form is currently associated with the view, then this method returns a null value. This happens if you did not supply a **Form** object in the constructor, or if you explicitly set the value of the Window property to a null value. In both cases, SapView automatically creates a view window, which is not a form.

Demo/Example Usage

Not available

SapView.GetCapability Method

bool **GetCapability**(SapView.Cap *capId*,out int *capValue*);
bool **GetCapability**(SapView.Cap *capId*,out SapView.Val *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the view module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapView class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for capId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORVIEW_CAP_LUT_ENABLE becomes SapView.Cap.LUT_ENABLE

You can also use the versions of GetCapability which take a SapView.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORVIEW_VAL_MODE_DIB becomes SapView.Val.MODE_DIB

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

Demo/Example Usage

Not available

SapView.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapView.Cap *capId*);

Parameters

capId	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

Demo/Example Usage

Not available

SapView.GetGraphics Method

System.Drawing.Graphics **GetGraphics**();

Return Value

.NET graphics object if successful, or a null value otherwise

Remarks

Gets the .NET graphics object corresponding to the view area.

If the current SapView object does not use the system display (see the SapDisplay.Type property), then this method returns a graphics object corresponding to the entire display instead.

Use the ReleaseGraphics method when you have finished using the graphics object.

Demo/Example Usage

Color Conversion Demo

SapView.GetParameter, SapView.SetParameter Methods

bool **GetParameter**(SapView.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapView.Prm *paramId*, out SapView.Val *paramValue*);
bool **GetParameter**(SapView.Prm *paramId*, out System.IntPtr *paramValue*);
bool **GetParameter**(SapView.Prm *paramId*, out string *paramValue*);

bool **SetParameter**(SapView.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapView.Prm *paramId*, SapView.Val *paramValue*);
bool **SetParameter**(SapView.Prm *paramId*, System.IntPtr *paramValue*);
bool **SetParameter**(SapView.Prm *paramId*, string *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read back or to write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the view module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter). If this value is SapCapPrmType.IntPtr, then *paramValue* is an address (uninitialized for GetParameter).

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for paramId, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORVIEW_PRM_LUT_ENABLE becomes SapView.Prm.LUT_ENABLE

You can also use the versions of GetParameter/SetParameter which take a SapView.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORVIEW_VAL_MODE_DIB becomes SapView.Val.MODE_DIB

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 SapView class. Also, directly setting parameter values may interfere with the correct operation of the class.

Demo/Example Usage

Not available

SapView.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapView.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.String	Text string
SapCapPrmType.IntPtr	Address value

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

Demo/Example Usage

Not available

SapView.HasRange Property

bool **HasRange** (read-only)

Description

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 Range property to set the viewing range value.

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

Demo/Example Usage

Not available

SapView.HorzScrollPosition, SapView.VertScrollPosition Property

int **HorzScrollPosition**
int **VertScrollPosition** (read-only)

Description

Current scrolling position of the viewing area relative to buffer coordinates. The initial value of both these properties is 0, and changes automatically through calls to the OnHScroll and OnVScroll methods. The maximum value depends on the scrolling range (see the SapView.HorzScrollRange and SapView.VertScrollRange properties).

Depending on the current view scaling mode, both scrolling positions remain fixed at 0 if the buffer contents fit entirely within the view area.

The value returned by these properties is only relevant after calling the Create method.

See the ScalingMode property and SetScalingMode method for details.

Demo/Example Usage

Demos Common Files

SapView.HorzScrollRange, SapView.VertScrollRange Property

int **HorzScrollRange**
int **VertScrollRange** (read-only)

Description

Scrolling range 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 horizontal and vertical values are both 0, then scrolling is disabled.

The value returned by these properties is only relevant after calling the Create method.

See the ScalingMode property for and SetScalingMode method details.

Demo/Example Usage

Demos Common Files

SapView.ImmediateMode Property

bool **ImmediateMode** (read/write)

Description

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 property is **false**.

Demo/Example Usage

Not available

SapView.Index Property

int **Index** (read-only)

Description

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.Init Method

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.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapView.Cap *capId*);

Parameters

capId	Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.IsParameterAvailable Method

bool **IsParameterAvailable**(SapView.Prm *prmId*);

Parameters

prmID	Low-level Sapera parameter to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

Checks for the availability of a low-level Sapera 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.Lut Property

SapLut **Lut** (read-only)

Description

Current view lookup table, which has already been automatically created and initialized when calling the Create method. You may manipulate it through the methods in the SapLut Class, and reprogram it using the ApplyLut method.

This property has a null value 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 the SapDisplay.PrimaryVGABoard Property).

Demo/Example Usage

Not available

SapView.NumPages Property

int **NumPages** (read-only)

Remarks

Gets the number of pages in a planar or multi-format buffer.

Demo/Example Usage

Not available

SapView.OnHScroll Method

void **OnHScroll**(int *position*);

Parameters

position	New horizontal scrolling position

Remarks

Call this method from your application .NET event handler for the horizontal scroll event to adjust the horizontal scrolling position.

Demo/Example Usage

Not available

SapView.OnMove Method

void **OnMove**();

Remarks

Call this method from your application .NET event handler for the move event to adjust the position of the viewing window

Demo/Example Usage

Not available

SapView.OnPaint Method

void **OnPaint**();

Remarks

Call this method from your application .NET event handler for the paint event to show the last displayed buffer again.

Demo/Example Usage

Not available

SapView.OnSize Method

void **OnSize**();

Remarks

Call this method from your application .NET event handler for resize event to adjust the size of the viewing window

Demo/Example Usage

Not available

SapView.OnVScroll Method

void **OnVScroll**(int *position*);

Parameters

position	New vertical scrolling position

Remarks

Call this method from your application .NET event handler for the vertical scroll event to adjust the vertical scrolling position.

Demo/Example Usage

Not available

SapView.Page Property

int **Page** (read/write)

Remarks

Gets/sets the page of a planar or multi-format buffer to display.

Demo/Example Usage

Not available

SapView.PageFormat Property

SapFormat **PageFormat** (read-only)

Remarks

Gets the SapFormat of the current page of a planar or multi-format buffer.

Demo/Example Usage

Not available

SapView.Range Property

int **Range** (read/write)

Description

Viewing range value. Before using this property, you should first check for availability of this feature using the HasRange and the RangeMin/RangeMax properties.

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 read or write the value of this property after calling the Create method.

Demo/Example Usage

Demo Common Files

SapView.RangeMax, SapView.RangeMin Property

int **RangeMax**
int **RangeMin** (read-only)

Description

Gets the maximum and minimum viewing range values allowed for the Range property. If both values are 0, then you cannot change the range.

You can only reda the values of these properties after calling the Create method.

Demo/Example Usage

Demo Common Files

SapView.ReleaseGraphics Method

bool **ReleaseGraphics**(System.Drawing.Graphics *graphic*);

Parameters

graphic	.NET graphics object

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Releases the .NET graphics object corresponding to the current view area. This object was previously allocated using the GetGraphics method.

If the current SapView object does not use the system display (see the SapDisplay.Type property), then this method releases the .NET graphics object corresponding to the entire display instead.

Demo/Example Usage

Demo Common Files

SapView.ScalingMode Property

SapView.DisplayScalingMode **ScalingMode** (read-only)

Description

Mode specifying how buffer content is scaled to the viewing area, which can be one of the following values

DisplayScalingMode.None	There is a one-to-one correspondence between buffer data and pixels shown in the view area. This is the default mode.
DisplayScalingMode.FitToWindow	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.
DisplayScalingMode.Zoom	Displayed buffer contents are scaled independently in the horizontal and vertical directions
DisplayScalingMode.UserDefined	Buffer contents are displayed using custom user-specified settings

Note that this property only allows reading the value of the current scaling mode. Use the SetScalingMode method in order to select a new mode.

The initial value for this property is None.

Demo/Example Usage

Demo Common Files

SapView.ScalingDestArea Property

System.Drawing.Rectangle **ScalingDestArea** (read-only)

Description

Coordinates and dimensions of the viewing area. Use the SetScalingMode method in order to directly or indirectly set the value of this property.

Demo/Example Usage

Demo Common Files

SapView.ScalingSrcArea Property

System.Drawing.Rectangle **ScalingSrcArea** (read-only)

Description

Coordinates and dimensions of the displayed buffer area. Use the SetScalingMode method in order to directly or indirectly set the value of this property.

Demo/Example Usage

Demo Common Files

SapView.ScalingZoomHorz, SapView.ScalingZoomVert Property

float **ScalingZoomHorz**
float **ScalingZoomVert** (read-only)

Description

Horizontal and vertical zooming factors. Use the SetScalingMode method in order to directly or indirectly set the value of this property.

Demo/Example Usage

Demo Common Files

SapView.SetScalingMode Method

bool **SetScalingMode**();
bool **SetScalingMode**(bool *keepAspectRatio*);
bool **SetScalingMode**(float *zoomHorz*, float *zoomVert*);
bool **SetScalingMode**(System.Drawing.Rectangle *srcArea*, System.Drawing.Rectangele *destArea*);

Parameters

keepAspectRatio	Specifies whether to keep the image aspect ratio when using FitToWindow mode
zoomHorz	Horizontal zooming factor to apply to displayed buffer contents
zoomVert	Vertical zooming factor to apply to displayed buffer contents
srcArea	Buffer area to be shown in the specified region of the viewing area
destArea	Region of the viewing area that will show the specified buffer area

Remarks

The first form of this method allows you to specify a one-to-one relationship between buffer contents and the view area (SapView.ScalingMode property = SapView.DisplayScalingMode.None).

The second form allows you to display buffer contents completely (SapView.ScalingMode property = SapView.DisplayScalingMode.FitToWindow).

The third form allows you to specify independent horizontal and vertical scaling factors (SapView.ScalingMode property = SapView.DisplayScalingMode.Zoom). These apply to displayed images only, they do not affect buffer data. This results in distorted images if the factors are different.

The fourth form gives you complete control over the scaling mode (SapView.ScalingMode property = SapView.DisplayScalingMode.UserDefined). You need to specify the exact rectangular regions in the source buffer and in the destination view area. SapView then automatically calculates the appropriate horizontal and vertical scaling factors.

After calling this method, use the following read-only properties to read back the scaling mode and its associated settings: ScalingMode, ScalingDestArea, ScalingSrcArea, ScalingZoomHorz, and ScalingZoomVert.

For multiformat buffers (for example, SapFormat.RGB888_MONO8 or RGB161616_MONO16) the SapBuffer.Page Property determines which part (RGB or Mono) of the buffer is used.

Demo/Example Usage

Multi-Board Sync Grab Demo

SapView.Show Method

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.Count Property property 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, SapFormat.RGB888_MONO8 or RGB161616_MONO16) the SapBuffer.Page Property determines which part (RGB or Mono) of the buffer is displayed. There is no noeed to call SapView.Destroy or Create when switching buffer pages.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab 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

SapView.ShowNext Method

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

SapView.ViewAreaHeight, SapView.ViewAreaWidth Property

int **ViewAreaHeight**
int **ViewAreaWidth** (read-only)

Description

Height and width of the viewing area. The value returned by these properties is only relevant after calling the Create method.

See also the BufferAreaWidth and BufferAreaHeight properties.

Demo/Example Usage

Not available

SapView.Window Property

System.Windows.Forms.Control **Window** (read/write)

Description

.NET GUI control object used for showing buffers.

You may set the value of this property to an instance of the **Control** class in the **System.Windows.Forms** namespace (or a derived class, for example, **Form**).

If you use a null value instead for this property, then SapView will automatically create a view window (supported on single monitor configurations only). This 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 property in the SapView constructor (throught the *control* argument), then it defaults to null.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Color Split Example

SapView.WindowTitle Property

string **WindowTitle** (read/write)

Remarks

Title of view windows automatically created by SapView.

You can only read or write the value of this property after calling the Create method.

Demo/Example Usage

Not available

SapXferFrameRateInfo

```
SapXferFrameRateInfo
```

The SapXferFrameRateInfo Class provides frame rate statistics for the associated SapTransfer object. The SapXferFrameRateInfo object is created automatically when constructing a SapTransfer object. Therefore you should not instantiate SapXferFrameRateInfo objects directly.

Namespace: DALSA.SaperaLT.SapClassBasic

SapXferFrameRateInfo Class Members

Construction	
SapXferFrameRateInfo	Class constructor
Properties	
BufferFrameRate	Returns the calculated frame rate
IsBufferFrameRateAvailable	Checks if the buffer frame rate is available
IsLiveFrameRateAvailable	Checks if live frame rate from timer is available
IsLiveFrameRateStalled	Checks if live frame rate calculation is stalled
LiveFrameRate	Returns the approximate real-time frame rate
MaxTimePerFrame	Returns the maximum time between consecutive frames
MinTimePerFrame	Returns the minimum time between consecutive frames
Methods	
Reset	Resets the frame rate calculator

SapXferFrameRateInfo Member Functions

The following functions are members of the SapXferFrameRateInfo Class.

SapXferFrameRateInfo.SapXferFrameRate (constructor)

SapXferFrameRateInfo();

Description

The SapXferFrameRateInfo object is created automatically when constructing a SapTransfer object therefore you should not instantiate SapXferFrameRateInfo objects directly.

Demo/Example Usage

Not available

SapXferFrameRateInfo.BufferFrameRate Property

float **BufferFrameRate** (read-only)

Description

Returns the frame rate calculated from acquisition buffer timestamps, in frames per second.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.IsBufferFrameRateAvailable Property

bool **IsBufferFrameRateAvailable** (read-only)

Description

Checks if the frame rate calculated from buffer acquisition timestamps is available.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.IsLiveFrameRateAvailable Property

bool **IsLiveFrameRateAvailable** (read-only)

Description

Checks if live frame rate calculation is available. The SapXferFrameRateInfo.LiveFrameRate Property returns the calculated from rate. The live frame rate calcuation is used when the timebase is in units other than time (for example, shaft encoder ticks).

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.IsLiveFrameRateStalled Property

bool **IsLiveFrameRateStalled** (read-only)

Description

Checks if the live frame rate calculation is stalled. This can occur if no new frame is received for 2 seconds.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.LiveFrameRate Property

float **LiveFrameRate** (read-only)

Description

Returns the approximate live frame rate, calculated using the performance times in the SapPerformance, when the timebase is not not a physical measure of time (for example, shaft encoder ticks).

To determine the timebase used by frame grabbers use the SapAcquisition.TimeStampBase Property. For feature-based devices use the SapAcqDevice.GetFeatureValue Method to get the *timestampSource* feature setting.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.MaxTimePerFrame Property

float **MaxTimePerFrame** (read-only)

Description

Returns the maximum time between two consecutive frames.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.MinTimePerFrame Property

float **MinTimePerFrame** (read-only)

Description

Returns the minimum time between two consecutive frames.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential Grab Demo

SapXferFrameRateInfo.Reset Method

void **Reset**();

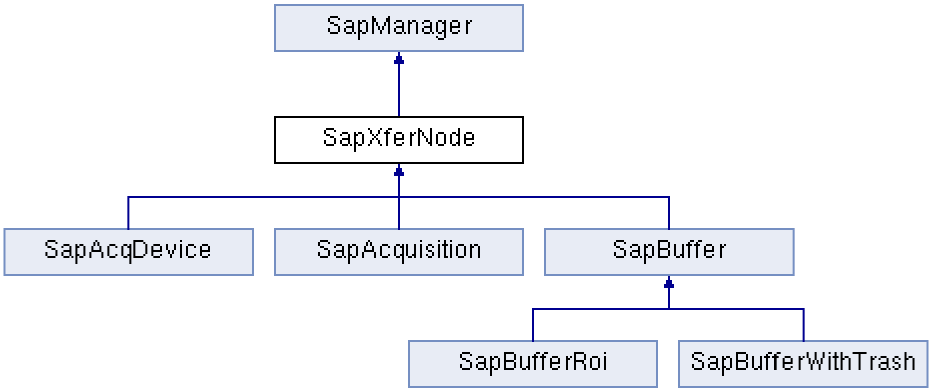
Description

Resets the frame rate calculator. This method is called automatically before starting a new transfer stream.

Demo/Example Usage

GigE MetaData Demo, GigE Sequential Grab Demo, Sequential 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.

In .NET, this class cannot be instantiated at all, since it is abstract. 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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapXferNode Class Members

Properties	
Location	Location where the transfer node resource is located
NodeType	Type of the current SapXferNode derived object
SrcNode	Source transfer node object used for compatibility of parameters with other transfer node objects
XferParams	Transfer parameters object used for compatibility of parameters with other transfer node objects

SapXferNode MemberFunctions

The following functions are members of the SapXferNode Class.

SapXferNode.Location Property

SapLocation **Location** (read-write)

Description

Location where the transfer node resource is located. You can only change the value of this property before calling the SapTransfer.Create method.

Demo/Example Usage

Not available

SapXferNode.NodeType Property

XferNodeType **NodeType** (read-only)

Description

Type of the current SapXferNode derived object, canbe one of the following values:

XferNodeType.Unknown	Unknown object type
XferNodeType.AcqDevice	Corresponds to a SapAcqDevice object
XferNodeType.Acquisition	Corresponds to a SapAcquisition object
XferNodeType.Buffer XferNodeType.BufferMulti	Corresponds to a SapBuffer object (application code should check for both values)

Demo/Example Usage

Not available

SapXferNode.SrcNode Property

SapXferNode **SrcNode** (read-only)

Description

Source transfer node object used for compatibility of parameters with other transfer node objects.

For example, when creating a SapBuffer object from an existing SapAcquisition object, the value of the SrcNode property of the former refers to the latter.

Demo/Example Usage

Not available

SapXferNode.XferParams Property

SapXferParams **XferParams** (read/write)

Description

Transfer parameters object used for compatibility of parameters with other transfer node objects.

You can only change the value of this property before calling the SapTransfer.Create method.

Demo/Example Usage

Not available

SapXferNotifyEventArgs

The SapXferNotifyEventArgs class contains the arguments to the application handler method for the SapTransfer.XferNotify, SapTransfer.XferTrashNotify, SapXferPair.XferNotify, and SapXferPair.XferTrashNotify events.

Namespace: DALSA.SaperaLT.SapClassBasic

SapXferNotifyEventArgs Class Members

Properties

AuxTimeStamp	Gets the auxiliary timestamp associated with transfer events
Context	Application context associated with transfer events
CustomData	Gets the data associated with a custom transfer event
CustomSize	Gets the size of the custom data returned by GetCustomData
EventCount	Current count of transfer events
EventType	Transfer events that triggered the invocation of the application event handler
GenericParam0	Gets generic parameters supported by some events
GenericParam1	
GenericParam2	
GenericParam3	
HostTimestamp	Gets the host timestamp associated with transfer events.
PairIndex	Index of the transfer pair associated with the current transfer event
Trash	Checks if the current transfer event is associated with a trash buffer

SapXferNotifyEventArgs Member Properties

The following properties are members of the SapXferNotifyEventArgs Class.

SapXferNotifyEventArgs.AuxTimeStamp Property

long **AuxTimeStamp** (read-only)

Description

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

SapXferNotifyEventArgs.Context Property

System.Object **Context** (read-only)

Description

Application context associated with transfer events. See the following properties for more details: SapTransfer.XferNotifyContext, SapTransfer.XferTrashNotifyContext, SapXferPair.XferNotifyContext, and SapXferPair.XferTrashNotifyContext.

Demo/Example Usage

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Auto-White Balance Example, GigE Camera Demo, GigE Camera LUT, GigE FlatField Demo, GigE Sequential Grab Demo, Grab CameraLink Example, Grab Console Example, Grab Demo, Grab LUT Example, Sequential Grab Demo

SapXferNotifyEventArgs.EventCount Property

int **EventCount** (read-only)

Description

Current count of transfer events. The initial value is 1 and increments every time the event handler method is invoked. 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.EventCountSource property 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

Color Conversion Demo, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo,

SapXferNotifyEventArgs.EventType Property

SapXferPair.XferEventType **EventType** (read-only)

Description

Combination of transfer events that triggered the invocation of the application event handler. Since it is possible for multiple events to trigger one such invocation, this property may actually return a combination of many events, using a bitwise OR operator. See the SapXferPair.EventType property for the list of possible values.

Note that, when the event type is SapXferPair.XferEventType.EndOfLine or SapXferPair.XferEventType.EndOfNLines, the line number for which the transfer event is invoked is not returned through this property, the corresponding bits are always set to 0.

Demo/Example Usage

Not available

SapXferNotifyEventArgs.GenericParamValue0 SapXferNotifyEventArgs.GenericParamValue1 SapXferNotifyEventArgs.GenericParamValue2 SapXferNotifyEventArgs.GenericParamValue3 Properties

int **GenericParamValue0**
int **GenericParamValue1**
int **GenericParamValue2**
int **GenericParamValue3** (read-only)

Description

Any of the four generic properties supported by some events. See the acquisition device User's Manual for a list of events using generic properties.

Demo/Example Usage

Not available

SapXferNotifyEventArgs.HostTimeStamp Property

long **HostTimeStamp** (read-only)

Description

Host CPU timestamp corresponding to the moment when the event occurred on the host. 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

SapXferNotifyEventArgs.PairIndex Property

int **PairIndex** (read-only)

Description

Index of the transfer pair associated with the current transfer event. Use this index together with the SapTransfer.Pairs property to access the corresponding SapXferPair object.

Demo/Example Usage

Not available

SapXferNotifyEventArgs.Trash Property

bool **Trash** (read-only)

Description

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, Multi-Board Sync Grab Demo, FlatField Demo, GigE Camera Demo, GigE Camera LUT, GigE FlatField Demo, Grab Demo,

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapXferPair Class Members

Construction	
SapXferPair	Class constructor
Dispose	Frees unmanaged memory resources
Properties	
CounterStampTimeBase	Base units used for counter stamp of destination buffers
Cycle	Buffer cycling mode when the destination node is a SapBuffer object
DestNode	Destination node for this pair
EventCountSource	Location at which the count of transfer events increases
EventType	Registered transfer event types
Flip	Flipping (that is, mirroring) mode for transferred images
FramesOnBoard	Number of internal buffers to be used on a source acquisition node
FramesPerCallback	Number of transferred frames that trigger a notification from the acquisition device to user level code
SrcIndex	Source node resource index for this pair
SrcNode	Source node for this pair
XferNotifyContext	Application callback context for transfer events
XferTrashNotifyContext	Application callback context for trash buffer transfer events
Events	
XferNotify	Notification of transfer events override for this pair
XferTrashNotify	Notification of trash buffer transfer events override for this pair

SapXferPair Member Functions

The following are members of the SapXferPair Class.

SapXferPair.SapXferPair (constructor)

SapXferPair()
SapXferPair(SapXferNode *srcNode*, SapXferNode *destNode*);
SapXferPair(SapXferNode *srcNode*, int *srcIndex*, SapXferNode *destNode*);

Parameters

srcNode	Source node for this pair
destNode	Destination node for this pair
srcIndex	Source node resource index 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.

The *srcIndex* argument applies only to the case where the source node is a SapBuffer object, where it identifies the source buffer resource index. In all other cases, *srcIndex* is ignored.

By default, application event handlers for the XferNotify and XferTrashNotify events of the associated SapTransfer object are used for regular and trash buffer events, respectively. You may define event handlers for the SapXferPair.XferNotify and SapXferPair.XferTrashNotify events if you need to override these event handlers for the current SapXferPair object.

By default, regular and trash buffer callback event handlers are invoked 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 changing the value of the EventType property. You cannot change the event type for trash buffers, however.

Demo/Example Usage

Not available

SapXferPair.CounterStampTimeBase Property

XferCounterStampTimeBase **CounterStampTimeBase** (read/write)

Description

Base units used for counter stamps of destination buffers for the current pair, can be one of the following values:

XferCounterStampTimeBase.MicroSecond	Microseconds
XferCounterStampTimeBase.MilliSecond	Milliseconds
XferCounterStampTimeBase.Line	Line valid or horizontal sync signal
XferCounterStampTimeBase.LineTrigger	External line trigger of shaft encoder pulse
XferCounterStampTimeBase.Frame	Frame valid or vertical sync signal
XferCounterStampTimeBase.ExtFrameTrigger	External frame trigger signal
XferCounterStampTimeBase.CounterStampShaftEncoder	Shaft encoder input (before drop or/and multiply factors)

Description

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.CounterStampInfo property to find out which common reference point is used for the current transfer pair.

The initial value for this property is MicroSecond.

Depending on the current transfer device, you may be allowed to change the value of this property at any time. However, you should still do this before calling SapTransfer.Create or before SapTransfer.Connect if you use SapTransfer.AutoConnect to turn off the auto-connect mechanism.

Note, for frame grabbers that support the acquisition timestamp (see SapAcquisition.TimeStampAvailable Property), the acquisition timestamp is used; the timestamp base is set using the SapAcquisition.TimeStampBase Property.Note also that this property is not available for GigE Vision cameras. Use the SapAcqDevice.GetFeatureValue and SapAcqDevice.SetFeatureValue methods with the ‘TimestampCounter’ feature.

Demo/Example Usage

Sequential Grab Demo

SapXferPair.Cycle Property

CycleMode **Cycle** (read/write)

Description

Buffer cycling mode when the destination node is a SapBuffer object, , can be one of the following values:

CycleMode.Unknown	Unknown cycle mode.
CycleMode.Asynchronous	Always transfer to the next buffer, regardless of its state.
CycleMode.Synchronous	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.
CycleMode.WithTrash	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.
CycleMode.Off	Always transfer to the current buffer.
CycleMode.NextEmpty	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.
CycleMode.NextWithTrash	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.

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 and NextWithTrash modes only. Its state is always considered to be empty by the transfer device.

The initial value for this attribute is Unknown. This means that the associated SapTransfer Class uses a WithTrash cycle mode for a SapBufferWithTrash object; otherwise, it uses Asynchronous. Change the value of this property if you want to override this initial value for the current transfer pair.

Depending on the current transfer device, you may be allowed to change the value of this property at any time. However, you should still do this before calling SapTransfer.Create or SapTransfer.Connect if you use SapTransfer.AutoConnect 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.DestNode Property

SapXferNode **DestNode** (read-only)

Description

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.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapXferPair .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties in the current SapXferPair object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Not available

SapXferPair.EventCountSource Property

XferEventCountSource **EventCountSource** (read/write)

Description

Location at which the count of transfer events increases, which can be one of the following values:

XferEventCountSource.CountNone	No event count available
XferEventCountSource.CountDest	Count is linked to the destination node
XferEventCountSource.CountSrc	Count is linked to the source node

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 CountDest.

Depending on the current transfer device, you may be allowed to change the value of this property at any time. However, you should still do this before calling SapTransfer.Create or SapTransfer.Connect if you use the SapTransfer.AutoConnect Property to turn off the auto-connect mechanism.

Note that this functionality is relevant for frame grabbers only. For GigE-Vision cameras, the behavior always corresponds to CountDest.

Demo/Example Usage

Not available

SapXferPair.EventType Property

XferEventType **EventType** (read/write)

Description

Combination of registered transfer event types for which the XferNotify or XferTrashNotify events will occur. One or more of the following values may be combined together using a bitwise OR operation:

XferEventType.None	No events
XferEventType.StartOfField	Start of field (odd or even)
XferEventType.StartOfOdd	Start of odd field
XferEventType.StartOfEven	Start of even field
XferEventType.StartOfFrame	Start of frame
XferEventType.EndOfField	End of field (odd or even)
XferEventType.EndOfOdd	End of odd field
XferEventType.EndOfEven	End of even field
XferEventType.EndOfFrame	End of frame
XferEventType.EndOfLine	After a specific line number is tranferred to the host. When used, the event type must be ORed with an unsigned integer (max 65535) representing the line number after which the callback function has to be called: *eventType* = EndOfLine | *lineNum* Note that *lineNum* only applies to SetEventType, its value is not returned when calling GetEventType, the corresponding bits are set to 0.
XferEventType.EndOfNLines	After a specific number of lines (linescan cameras only) is transferred to the host. When used, the event type must be ORed with an unsigned integer (max 65535) representing the number of lines after which the callback function has to be called: *eventType* = EndOfNLines | *numLines* Note that *numLines* only applies to SetEventType, its value is not returned when calling GetEventType, the corresponding bits are set to 0.
XferEventType.EndOfTransfer	End of transfer, that is, after all frames have been transferred following calls to SapTransfer.Snap or SapTransfer.Grab/SapTransfer.Freeze.
XferEventType.LineUnderrun	The number of active pixels per line received from a video source is less than it should be.
XferEventType.FieldUnderrun	The number of active lines per field received from a video source is less than it should be.

The initial value for this property is EndOfFrame.

You can only change the value of this property before calling SapTransfer.Create or SapTransfer.Connect if you use the SapTransfer.AutoConnect Property to turn off the auto-connect mechanism.

Demo/Example Usage

Not available

SapXferPair.Flip Property

FlipMode **Flip** (read/write)

Description

Flipping (that is, mirroring) mode for transferred images for the current transfer pair, which can be one of the following values:

FlipMode.None	No flipping
FlipMode.Horizontal	Transferred images are flipped horizontally
FlipMode.Vertical	Transferred images are flipped vertically

The initial value for this property is None.

Depending on the current transfer device, you may be allowed to change the value of this property at any time. However, you should still do this before calling SapTransfer.Create or SapTransfer.Connect if you use the SapTransfer.AutoConnect Property to turn off the auto-connect mechanism.

Demo/Example Usage

Not available

SapXferPair.FramesOnBoard Property

int **FramesOnBoard** (read/write)

Description

Number of internal buffers to be used on a source acquisition node.

The value returned by this property is only valid after calling the SapTransfer.Create method (or SapTransfer.Connect is you use the SapTransfer.AutoConnect property 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 change the value of this property. If you do, however, you should always use the following sequence:

pair.FramesOnBoard = *numFrames*;
xfer.Create();
newNumFrames = pair.FramesOnBoard

If the returned value 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.FramesPerCallback Property

int **FramesPerCallback** (read/write)

Description

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, set FramesPerCallback to 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 event handler function will still be invoked for every acquired frame.

The default value for this property is 1.

You can only call this method before calling SapTransfer.Create or SapTransfer.Connect if you use SapTransfer.AutoConnect property to turn off the auto-connect mechanism.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo

SapXferPair.SrcIndex Property

int **SrcIndex** (read-only)

Description

Source node resource index for this pair. This applies only when the node is a SapBuffer object.

Demo/Example Usage

Not available

SapXferPair.SrcNode

SapXferNode **SrcNode** (read-only)

Description

Source 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.XferNotify Event

SapXferNotifyHandler **XferNotify**

Description

Notifies the application of transfer events, overriding (for this pair only) the XferNotify event handler defined in the associated SapTransfer class.

For each transfer pair (SapXferPair) which belongs to the transfer object, use the EventType property to set the events that the application needs to be notified of. Use the XferNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapXferPair_XferNotify(Object sender, SapXferNotifyEventArgs args)
```

If a SapBufferWithTrash object is used as a destination node for a transfer pair, the event handler for the XferNotify event is also used for trash buffers, unless you override it using an event handler for the XferTrashNotify event.

Demo/Example Usage

Not available

SapXferPair.XferNotifyContext Property

System.Object **XferNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the XferNotify event is invoked. This can be any object instance derived from the System.Object base type. See the XferNotify event description for more details.

Demo/Example Usage

Not available

SapXferPair.XferTrashNotify Event

SapXferNotifyHandler **XferTrashNotify**;

Description

Notifies the application of trash buffer transfer events, thus overriding the XferNotify event, and also overriding (for this pair only) the XferTrashNotify event handler defined in the associated SapTransfer class.Only end of frame events (SapXferPair.XferEventType.EndOfFrame) are available for trash buffers

Use the XferTrashNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

```
static void SapXferPair_XferTrashNotify(Object sender, SapXferNotifyEventArgs args)
```

Demo/Example Usage

Not available

SapXferPair.XferTrashNotifyContext Property

System.Object **XferTrashNotifyContext** (read/write)

Description

Supplies application specific data when the application event handler for the XferTrashNotify event is invoked. This can be any object instance derived from the System.Object base type. See the XferTrashNotify event description 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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapXferParams Class Members

Construction	
Dispose	Frees unmanaged memory resources
Properties	
FieldOrder	Field order for interlaced frames
Format	Data format of the transferred data
FrameType	Field interlacing type in a frame
Height	Height of one frame
PixelDepth	Number of significant bits of the transferred data
Width	Width of one frame

SapXferParams Member Functions

The following are members of the SapXferParams Class.

SapXferParams.Dispose Method

void **Dispose**();

Remarks

Frees unmanaged memory used internally by a SapXferParams .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties in the current SapXferParams object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

Demo/Example Usage

Not available

SapXferParams.FieldOrder Property

XferFieldOrder **FieldOrder** (read/write)

Description

Field order for interlaced frames, which can be one of the following values:

XferFieldOrder.OddEven	The odd field is transferred before the even field
XferFieldOrder.EvenOdd	The even field is transferred before the odd field
XferFieldOrder.Next	The next field is transferred, whether it is odd or even

Remarks

Note that this property does not apply for progressive video.

Demo/Example Usage

Not available

SapXferParams.Format Property

SapFormat **Format** (read/write)

Description

Pixel format of the transferred data. See the SapBuffer constructor for possible values.

Demo/Example Usage

Multi-Board Sync Grab Demo

SapXferParams.FrameType Property

XferFrameType **FrameType** (read/write)

Description

Field interlacing type in a frame, which can be one of the following values:

XferFrameType.Interlaced	Video fields are interlaced
XferFrameType.Progressive	Video fields are non-interlaced (progressive video)

Demo/Example Usage

Not available

SapXferParams.Height Property

int **Height** (read/write)

Description

Height (in lines) of one frame

Demo/Example Usage

Multi-Board Sync Grab Demo

SapXferParams.PixelDepth Property

int **PixelDepth** (read/write)

Description

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 and SapManager.GetPixelDepthMax methods.

Demo/Example Usage

Multi-Board Sync Grab Demo

SapXferParams.Width Property

int **Width** (read/write)

Description

Width (in pixels) of one frame

Demo/Example Usage

Multi-Board Sync Grab 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 GigE Vision 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 Method and SapAcqDevice.SetFeatureValue Method . To get more information on a feature, retrieve the SapFeature object for this specific feature using the SapAcqDevice.GetFeatureInfo Method. 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.SelectedFeatureCount Property, SapFeature.SelectedFeatureNames Property, SapFeature.SelectedFeatureIndexes Property 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.FileCount Property and SapAcqDevice.FileNames Property, which allow for the enumeration of the available device files. The SapAcqDevice.WriteFile Method and SapAcqDevice.ReadFile Method 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.UpdateMode Property: only the *UpdateFeatureAuto* mode is implemented. Therefore, the SapAcqDevice.UpdateFeaturesFromDevice Method and SapAcqDevice.UpdateFeaturesToDevice Method functions are not implemented.
- SapFeature.PollingTime Property: GenICam does not provide polling information to the user, therefore this property always returns 0.
- SapFeature.SavedToConfigFile Property: The SapFeature class provides properties 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.SavedToConfigFile Property 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	SapFeature::TypeInt64
IFloat	SapFeature::TypeDouble
IString	SapFeature::TypeString
IEnummeration	SapFeature::TypeEnum
ICommand	SapFeature::TypeBool (write only)
IBoolean	SapFeature::TypeBool
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.DataType Property . 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.

GigE Vision in Sapera LT

The Sapera LT GigE Vision 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 GigE Vision 1.2 functionality: action command, extended status code, primary application switchover, pending ack, and event data.

Acquisition

GigE Vision 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 Method and SapTransfer.Snap Method control the following features: "AcquisitionMode", "AcquisitionFrameCount" and "AcquisitionStart". The SapTransfer.Freeze Method controls the "AcquisitionStop". The SapTransfer.Abort Method 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 GigE Vision 1.2 specification: unconditional streaming, multiple streams and non-streaming devices.

Cycling

When the first packet of a GigE Vision 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 GigE Vision block depends on the cycling mode; the cycling mode is set using the SapXferPair.Cycle Property 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 GigE Vision are: *SapXferPair::EventEndOfFrame* and *SapXferPair::EventEndOfTransfer*.

The *SapXferPair::EventEndOfFrame* event informs the user when all data of a GigE Vision 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 Method 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 Method 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.CounterStamp Property 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.CounterStampTimeBase Property and SapXferPair.EventCountSource Property 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.SpaceUsed Propertyto 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.

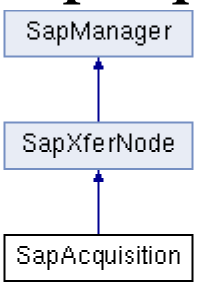
Pixel Format
The Sapera LT supports the following GigE Vision pixel formats:

Mono8 Mono8Signed Mono10 Mono12 Mono14 Mono16	BayerGR8 BayerGR10 BayerGR12 BayerGR16	BayerRG8 BayerRG10 BayerRG12 BayerRG16	BayerGB8 BayerGB10 BayerGB12 BayerGB16
BayerBG8 BayerBG10 BayerBG12 BayerBG16	BGR8Packed BGRA8Packed BGR12Packed BGR10Packed	BayerBG8 BayerBG10 BayerBG12 BayerBG16	YUV422Packed YUV411Packed YUV422_YUYV_Packed

Appendix B: Obsolete Classes & Functions

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.

SapAcquisition (Obsolete Functions)



SapAcquisition Class Obsolete Functions

Properties	
BayerAvailable	Availability of hardware-based Bayer conversion

SapAcquisition.BayerAvailable Property

bool **BayerAvailable** (read-only)

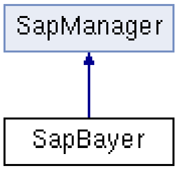
Description

Availability of hardware-based Bayer conversion. You can only read this property after calling the Create method.

Notes

Replaced by SapAcquisition.ColorConversionAvailable Property.

SapBayer (Obsolete)



The purpose of the SapBayer Class is to support conversion of Bayer encoded images. In the first case, images are acquired from a Bayer 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

Namespace: DALSA.SaperaLT.SapClassBasic

SapBayer Class Obsolete Members

Construction	
SapBayer	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Dispose	Frees unmanaged memory resources
Properties	
Acquisition	Acquisition object for acquiring Bayer images
Align	Bayer alignment mode
AvailAlign	Available alignment modes
AvailMethod	Available pixel value calculation methods
BayerBuffer	Buffer object used as the destination for software conversion
BayerBufferCount	Number of buffer resources used for software conversion
Buffer	Buffer object in which images are acquired or loaded
Enabled	Checks if Bayer conversion is enabled
Gamma	Gamma correction factor for the Bayer lookup table
IsAcqLut	Checks if the Bayer lookup table corresponds to the acquisition LUT
Lut	Current Bayer lookup table
LutEnable	Bayer lookup table enable value
Method	Bayer pixel value calculation method
OutputFormat	Data output format of Bayer conversion
SoftwareConversion	Checks if Bayer conversion is performed in software or using the hardware
WBGain	Bayer white balance gain coefficients
WBOffset	Bayer white balance offset coefficients
Methods	
Convert	Converts a Bayer-encoded image to an RGB image using software
Enable	Enables/disables Bayer conversion
WhiteBalance	Calculates the white balance gain coefficients for Bayer conversion

SapBayer Obsolete Member Functions

The following are members of the obsolete SapBayer Class.

SapBayer.SapBayer (constructor)

SapBayer();
SapBayer(SapAcquisition *acquisition,* SapBuffer *buffer*);
SapBayer(SapBuffer *buffer*);

Parameters

acquisition	SapAcquisition object to use for image acquisition and Bayer conversion (if available in hardware)
buffer	SapBuffer object in which images will be acquired or loaded

Remarks

The SapBayer 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 *buffer*. When using software conversion, the buffer object for the result of the conversion is automatically created using relevant attributes from *buffer*.

In both cases, the resulting SapBuffer object will be available through the BayerBuffer property.

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer.Acquisition Property

SapAcquisition **Acquisition** (read/write)

Description

SapAcquisition object to be used for image acquisition and for Bayer conversion. You can only set the value of this property before calling the Create method.

Demo/Example Usage

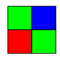
Not available

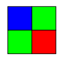
SapBayer.Align Property

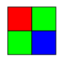
SapBayer.AlignMode **Align** (read/write)

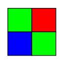
Description

Bayer alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of the camera. This mode may be one of the following values

AlignMode.GBRG 

AlignMode.BGGR 

AlignMode.RGGB 

AlignMode.GRBG 

The initial value for this property is GRBG. It is then set to the acquisition Bayer alignment value when calling the Create method (except when no acquisition is used).

Demo/Example Usage

GigE Auto-White Balance Example

SapBayer.AvailAlign Property

SapBayer.AlignMode **AvailAlign** (read-only)

Description

Available Bayer alignment modes, combined together using bitwise OR.

The initial value for this property includes all available modes. It is then set to the valid acquisition alignment modes when calling the Create method (except when no acquisition is used).

See the Align property for a list of possible modes.

Demo/Example Usage

Not available

SapBayer.AvailMethod Property

SapBayer.CalculationMethod **AvailMethod** (read-only)

Description

Available Bayer pixel value calculation methods, combined together using bitwise OR.

The initial value for this property includes all available methods. It is then set to the valid acquisition calculation methods when calling the Create method (except when no acquisition is used).

See the Method property for a list of possible methods.

Demo/Example Usage

Not available

SapBayer.BayerBuffer Property

SapBuffer **BayerBuffer** (read-only)

Description

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 Bayer conversion is performed in hardware, this method returns the same buffer object as the Buffer property.

You cannot read this property before calling the Create method.

Demo/Example Usage

Bayer Demo

SapBayer.BayerBufferCount

int **BayerBufferCount** (read/write)

Description

Number of buffer resources used for software conversion. The initial value for this property is 2.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Not available

SapBayer.Buffer Property

SapBuffer **Buffer** (read/write)

Description

SapBuffer object in which images are be acquired or loaded.

For software conversion, the buffer format must be either SapFormat.Mono8 or SapFormat.Mono16. The buffer object with the result of the conversion is then available by reading the value of the BayerBuffer property.

For hardware conversion, the buffer format may be SapFormat.RGB888, SapFormat.RGB8888, or SapFormat.RGB101010 (16-bit input image only). In this case, the buffer object returned by this property is the same as the one returned by reading the value of the BayerBuffer property.

You can only change the value of this property before calling the Create method.

Demo/Example Usage

Bayer Demo

SapBayer.Convert Method

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

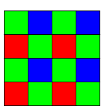
Converts a Bayer-encoded image to an RGB image using software.

The source buffer for the conversion is the current buffer resource in the main buffer object, unless you specify a source index. The Buffer property allows you to access this buffer.

The destination buffer for the conversion is the current buffer resource in the internal Bayer buffer object, unless you specify a destination index. The BayerBuffer property allows you to access this buffer.

The Bayer 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 Bayer 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

Bayer Demo

SapBayer.Create Method

bool **Create**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Creates all the internal resources needed by the Bayer conversion object.

If the Bayer object is associated with a SapAcquisition object (using the SapBayer constructor or the Acquisition property), 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 SapBayer constructor or the Buffer property).

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer.Destroy Method

BOOL **Destroy**();

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Destroys all the internal resources needed by the Bayer conversion object

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer.Dispose Method

void **Dispose();**

Remarks

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer.Enable Method

bool **Enable**(bool *enable*, bool *useHardware*);

Parameters

enable	Set to **true** to enable Bayer conversion, **false** to disable it
useHardware	Set to **true** to use hardware conversion, **false** to use the software implementation

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Enables/disables conversion of Bayer 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.BayerAvailable property to find out if hardware correction is available in the acquisition device.

Demo/Example Usage

Bayer Demo

SapBayer.Enabled Property

bool **Enabled** (read-only)

Description

Checks if Bayer conversion is enabled. The initial value for this property depends on the acquisition device.

Use the Enable method if you need to enable or disable Bayer conversion.

Demo/Example Usage

Bayer Demo

SapBayer.Gamma Property

float **Gamma** (read/write)

Description

Bayer gamma correction factor. If Bayer conversion is enabled, and the Bayer lookup table is also enabled (using the LutEnableLut property), then Gamma correction with the specified factor is applied after Bayer conversion has been performed.

The initial value for this attribute is 1.0, which effectively disables Gamma correction.

Demo/Example Usage

Not available

SapBayer.IsAcqLut Property

bool **IsAcqLut** (read-only)

Description

Checks if the Bayer lookup table corresponds to the acquisition LUT. If the value of this property is **false**, then a software lookup table is used instead.

The initial value for this property is **false**. It is then set according to the current acquisition lookup table availability when calling the Create method.

Demo/Example Usage

Not available

SapBayer.Lut Property

SapLut **Lut** (read-only)

Description

Current Bayer lookup table that is applied to image data after Bayer conversion has been performed, if the lookup table has been enabled using the LutEnable property.

For hardware conversion, this is actually the acquisition lookup table, which you may also obtain through the SapAcquisition.Luts property. If the acquisition hardware has no lookup table, then the value of this property is null.

For software conversion, the lookup table is created automatically inside the SapBayer object so that it is compatible with the buffer object on which Bayer conversion is performed.

Demo/Example Usage

Not available

SapBayer.LutEnable Property

bool **LutEnable** (read/write)

Description

Enables or disables the Bayer lookup table that is applied to image data after Bayer 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 SapBayer object so that it is compatible with the buffer object on which Bayer conversion is performed.

Demo/Example Usage

Not available

SapBayer.Method Property

SapBayer.CalculationMethod **Method** (read/write)

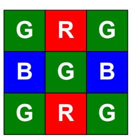
Description

Bayer pixel value calculation methodwhich may be one of the following values:

CalculationMethod. Method1	Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image. Based on a 3x3 neighborhood operation.
CalculationMethod. 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.
CalculationMethod. 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.
CalculationMethod. 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.
CalculationMethod. Method5	Technique based on a set of linear filters. It assumes that edges have a much stronger luminance than chrominance component.

The initial value for this property is Method1. It is then set to the acquisition Bayer method when calling the Create method (except when no acquisition is used).

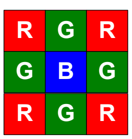
For CalculationMethod.Method1, four cases are possible according to window position:



R = (R[up] + R[down]) / 2;

G = G

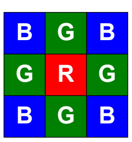
B = (B[left] + B[right]) / 2



R = (R[left,up] + R[right,up] + R[left,down] + R[right,down]) / 4

G = (G[left] + G[right] + G[up] + G[down]) / 4

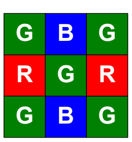
B = B



R = R

G = (G[left] + G[right] + G[up] + G[down]) / 4

B = (B[left,up] + B[right,up] + B[left,down] + B[right,down]) / 4



R = (R[left] + R[right]) / 2;

G = G

B = (B[up] + B[down]) / 2

Demo/Example Usage

Not available

SapBayer.OutputFormat Property

SapFormat **OutputFormat** (read/write)

Description

Data output format of Bayer conversion. The only two possible values for this attribute are SapFormat.RGB8888 and SapFormat.RGB101010.

The initial value for this property is SapFormat.Unknown. It is then set to the appropriate value when calling the Create method, or through this property.

You can only change the value of this property before calling the Create method

Demo/Example Usage

Bayer Demo

SapBayer.SoftwareConversion Property

bool **SoftwareConversion** (read-only)

Description

Checks if Bayer conversion is performed in software or using the hardware

The value of this property is **true** if Bayer conversion is not available in the acquisition, or if software conversion has been explicitly chosen by calling the Enable method.

The value of this property is **false** if Bayer conversion is available in the acquisition, and software conversion has not been explicitly chosen by calling the Enable method.

Demo/Example Usage

Bayer Demo

SapBayer.WBGain Property

SapDataFRGB **GetWBGain** (read/write)

Description

Bayer 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

Bayer Demo

SapBayer.WBOffset Property

SapDataFRGB **WBOffset** (read/write)

Description

Bayer white balance offset coefficients. These apply only for hardware conversion, that is, when the value of the SoftwareConversion property is **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

Bayer Demo

SapBayer.WhiteBalance Method

bool **WhiteBalance**(int *x*, int *y*, int *width*, int *height*);
bool **WhiteBalance**(SapBuffer *buffer*, 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
buffer	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 Bayer conversion. The region of interest of a Bayer-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:

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

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

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

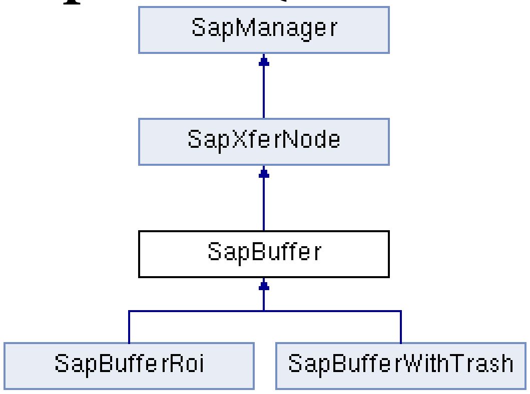
where $\overline{R}$, $\overline{G}$ and $\overline{B}$ are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormat.Mono8 or SapFormat.Mono16. 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 *buffer* argument..

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBuffer (Obsolete Function Prototypes)



The SapBuffer constructor contains obsolete prototypes; the function itself is not obsolete.

Namespace: DALSA.SaperaLT.SapClassBasic

SapBuffer Class Members

Construction

SapBuffer	Class constructor (obsolete prototypes)

SapBuffer.SapBuffer (constructor: obsolete prototypes)

SapBuffer(
   int *count*,
   SapDisplay *display*,
   int *width*,
   int *height*,
   SapFormat *format*,
   SapBuffer.MemoryType *type*
);

SapBuffer(
   int *count*,
   SapDisplay *display*,
   SapXferNode *srcNode*,
   SapBuffer.MemoryType *type*
);

Parameters

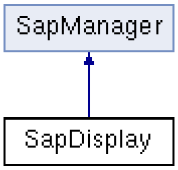
MemoryType. Offscreen [Obsolete]	Buffers are allocated in system memory. SapView objects created using these buffers may use display adapter hardware to copy from the buffer to video memory. System memory offscreen buffers may be created using any pixel format, but calling the SapView.Show method will take longer to execute if the display hardware does not efficiently support its pixel format.
MemoryType. OffscreenVideo [Obsolete]	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.

MemoryType. Overlay [Obsolete]	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.

Remarks

For the SapBuffer constructor, the above prototypes are obsolete, as well as the associated parameter type values. The SapBuffer constructor function itself is not obsolete.

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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapDisplay Class Obsolete Members

Properties	
FormatDetection	Automatic detection of available offscreen and overlay buffer formats
Methods	
IsOffscreenAvailable	Checks if offscreen display support of a specific buffer format is available
IsOverlayAvailable	Checks if overlay display support of a specific buffer format is available

SapDisplay Obsolete Member Functions

The following are members of the SapDisplay Class.

SapDisplay.FormatDetection Property

bool **FormatDetection** (read/write)

Description

Automatic detection of available offscreen and overlay buffer formats.If the value of this property 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 change the value of this property before calling the Create method. The initial value for this property is **true**.

Demo/Example Usage

Not available

SapDisplay.IsOffscreenAvailable Method

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.

SapDisplay.IsOverlayAvailable Method

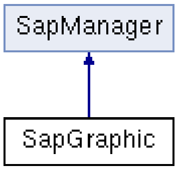
bool **IsOverlayAvailable**(SapFormat *format*);

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.

SapGraphic (Obsolete)



The SapGraphic Class implements the drawing of graphic primitives and text strings. It supports these operations either destructively on image data itself (using the low-level Sapera graphic module), or in non-destructive overlay over displayed images (using Windows GDI functions).

If you need more advanced graphic capabilities in non-destructive overlay, you will have to use Windows Forms directly instead. You will also have to call the SapView.GetGraphics Method and SapView.ReleaseGraphics Method to first obtain a valid System.Drawing.Graphics object, and then release it when you are done.
Namespace: DALSA.SaperaLT.SapClassBasic

Supported Buffer Formats For Drawing

When drawing directly on image data, the following buffer formats are supported:

Supported Format	Corresponding Buffer Format
Unsigned 8 bits/pixel	SapFormat.Mono8
Unsigned 16 bits/pixel	SapFormat.Mono16
Signed 8 bits/pixel	SapFormat.Int8
Signed 16 bits/pixel	SapFormat.Int16
Color, 48 bits/pixel	SapFormat.RGB161616
Color, 64 bits/pixel	SapFormat.RGB16161616

SapGraphic Obsolete Class Members

Construction	
SapGraphic	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
BackColor	Background drawing color
Color	Foreground drawing color
DrawingMode	Foreground drawing mode
Location	Location where the graphic resource is located
TextAlignment	Horizontal text alignment mode
Transparency	Transparency mode relative to the background
Methods	
Circle	Draws a circle
Clear	Draws a single dot
Dot	Clears the drawing area

Ellipse	Draws an ellipse
Flush	Updates non-destructive overlay with accumulated drawing commands
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
Line	Draws a line
Rectangle	Draws a rectangle
SetBatchMode	Allows delayed screen update of drawing commands in non-destructive overlay
Text	Draws a text string

SapGraphic Obsolete Member Functions

The following are members of the obsolete SapGraphic Class

SapGraphic.SapGraphic (constructor)

SapGraphic();

Remarks

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

Although a SapGraphic object could in theory be implemented on a location other than the System server, there is currently no such case.

SapGraphic.BackColor Property

SapData **BackColor** (read/write)

Description

Background drawing color. For a monochrome drawing surface, this is actually a SapDataMono object. For a color surface, this is a SapDataRGB object.

The initial value for this property is black (that is, monochrome or individual RGB components with all bits equal to 0).

The background color applies to text only, not to the drawing of graphic shapes.

SapGraphic.Circle Method

bool **Circle**(SapBuffer *buffer*, int *x*, int *y*, int *radius*, bool *fill*);
bool **Circle**(SapView *view*, int *x*, int *y*, int *radius*, bool *fill*);

Parameters

buffer	SapBuffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x	Horizontal coordinate of circle origin
y	Vertical coordinate of circle origin
radius	Radius of circle (in pixels)
fill	Specifies whether a filled shape should be drawn

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws a circle at (*x*, *y*) with the specified *radius*. The current foreground color and drawing mode are used.

If *fill* is **true**, the whole area covered by the circle is filled. If **false**, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class.

SapGraphic.Clear Method

bool **Clear**(SapBuffer *buffer*);
bool **Clear**(SapView *view*);

Parameters

buffer	SapBuffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Clears the drawing area using the current foreground color and drawing mode.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class..

SapGraphic.Color Property

SapData **Color** (read/write)

Description

Foreground ground drawing color. For a monochrome drawing surface, this is actually a SapDataMono object. For a color surface, this is actually a SapDataRGB object.

The initial value for this property is white (i.e., monochrome or individual RGB components with all bits equal to 1.

SapGraphic.Create Method

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 graphic object.

SapGraphic.Destroy Method

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 graphic object.

SapGraphic.Dispose Method

void **Dispose();**

Remarks

Frees unmanaged memory used internally by a SapGraphic .NET object. Because there is no simple way to find out when the .NET garbage collector actually reclaims this memory, you should use the Dispose method to explicitly control this behavior.

After this method has been called, you cannot access the properties and methods in the current SapGraphic object anymore. If you do, you will get a exception of type SapNativePointerException. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

SapGraphic.Dot Method

bool **Dot**(SapBuffer *buffer*, int *x*, int *y*);
bool **Dot**(SapView *view*, int *x*, int *y*);

Parameters

buffer	SapBuffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x	Horizontal dot coordinate
y	Vertical dot coordinate

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws a single dot at (*x*, *y*). The current foreground color and drawing mode are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class..

SapGraphic.DrawingMode Property

SapGraphic.DrawMode **DrawingMode (**read/write)

Description

Foreground drawing mode that specifies how the foreground color and the existing color on the drawing surface are combined together. The following values are allowed:

DrawMode.Replace	Use the foreground color only
DrawMode.And	Use bitwise AND between the two colors
DrawMode.Or	Use bitwise OR between the two colors
DrawMode.Xor	Use bitwise XOR between the two colors

Note that this mode applies to shape drawing methods only (not text).

The initial value for this property is Replace.

You can only change the value of this property before the Create method.

SapGraphic.Ellipse Method

bool **Ellipse**(SapBuffer *buffer*, int *x*, int *y*, int *xRadius*, int y*Radius*, bool *fill*);
bool **Ellipse**(SapView *view*, int *x*, int *y*, int *xRadius*, int y*Radius*, bool *fill*);

Parameters

buffer	SapBuffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x	Horizontal coordinate of ellipse origin
y	Vertical coordinate of ellipse origin
xRadius	Horizontal radius of ellipse (in pixels)
yRadius	Vertical radius of ellipse (in lines)
fill	Specifies whether a filled shape should be drawn

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws an ellipse at (*x*, *y*) with the specified *xRadius* and *yRadius*. The current foreground color and drawing mode are used.

If *fill* is **true**, the whole area covered by the ellipse is filled. If **false**, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class..

SapGraphic.Flush Method

bool **Flush**(SapView *view*);
bool **Flush**(SapView *view*, int *x1*, int *y1*, int *x2*, int *y2*);

Parameters

view	SapView object to use when drawing in non-destructive image overlay
x1	Horizontal coordinate of top left corner of updated area
y1	Vertical coordinate of top left corner of updated area
x2	Horizontal coordinate of bottom right corner of updated area
y2	Vertical coordinate of bottom right corner of updated area

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Updates non-destructive overlay with accumulated drawing commands. The area from (*x1*, *y1*) to (*x2*, *y2*) of the internal drawing surface is copied to the display in one operation. The contents of the drawing surface remain unaffected and may be modified again so that you may call Flush later using the newest data.

When the update area is specified as (0, 0) to (-1, -1), the whole drawing area is copied to the display. This is the default behavior. Specifying a smaller area improves performance of screen updates.

Flush is only available in batch mode and is not supported when drawing in image data.

See the SapGraphic.SetBatchMode method for more details.

SapGraphic.GetCapability Method

bool **GetCapability**(SapGraphic.Cap *capId*,out int *capValue*);

Parameters

capId	Low-level Sapera capability to read
capValue	Capability value to read back

Return Value

Returns **true** if successful, **false** otherwise

Remarks

This method allows direct read access to low-level Sapera capabilities for the graphic module.

Use the GetCapabilityType method to find out which version of GetCapability to use. For the SapGraphic class, the return value is always SapCapPrmType.Int32, so *capValue* must be an integer.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument.

To find out possible values for *capId*, first see the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORGRAPHIC_CAP_FILL becomes SapGraphic.Cap.FILL

Note that this method is rarely needed. The SapGraphic class already uses important capabilities internally for self-configuration and validation. Also, calling GetCapability has no effect when using .NET Windows Forms graphics.

SapGraphic.GetCapabilityType Method

static SapCapPrmType **GetCapabilityType**(SapGraphic.Cap *capId*);

Parameters

capId	Low-level Sapera capability for which the type is required

Return Value

The returned type is always SapCapPrmType.Int32, which means a 32-bit integer

Remarks

This method retrieves the exact data type of a low-level Sapera capability. See the GetCapability method for more information.

SapGraphic.GetParameter, SapGraphic.SetParameter Methods

bool **GetParameter**(SapGraphic.Prm *paramId*, out int *paramValue*);
bool **GetParameter**(SapGraphic.Prm *paramId*, out SapGraphic.Val *paramValue*);
bool **GetParameter**(SapGraphic.Prm *paramId*, out string *paramValue*);

bool **SetParameter**(SapGraphic.Prm *paramId*, int *paramValue*);
bool **SetParameter**(SapGraphic.Prm *paramId*, SapGraphic.Val *paramValue*);

Parameters

paramId	Low-level Sapera parameter to read or write
paramValue	Parameter value to read back or to write

Return Value

Returns **true** if successful, **false** otherwise

Remarks

These methods allow direct read/write access to low-level Sapera parameters for the graphic module.

Use the GetParameterType method to find out which version of GetParameter/SetParameter to use. If the return value is SapCapPrmType.Int32, then *paramValue* is an integer. If this value is SapCapPrmType.String, then *paramValue* is a text string (uninitialized for GetParameter.

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument.

To find out possible values for *paramId*, first see the *Sapera LT Basic Modules Reference Manual* for a description of all parameters. Then use the following example as a model for translating the definitions from this manual to their .NET equivalent

CORGRAPHIC_PRM_OPM becomes SapGraphic.Prm.OPM

You can also use the versions of GetParameter/SetParameter which take a SapGraphic.Val argument. In this case, first see the aforementioned manual for a description of all possible values. Then use the following example as a model for translating the definitions from this manual to .NET equivalent

CORGRAPHIC_VAL_OPM_AND becomes SapGraphic.Val.OPM_AND

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 SapGraphic class. Also, directly setting parameter values may interfere with the correct operation of the class.

Also note that calling GetParameter/SetParameter has no effect when using .NET Windows Forms graphics.

SapGraphic.GetParameterType Method

static SapCapPrmType **GetParameterType**(SapGraphic.Prm *paramId*);

Parameters

paramId Low-level Sapera parameter for which the type is required

Return Value

The returned type may be one of the following:

SapCapPrmType.Int32	32-bit integer
SapCapPrmType.String	Text string

Remarks

This method retrieves the exact data type of a low-level Sapera parameter. See the GetParameter method for more information.

SapGraphic.IsCapabilityAvailable Method

bool **IsCapabilityAvailable**(SapGraphic.Cap *capId*);

Parameters

capId Low-level Sapera capability to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

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

Note that this method is rarely needed. The SapGraphic 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.

SapGraphic.IsParameterAvailable Method

bool **IsParameterAvailable**(SapGraphic.Prm *prmId*);

Parameters

prmID Low-level Sapera parameter to check

Return Value

Returns **true** if the capability is supported, **false** otherwise

Remarks

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

Note that this method is rarely needed. The SapGraphic 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.

SapGraphic.Line Method

bool **Line**(SapBuffer *buffer*, int *x1*, int *y1*, int *x2*, int *y2*);
bool **Line**(SapView *view*, int *x1*, int *y1*, int *x2*, int *y2*);

Parameters

buffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x1	Starting horizontal coordinate
y1	Starting vertical coordinate
x2	Ending horizontal coordinate
y2	Ending vertical coordinate

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws a line from (*x1*, *y1*) to (*x2*, *y2*). The ending point at (*x2*, *y2*) is drawn. The current foreground color and drawing mode are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class.

SapGraphic.Location Property

SapLocation **Location (**read/write)

Description

Location where the graphic resource is located. Although a SapGraphic object could in theory be implemented on a location other than the System server, there is currently no such case.

You can only change the value of this property before calling the Create method.

SapGraphic.Rectangle Method

bool **Rectangle**(SapBuffer *buffer*, int *x1*, int *y1*, int *x2*, int *y2*, bool *fill*);
bool **Rectangle**(SapView *view*, int *x1*, int *y1*, int *x2*, int *y2*, bool *fill*);

Parameters

buffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x1	Horizontal coordinate of top left corner
y1	Vertical coordinate of top left corner
x2	Horizontal coordinate of bottom right corner
y2	Vertical coordinate of bottom right corner
fill	Specifies whether a filled shape should be drawn

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws a rectangle with corners at (*x1*, *y1*) and (*x2*, *y2*). The corner at (*x2*, *y2*) is drawn. The current foreground color and drawing mode are used.

If *fill* is **true**, the whole area covered by the rectangle is filled. If **false**, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class.

SapGraphic.SetBatchMode Method

bool **SetBatchMode**(bool *batchMode*, SapView *view*);

Parameters

batchMode	**true** to enable buffering of drawing commands, **false** to disable it
view	View object to use when drawing in non-destructive image overlay

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Allows delayed screen update of drawing commands in non-destructive overlay.

By default, drawing commands update the display as they are executed. When batch mode is active, these commands do not update the display immediately. Rather, they update an internally managed and invisible drawing area. It is then possible to update the display whenever needed by calling the Flush method.

This technique improves performance of screen updates, and may reduce flicker effects often associated with graphics.

Batch mode is only supported for the primary VGA board in the system. It is furthermore not supported when drawing in image data.

SapGraphic.Text Method

bool **Text**(SapBuffer *buffer*, int *x*, int *y*, string t*ext*);
bool **Text**(SapView *view*, int *x*, int *y*, string t*ext*);

Parameters

buffer	SapBuffer object to use when drawing in image data. The current buffer index is assumed.
view	SapView object to use when drawing in non-destructive image overlay
x	Horizontal text coordinate
y	Vertical text coordinate
text	Text string to draw

Return Value

Returns **true** if successful, **false** otherwise

Remarks

Draws a text string at (*x*, *y*). The current foreground/background colors, transparency mode, and text alignment options are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *view* has the SapBuffer::MemoryType.Overlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Formats For Drawing" table at the beginning of the section for the SapGraphic class..

SapGraphic.TextAlignment Property

SapGraphic.TextAlign **TextAlignment (**read/write)

Description

Horizontal text alignment mode which specifies where text strings are drawn relative to their starting (x, y) coordinates. The following values are allowed:

TextAlign.Left	Coordinates represent left side of text string
TextAlign.Center	Coordinates represent middle of text string
TextAlign.Right	Coordinates represent right side of text string

Note that text alignment does not apply to graphic shapes.

The initial value for this property is Left.

You can only change the value of this property before calling the Create method.

SapGraphic.Transparency Property

bool **Transparency** (read/write)

Description

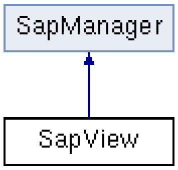
Transparency mode relative to the background. When transparency is active, the existing background content is unaffected when drawing text strings. When transparency is off, the current background drawing color is used instead.

The initial value for this property is **false**.

Transparency does not apply to the drawing of graphic shapes.

You can only change the value of this property before calling the Create method.

SapView (Obsolete Functions)



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.

Namespace: DALSA.SaperaLT.SapClassBasic

SapView Class Obsolete Members

KeyingColor	Keying color for buffers of overlay type
OverlayMode	Viewing mode when dealing with buffers of overlay type
Hide	Hides the currently displayed buffer

SapView Obsolete Member Functions

The following are obsolete members of the SapView Class.

SapView.Hide Method

void **Hide**();

Remarks

Hides the currently displayed buffer. This is only relevant when dealing with buffers of overlay type (SapBuffer.MemoryType.Overlay).

Demo/Example Usage

Not available

SapView.KeyingColor Property

SapDataRGB **KeyingColor** (read/write)

Description

Keying color when dealing with buffers of overlay type (SapBuffer.MemoryType.Overlay). See the SapDataRGB classfor a description of the related data type.

For an 8-bit display mode, that is, when the SapDisplay.PixelDepth property is equal to 8, then only the red color component is relevant.

The initial value for this property 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 change the value of this property after calling the Create method.

Demo/Example Usage

Not available

SapView.OverlayMode Property

SapView.OverlayKeyingMode **OverlayMode** (read/write)

Description

Viewing mode for buffers of overlay type (SapBuffer.MemoryType.Overlay), can be one of the following values:

OverlayKeyingMode.None	Overlay mode is not initialized yet
OverlayKeyingMode.lwaysOnTop	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.
OverlayKeyingMode.AutoKeying	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.
OverlayKeyingMode.ManualKeying	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.

The initial value for this property is None. If you do not change this value before calling the Create method, then the latter will initialize it appropriately.

Demo/Example Usage

Not available

Appendix C: Additional Buffer Information

AIA Pixel Format Naming Convention (PFNC) Equivalents

Data Formats

BICOLOR88	HSI	RGB565	RGBP16
BICOLOR1212	HSIP8	RGB888	RGBAP16
BICOLOR1616	HSV	RGB888_MONO8	Y411
COORD3D_C16	INT8	RGB8888	YUV
COORD3D_AC16	INT16	RGBP8	YUY2
COORD3D_ACRW16	INT32	RGBAP8	YUYV
COORD3D_PC_XYZ	MONO1	RGBR888	YVYU
FLOAT	MONO8	RGB101010	
FPOINT	MONO16	RGB161616	
POINT	MONO32	RGB161616_MONO16	
	RGB5551	RGB16161616	

AIA Pixel Format Naming Convention (PFNC) Equivalents

PFNC Format	Sapera Data Format	Buffer Byte Alignment (bit order is little endian)
BiColorRGBG8 BiColorBGRG8	BICOLOR88	Byte 0: R_1, Byte 1: G_1, Byte 2: B_1, Byte 3: G_1, Byte 4: R_2, Byte 5: G_3, Byte 6: B_2, Byte 7: G_4 Or Byte 0: B_1, Byte 1: G_1, Byte 2: R_1, Byte 3: G_2, Byte 4: B_2, Byte 5: G_3, Byte 6: R_2, Byte 7: G_4

BiColorRGBG12p
BiColorBGRG12p

BICOLOR1212

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
R_1 (11...4)	R_1 (3...0) G_1 (3...0)	G_1 (11...4)	B_1 (11...4)	B_1 (3...0) G_2 (3...0)	G_2 (11...4)	R_2 (11...4)	R_2 (3...0) G_3 (3...0)

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
G_3 (11...4)	B_2 (11...4)	B_2 (3...0) G_4 (3...0)	G_4 (11...4)	R_3 (11...4)	R_3 (3...0) G_5 (3...0)	G_5 (11...4)	B_3 (11...4)

Byte16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
B_3 (3...0) G_6 (3...0)	G_6 (11...4)	R_4 (11...4)	R_4 (3...0) G_7 (3...0)	G_7 (11...4)	B_4 (11...4)	B_4 (3...0) G_8 (3...0)	G_8 (11...4)

Or

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
B_1 (11...4)	B_1 (3...0) G_1 (3...0)	G_1 (11...4)	R_1 (11...4)	R_1 (3...0) G_2 (3...0)	G_2 (11...4)	B_2 (11...4)	B_2 (3...0) G_3 (3...0)

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
G_3 (11...4)	R_2 (11...4)	R_2 (3...0) G_4 (3...0)	G_4 (11...4)	B_3 (11...4)	B_3 (3...0) G_5 (3...0)	G_5 (11...4)	R_3 (11...4)

Byte16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
R_3 (3...0) G_6 (3...0)	G_6 (11...4)	B_4 (11...4)	B_4 (3...0) G_7 (3...0)	G_7 (11...4)	R_4 (11...4)	R_4 (3...0) G_8 (3...0)	G_8 (11...4)

BiColorRGBG16
BiColorBGRG16

BICOLOR1616

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
R_1	G_1	B_1	G_2

Or

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
B_1	G_1	R_1	G_2

Coord3D_C16

COORD3D_C16

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
C_1	C_2	C_3	C_4

Coord3D_AC16

COORD3D_AC16

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
A_1	C_1	A_2	C_2

Coord3D_ACRW16

COORD3D_ACRW16

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
A_1	C_1	R_1	reserved

Coord3D_ABC32f

COORD3D_PC_XYZ

Byte 0-3	Byte 4-7	Byte 8-11	Byte 12-15	Byte 16-19	Byte 20-23
X_1	Y_1	Z_1	X_2	Y_2	Z_2

ISHa8

HSI

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
I_1	S_1	H_1	A_1	I_2	S_2	H_2	A_2

HSI8_Planar

HSIP8

Page 0	Page 1	Page 2
H_1	S_1	I_1

VSHa8

HSV

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
V_1	S_1	H_1	A_1	V_2	S_2	H_2	A_2

Mono1p

MONO1

Byte 0								Byte 1							
Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_{16}	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9

Mono8

MONO8

Byte 0	Byte 1	Byte 2	Byte 3
Y_1	Y_2	Y_3	Y_4

Mono16

MONO16

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
Y_1	Y_2	Y_3	Y_4

Mono32

MONO32

Byte 0-3	Byte 4-7	Byte 8-11	Byte 12-15
Y_1	Y_2	Y_3	Y_4

BayerGR8
BayerRG8
BayerGB8
BayerBG8

MONO8

Byte 0	Byte 1	Byte 2	Byte 3
Y_1	Y_2	Y_3	Y_4

BayerGR10
BayerRG10

MONO16

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
Y_1	Y_2	Y_3	Y_4

BayerGB10 BayerBG10																	
BGRa5551	RGB5551	Bit 4:0	Bit 9:5	Bit 14:10	Bit 15	Bit 4:0	Bit 9:5	Bit 14:10	Bit 15								
		B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2								
BGR565	RGB565	Bit 4:0	Bit 10:5	Bit 15:11	Bit 20:16	Bit 26:21	Bit 31:27	Bit 4:0	Bit 10:5								
		B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3								
BGR8	RGB888	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3								
BGRa8	RGB8888	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2								
BGR10p	RGB101010	Bit 9:0	Bit 19:10	Bit 29:20	Bit 31:30	Bit 9:0	Bit 19:10	Bit 29:20	Bit 31:30								
		B_1	G_1	R_1	Not Used	B_2	G_2	R_2	Not Used								
BGR16	RGB161616	Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15								
		B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3								
BGRY16	RGB161616_MONO16	Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15								
		B_1	G_1	R_1	Y_1	B_2	G_2	R_2	Y_2								
BGRa16	RGB16161616	Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15								
		B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2								
RGB8_Planar	RGBP8	Page 0	Page 1	Page 2													
		R_1	G_1	B_1													
RGB16_Planar	RGBP16	Page 0	Page 1	Page 2													
		R_1	G_1	B_1													
RGBA8_Planar	RGBAP8	Page 0	Page 1	Page 2	Page 3												
		R_1	G_1	B_1	A_1												
RGBA16_Planar	RGBAP16	Page 0	Page 1	Page 2	Page 3												
		R_1	G_1	B_1	A_1												
RGB8	RGBR888	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		R_1	G_1	B_1	R_2	G_2	B_2	R_3	G_3								
YUV422_8_UYVY	UYVY	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		U_1	Y_1	V_1	Y_2	U_2	Y_3	V_2	Y_4								
YUVa8	YUV	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		Y_1	U_1	V_1	A_1	Y_2	U_2	V_2	A_2								
YUV422_8	YUY2	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		Y_1	U_1	Y_2	V_1	Y_3	U_2	Y_4	V_2								
YUV8_YVYU	YVYU	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		Y_1	V_1	Y_2	U_1	Y_3	V_2	Y_4	U_2								
YUV422_8	YUYV	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7								
		Y_1	U_1	Y_2	V_1	Y_3	U_2	Y_4	V_2								
YUV411_8_UYVY	Y411	Byte 0			Byte 3			Byte 6			Byte 9						
		Y_1	Y_2	U_1	Y_3	Y_4	V_1	Y_5	Y_6	U_5	Y_7	Y_8	V_5	U_2	Y_7	V_2	Y_8

Data Format Descriptions

BICOLOR88

Related Parameter Values	SapFormat.BICOLOR88
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255] (*unsigned*)
Bit Organization	The bit organization is set using SapColorConverion.SetAlign. Possible values are SapColorConversion::AlignRGBG or SapColorConversion::AlignBGRG. 1 pixel is generated for 2 components (RG or BG) therefore the buffer width is twice the size of resulting image.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
R_1	G_1	B_1	G_1	R_2	G_3	B_2	G_4

Or

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
B_1	G_1	R_1	G_2	B_2	G_3	R_2	G_4

Note	Represents an RGB color value.

BICOLOR1212

Related Parameter Values SapFormat.BICOLOR1212

Number of Components 3

Number of Bits 12 per component, 192 total

Value Range [0...255] (*unsigned*)

Bit Organization The bit organization is set using SapColorConverion.SetAlign. Possible values are SapColorConversion::AlignRGBG or SapColorConversion::AlignBGRG.
1 pixel is generated for 2 components (RG or BG) therefore the buffer width is twice the size of resulting image.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
R_1 (11...4)	R_1 (3...0) G_1 (3...0)	G_1 (11...4)	B_1 (11...4)	B_1 (3...0) G_2 (3...0)	G_2 (11...4)	R_2 (11...4)	R_2 (3...0) G_3 (3...0)
Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
G_3 (11...4)	B_2 (11...4)	B_2 (3...0) G_4 (3...0)	G_4 (11...4)	R_3 (11...4)	R_3 (3...0) G_5 (3...0)	G_5 (11...4)	B_3 (11...4)
Byte16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
B_3 (3...0) G_6 (3...0)	G_6 (11...4)	R_4 (11...4)	R_4 (3...0) G_7 (3...0)	G_7 (11...4)	B_4 (11...4)	B_4 (3...0) G_8 (3...0)	G_8 (11...4)

Or

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
B_1 (11...4)	B_1 (3...0) G_1 (3...0)	G_1 (11...4)	R_1 (11...4)	R_1 (3...0) G_2 (3...0)	G_2 (11...4)	B_2 (11...4)	B_2 (3...0) G_3 (3...0)
Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
G_3 (11...4)	R_2 (11...4)	R_2 (3...0) G_4 (3...0)	G_4 (11...4)	B_3 (11...4)	B_3 (3...0) G_5 (3...0)	G_5 (11...4)	R_3 (11...4)
Byte16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
R_3 (3...0) G_6 (3...0)	G_6 (11...4)	B_4 (11...4)	B_4 (3...0) G_7 (3...0)	G_7 (11...4)	R_4 (11...4)	R_4 (3...0) G_8 (3...0)	G_8 (11...4)

Note Represents an RGB color value.

BICOLOR1616

Related Parameter Values	SapFormat.BICOLOR1616
Number of Components	3
Number of Bits	16 per component, 64 total
Value Range	[0...65535] (*unsigned*)
Bit Organization	The bit organization is set using SapColorConverion.SetAlign. Possible values are SapColorConversion::AlignRGBG or SapColorConversion::AlignBGRG. 1 pixel is generated for 2 components (RG or BG) therefore the buffer width is twice the size of resulting image.

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
R_1	G_1	B_1	G_2

Or

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
B_1	G_1	R_1	G_2

Note Represents an RGB color value.

COORD3D_AC16

Related Parameter Values	SapFormat.Coord3D_AC16
Number of Components	2
Number of Bits	16 per component, 64 total
Value Range	[0...65535] (*unsigned*)
Bit Organization	0-15: A_1 (X) component 16-31: C_1 (Z) component 32-47: A_2 (X) component 48-63: C_2 (Z) component

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
A_1	C_1	A_2	C_2

Note Represents a pair of AC (XZ) coordinates. This data format is always *unsigned*.

COORD3D_ACRW16

Related Parameter Values	SapFormat.Coord3D_ACRW16
Number of Components	4
Number of Bits	16 per component, 64 total
Value Range	[0...65535] (*unsigned*)
Bit Organization	0-15: A_1 (X) component 16-31: C_1 (Z) component 32-47: R_1 reflectance component 48-63: reserved

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
A_1	C_1	R_1	reserved

Note Represents a pairof AC (XZ) coordinates and corresponding reflectance value. This data format is always *unsigned*.

COORD3D_C16

Related Parameter Values	SapFormat.Coord3D_C16
Number of Components	1
Number of Bits	16 per component, 64 total
Value Range	[0...65535] (*unsigned*)
Bit Organization	0-15: C_1 (Z) component 16-31: C_2 (Z) component 32-47: C_3 (Z) component 48-63: C_4 (Z) component

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
C_1	C_2	C_3	C_4

Note	Represents a C (Z) coordinate. This data format is always *unsigned*.

COORD3D_PC_XYZ

Related Parameter Values	SapFormat.Coord3D_PC_XYZ
Number of Components	3
Number of Bits	32 per component, 96 total
Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e−38
Bit Organization	0-31: X component 32-63: Y component 64-95: Z component

Byte 0-3	Byte 4-7	Byte 8-11	Byte 12-15	Byte 16-19	Byte 20-23
X_1	Y_1	Z_1	X_2	Y_2	Z_2

Note	Represents a point cloud XYZ coordinate. This data format is always *signed*.

FLOAT

Related Parameter Values	SapFormat.Float
Number of Components	1
Number of Bits	32
Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e−38
Note	Represents a single floating-point number. This data format is always *signed*.

FPOINT

Number of Components	2
Number of Bits	32 per component, 64 total
Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e−38
Bit Organization	0-31: X component 32-63: Y component
Note	Represents a pair of float. It is usually used for storing image coordinates. This data format is always *signed*.

HSI

Related Parameter Values	SapFormat.HSI
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Intensity component 8-15: Saturation component 16-23: Hue component 24-31: Alpha channel

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
I_1	S_1	H_1	A_1	I_2	S_2	H_2	A_2

Note	Represents a HSI color value.

HSIP8

Related Parameter Values	SapFormat.HSIP8
Number of Components	1
Number of Pages	3

Page 0	Page 1	Page 2
H_1	S_1	I_1

Number of Bits	8 per component
Value Range	[0...255]
Note	Represents a planar HSI color value.

HSV

Related Parameter Values	SapFormat.HSV
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Value component 8-15: Saturation component 16-23: Hue component 24-31: Alpha channel

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
V_1	S_1	H_1	A_1	V_2	S_2	H_2	A_2

Note	Represents a HSV color value.

INT8

Related Parameter Values	SapFormat.Int8
Number of Components	1
Number of Bits	8
Value Range	[-128...127]
Note	Represents a single monochrome value.

INT16

Related Parameter Values	SapFormat.Int16
Number of Components	1
Number of Bits	16 (pixel depth can range from 9 to 16)
Value Range	[-32768,32767]
Note	Represents a single monochrome value.

INT32

Related Parameter Values SapFormat.Int32

Number of Components 1

Number of Bits 32

Value Range [-2147483648...2147483647]

Note Represents a single monochrome value.

MONO1

Related Parameter Values SapFormat.Mono1

Number of Components 1

Number of Bits 1

Byte 0						
Y_1	Y_2	Y_3	....	Y_{30}	Y_{31}	Y_{32}

Value Range [0...1] (*unsigned*)

Note Represents a single monochrome value.

MONO8

Related Parameter Values SapFormat.Mono8

Number of Components 1

Number of Bits 8

Byte 0	Byte 1	Byte 2	Byte 3
Y_1	Y_2	Y_3	Y_4

Value Range [0...255] (*unsigned*)
[-128...127] (*signed*)

Note Represents a single monochrome value.

MONO16

Related Parameter Values SapFormat.Mono16

Number of Components 1

Number of Bits 16 (pixel depth can range from 9 to 16)

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7
Y_1	Y_2	Y_3	Y_4

Value Range [0...65535] (*unsigned*)
[-32768,32767] (*signed*)

Note Represents a single monochrome value.

MONO32

Related Parameter Values	SapFormat.Mono32
Number of Components	1
Number of Bits	32

Byte 0-3	Byte 4-7	Byte 8-11	Byte 12-15
Y_1	Y_2	Y_3	Y_4

Value Range	[0...4294967295] (*unsigned*) [-2147483648...2147483647] (*signed*)
Note	Represents a single monochrome value.

POINT

Related Parameter Values	SapFormat.Point
Number of Components	2
Number of Bits	32 per component, 64 total
Value Range	[-2147483648...2147483647]
Bit Organization	0-31: X component 32-63: Y component
Note	Represents a pair of integers. It is usually used for storing image coordinates. This data format is always *signed*.

RGB5551

Related Parameter Values	SapFormat.RGB5551
Number of Components	3
Number of Bits	5 per component, 16 total
Value Range	[0...31] (*unsigned*) [-16...15] (*signed*)
Bit Organization	0-4: Blue component 5-9: Green component 10-14: Red component 15: 1-bit alpha channel

Bit 4:0	Bit 9:5	Bit 14:10	Bit 15	Bit 4:0	Bit 9:5	Bit 14:10	Bit 15
B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2

Note	Represents a RGB color value.

RGB565

Related Parameter Values	SapFormat.RGB565
Number of Components	3
Number of Bits	5, 6, 5 (for red, green and blue components respectively),16 total
Value Range	Red/blue: [0...31] (*unsigned*), [-16...15] (*signed*) Green: [0...63] (*unsigned*), [-32...31] (*signed*)
Bit Organization	0-4: Blue component 5-10: Green component 11-15: Red component

Bit 4:0	Bit 10:5	Bit 15:11	Bit 20:16	Bit 26:21	Bit 31:27	Bit 4:0	Bit 10:5
B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3

Note	Represents a RGB color value.

RGB888

Related Parameter Values	SapFormat.RGB888
Number of Components	3
Number of Bits	8 per component, 24 total
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Blue component 8-15: Green component 16-23: Red component

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3

Note	Represents a RGB color value with the blue component stored first.

RGB888_MONO8

Related Parameter Values	SapFormat.RGB888_MONO8
Number of Components	4
Number of Bits	8 per component, 24 total for RGB; 8 for IR (mono)
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Blue component 8-15: Green component 16-23: Red component or 0-7: IR (mono) component

Left ¾ of line							Right ¼ of line		
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5		Byte n	Byte n+1	
B_1	G_1	R_1	B_2	G_2	R_2	...	Y_1	Y_2	...

Note	Represents an 8-bit multiformat buffer with RGB and IR (mono) components. The left ¾ of the line are the RGB components and the right ¼ of the line is the IR component.

RGB8888

Related Parameter Values	SapFormat.RGB8888
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Blue component 8-15: Green component 16-23: Red component 24-31: Alpha channel

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2

Note	Represents a RGB color value.

RGB101010

Related Parameter Values	SapFormat.RGB101010
Number of Components	3
Number of Bits	10 per component, 32 total
Value Range	[0...1023] (*unsigned*) [-512...511] (*signed*)
Bit Organization	0-9: Blue component 10-19: Green component 20-29: Red component 30-31: Not used

Bit 9:0	Bit 19:10	Bit 29:20	Bit 31:30	Bit 9:0	Bit 19:10	Bit 29:20	Bit 31:30
B_1	G_1	R_1	Not Used	B_2	G_2	R_2	Not Used

Note	Represents a RGB color value.

RGB161616

Related Parameter Values	SapFormat.RGB161616
Number of Components	3
Number of Bits	16 per component, 48 total (pixel depth can range from 9 to 16)
Value Range	[0...65535] (*unsigned*) [-32768...32767] (*signed*)
Bit Organization	0-15: Blue component 16-31: Green component 32-47: Red component

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15
B_1	G_1	R_1	B_2	G_2	R_2	B_3	G_3

Note	Represents a RGB color value.

RGB161616_MONO16

Related Parameter Values	SapFormat.RGB161616_MONO16
Number of Components	4
Number of Bits	16 per component, 64 total (pixel depth can range from 9 to 16)
Value Range	[0...65535] (*unsigned*) [-32768...32767] (*signed*)
Bit Organization	0-15: Blue component 16-31: Green component 32-47: Red component 48-63: IR (mono) component

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15
B_1	G_1	R_1	Y_1	B_2	G_2	R_2	Y_2

Note	Represents a 16-bit multiformat buffer with RGB and IR (mono) components.

RGB16161616

Related Parameter Values	SapFormat.RGB16161616
Number of Components	4
Number of Bits	16 per component, 64 total (pixel depth can range from 9 to 16)
Value Range	[0...65535] (unsigned) [-32768...32767] (signed)
Bit Organization	0-15: Blue component 16-31: Green component 32-47: Red component 48-63: Alpha component

Byte 0-1	Byte 2-3	Byte 4-5	Byte 6-7	Byte 8-9	Byte 10-11	Byte 12-13	Byte 14-15
B_1	G_1	R_1	A_1	B_2	G_2	R_2	A_2

Note	Represents a RGBA color value.

RGBP8

Related Parameter Values SapFormat.RGBP8

Number of Components 1

Number of Pages 3

Page 0	Page 1	Page 2
R_1	G_1	B_1

Number of Bits 8

Value Range [0...255]

Note Represents a planar RGB value

RGBP16

Related Parameter Values SapFormat.RGBP16

Number of Components 1

Number of Pages 3

Page 0	Page 1	Page 2
R_1	G_1	B_1

Number of Bits 16 (pixel depth can range from 9 to 16)

Value Range [0...65535]

Note Represents a planar RGB value

RGBAP8

Related Parameter Values SapFormat.RGBAP8

Number of Components 2

Number of Pages 4

Page 0	Page 1	Page 2	Page 3
R_1	G_1	B_1	A_1

Number of Bits 8

Value Range [0...255]

Note Represents a planar RGB value

RGBAP16

Related Parameter Values SapFormat.RGBAP16

Number of Components 2

Number of Pages 4

Page 0	Page 1	Page 2	Page 3
R_1	G_1	B_1	A_1

Number of Bits 16 (pixel depth can range from 9 to 16)

Value Range [0...65535]

Note Represents a planar RGBA value

RGBR888

Related Parameter Values	SapFormat.RGBR888
Number of Components	3
Number of Bits	8 per component, 24 total
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Red component 8-15: Green component 16-23: Blue component

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
R_1	G_1	B_1	R_2	G_2	B_2	R_3	G_3

Note	Represents a RGB color value with the red component stored first.

UYVY

Related Parameter Values	SapFormat.UYVY
Number of Components	3
Number of Bits	8 per component (16 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: U_0 8–15: Y_0 Second element: 0-7: V_0 8-15: Y_1

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
U_1	Y_1	V_1	Y_2	U_2	Y_3	V_2	Y_4

Note	This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YUV

Related Parameter Values	SapFormat.YUV
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Y component 8-15: U component 16-23: V component 24-31: Alpha channel

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Y_1	U_1	V_1	A_1	Y_2	U_2	V_2	A_2

Note	Represents a YUV color value.

YUY2

Related Parameter Values	SapFormat.YUY2
Number of Components	3
Number of Bits	8 per component (16 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: U_0 Second element: 0-7: Y_1 8-15: V_0

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Y_1	U_1	Y_2	V_1	Y_3	U_2	Y_4	V_2

Note	Alias for the YUYV format. This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YVYU

Related Parameter Values	SapFormat.YVYU
Number of Components	3
Number of Bits	8 per component, effectively 16 per element
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: V_0 Second element: 0-7: Y_1 8-15: U_0

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Y_1	V_1	Y_2	U_1	Y_3	V_2	Y_4	U_2

Note	This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YUYV

Related Parameter Values
SapFormat.YUYV

Number of Components
3

Number of Bits
8 per component (16 per element)

Value Range
Y: [0...255]
U: [-128...127]
V: [-128...127]

Bit Organization
First element:
0–7: Y_0
8–15: U_0
Second element:
0-7: Y_1
8-15: V_0

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Y_1	U_1	Y_2	V_1	Y_3	U_2	Y_4	V_2

Note
Alias for the YUY2 format.
This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

Y411

Related Parameter Values
SapFormat.Y411

Number of Components
3

Number of Bits
8 per component (12 bits average per pixel)

Value Range
Y: [0...255]
U: [-128...127]
V: [-128...127]

Bit Organization
First element:
0–7: Y_0
8–15: Y_1
16-23: U_0
Second element:
24–31: Y_3
0-7: Y_4
8-15: V_0

Byte 0			Byte 3			Byte 6			Byte 9						
Y_1	Y_2	U_1	Y_3	Y_4	V_1	Y_5	Y_6	U_5	Y_7	Y_8	V_5	U_2	Y_7	V_2	Y_8

Note
This is a 4:1:1 subsampled format in which for every four luminance components (Y) there is one set of color components (U, V). At least 6 consecutive bytes are needed to retrieve all the information for the individual components, for 12 bits average per pixel.

Image Data Format Conversions

The following image data format conversions are available when using the SapBuffer.Copy or SapBuffer.CopyAll functions.

Buffer Data Format	Supported Conversions
SapFormat.BICOLOR88	RGB888, RGB8888, RGB16161616, BICOLOR1616
SapFormat.BICOLOR1212	RGB888, RGBR888, RGB8888, RGB161616, RGB16161616, BICOLOR1616
SapFormat.BICOLOR1616	RGB888, RGB8888, RGB161616, RGB16161616
SapFormat.COORD3D_C16	MONO16, COORD3D_PC_XYZ
SapFormat.COORD3D_AC16	MONO16, RGB161616 (A = R, C = G), COORD3D_PC_XYZ
SapFormat.COORD3D_ACRW16	MONO16, RGB16161616 (A = R, C = G, R = B, W = alpha), COORD3D_PC_XYZ
SapFormat.Float	MONO8, MONO16
SapFormat.HSI	RGB8888
SapFormat.HSV	MONO8, RGB8888, HSV
SapFormat.Mono1	MONO8
SapFormat.Mono8	MONO16, RGB8888, YUY2, YUYV, FLOAT, MONO1
SapFormat.Mono16	MONO8, RGB8888, RGB161616, YUY2, YUYV, FLOAT
SapFormat.RGB888	RBGP8, RGB8888, UYVY, YUY2, YUYV, BICOLOR88, BICOLOR1616
SapFormat.RGB888_MONO8	MONO8, RGB888, RGB8888
SapFormat.RGB8888	MONO8, RGBP8, RGB888, HSV, UYVY, YUY2, YUYV, YUV, HSI, LAB, BICOLOR88, BICOLOR1616
SapFormat.RGB101010	MONO16, RGBP16, RGB888, RGB8888, RGB16161616
SapFormat.RGB161616	RGBP16, BICOLOR1616
SapFormat.RGB161616_MONO16	MONO16, RGB161616, RGB16161616
SapFormat.RGB16161616	RGBP16, RGB888, RGB8888, RGB101010, BICOLOR1616
SapFormat.RGBP8	RGB8888
SapFormat.RGBP16	RGB161616
SapFormat.RGBR888	RGB8888, RGBR8888, BICOLOR88, BICOLOR1616
SapFormatRGBAP8	MONO8, RGB888, RGB8888
SapFormatRGBAP16	MONO16, RGB161616, RGB16161616
SapFormat.UYVY	MONO8, RGB888, RGB8888, YUV
SapFormat.YUV	MONO8, RGB8888
SapFormat.YUY2	MONO8, RGB888, RGB8888, YUV
SapFormat.YVYU	MONO8, RGB888, RGB8888, YUV
SapFormat.Y411	MONO8, RGB888, RGB8888, YUV
SapFormat.IYU2	MONO8, RGB888, RGB8888, YUV
SapFormat.LAB	MONO8, RGB8888
SapFormat.LAB101010	MONO16, LAB
SapFormat.Mono8P2	MONO8
SapFormat.Mono8P3	MONO8
SapFormat.Mono16P2	MONO16

SapFormat.Mono16P3	MONO16
SapFormat.Mono8P4	MONO8
SapFormat.Mono16P4	MONO16

Buffer Data Formats Supported as Input by FileSave Functions

Buffer Data Format	File Format						
	BMP	**TIF**	**CRC**	**RAW**	**JPEG**	**JPEG 2000**	**AVI** uncompressed	**STL PLY VTU VTP PCD**
SapFormat.BICOLOR88	X(1)	X(1)	X	X				
SapFormat.BICOLOR1212	X(1)	X(1)	X	X				
SapFormat.BICOLOR1616	X(1)	X(1)	X	X				
SapFormat.Coord3D_C16		X(5)(8)	X	X				X
SapFormat.Coord3D_AC16		X(4)(7)(8)	X	X				X
SapFormat.Coord3D_ACRW16		X(6)(7)(8)	X	X				X
SapFormat.Coord3D_PC_XYZ			X	X				
SapFormat.Float			X	X				
SapFormat.FPoint			X	X				
SapFormat.HSI			X	X				
SapFormat.HSIP8			X	X				
SapFormat.HSV			X	X				
SapFormat.Int8	X (2)	X	X	X		X		
SapFormat.Int16	X (2)	X	X	X		X		
SapFormat.Int32			X	X				
SapFormat.Int64			X	X				
SapFormat.Mono8	X	X	X	X	X	X	X	
SapFormat.Mono8P2			X	X				
SapFormat.Mono8P3			X	X				
SapFormat.Mono8P4			X	X				
SapFormat.Mono16	X(2)	X	X	X	X(2)	X	X(2)	
SapFormat.Mono16P2			X	X				
SapFormat.Mono16P3			X	X				
SapFormat.Mono16P4			X	X				
SapFormat.Mono32			X	X				
SapFormat.Point			X	X				
SapFormat.RGB5551	X	X(2)	X	X			X	
SapFormat.RGB565	X	X(2)	X	X		X		
SapFormat.RGB888	X	X	X	X	X	X	X	
SapFormat.RGB888_MONO8			X	X				

SapFormat.RGB8888	X	X[2]	X	X	X	X	X
SapFormat.RGB101010	X	X[4]	X	X		X	
SapFormat.RGB161616	X[3]	X	X	X		X	
SapFormat.RGB161616_MONO16			X	X			
SapFormat.RGB16161616	X[3]	X[4]	X	X			
SapFormat.RGBP8	X[2]	X[2]	X	X		X	
SapFormat.RGBP16			X	X			
SapFormat.RGBAP8			X	X			
SapFormat.RGBAP16	X[3]	X[4]	X	X			
SapFormat.RGBR888	X[2]	X[2]	X	X	X[2]	X[2]	X[2]
SapFormat.UYVY	X[2]	X[2]	X	X	X[2]	X[2]	X[2]
SapFormat.YUV			X	X			
SapFormat.YUY2	X[2]	X[2]	X	X	X[2]	X[2]	X[2]
SapFormat.YVYU	X[2]	X[2]	X	X	X[2]	X[2]	X[2]
SapFormat.YUYV	X[2]	X[2]	X	X	X[2]	X[2]	X[2]
SapFormat.Y211			X	X			
SapFormat.Y411			X	X			

[1] Buffer data are converted to RGB888 format prior to being saved into file.
[2] Buffer data are converted to MONO8 (equivalent to UINT8) format prior to being saved into file.
[3] Buffer data are converted to RGB101010 format prior to being saved into file.
[4] Buffer data are converted to RGB161616 format prior to being saved into file.
[5] Buffer data are converted to MONO16 format prior to being saved into file.
[6] Buffer data are converted to RGB16161616 format prior to being saved into file.
[7] A is copied to red, C to green, R to blue, and W to alpha
[8] 3D buffers saved to TIFF files can be read back to a buffer with the original 3D format. In this case, the value of CORBUFFER_PRM_DEVICE_SCAN_TYPE and the following buffer parameters for the 3D acquisition settings (with the CORBUFFER_PRM_SCAN3D prefix) are available after loading from the file: COORD_SCALE_A, COORD_SCALE_B, COORD_SCALE_C, COORD_OFFSET_A, COORD_OFFSET_B, COORD_OFFSET_C, INVALID_DATA_FLAG_C, INVALID_DATA_VALUE_C, DISTANCE_UNIT, and OUTPUT_MODE

Contact Information

The following sections provide sales and technical support contact information.

Sales Information

Visit our web site:	www.teledynedalsa.com/corp/contact/
Email:	mailto:info@teledynedalsa.com

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	http://www.teledynedalsa.com/imaging/support
Support requests for imaging applications	
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 (for frame grabbers)
- The Device Manager BoardInfo.txt file (for frame grabbers)

	Note, the Sapera Log Viewer and PCI Diagnostic tools are available from the Windows Start menu under **Teledyne DALSA Sapera LT**. 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 under **Teledyne DALSA *Device Name*** > **Device Manager**.