Teledyne DALSA • 880 McCaffrey • St-Laurent, Quebec, H4T 2C7 • Canada
http://www.teledynedalsa.com/imaging

Sapera++ LT™ Legacy Classes Reference Manual

Part number OC-SAPM-LTLCR

Released on October 13, 2016

Document Number: OC-SAPM-LTLCR

About This Manual

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

About Teledyne DALSA

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

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

Contents

Introduction

Overview of the Manual

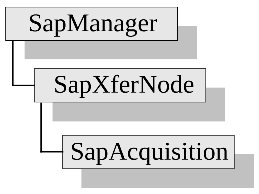
The Legacy Reference manual covers deprecated Sapera classes or functions commonly used to support older Teledyne DALSA hardware products. These Sapera classes or functions are not technically obsolete but developers can be certain that the classes described in this manual do not apply to any current Teledyne DALSA product. Future updates to this manual are dependent on when Sapera classes or functions are moved to legacy status.

Refer to the Sapera++ Programmers Reference manual for any class mentioned that is not found in this legacy class manual.

Sapera++ LT class hierarchy charts indicates which classes have been tagged as legacy classes. Again refer to the Sapera++ Programmers Reference manual for details on all current classes.

Sapera++ LT Legacy Classes

SapAcquisition



SapAcquisition Class Obsolete Functions

IsBayerAvailable	Gets availability of hardware-based Bayer conversion

SapAcquisition::IsBayerAvailable

BOOL **IsBayerAvailable**();

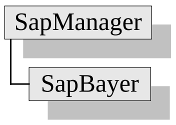
Remarks

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

Demo/Example Usage

Bayer Demo, FlatField Demo

SapBayer



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

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

SapBayer Class Members

Construction	
SapBayer	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Attributes	
GetAcquisition, SetAcquisition	Gets/sets the acquisition object for acquiring Bayer images
GetBuffer, SetBuffer	Gets/sets the buffer object in which images are acquired or loaded
GetBayerBuffer	Gets the buffer object used as the destination for software conversion
GetBayerBufferCount, SetBayerBufferCount	Gets/sets the number of buffer resources used for software conversion
IsEnabled	Checks if Bayer conversion is enabled
IsSoftware	Checks if Bayer conversion is performed in software or using the hardware
GetAlign SetAlign	Gets/sets the Bayer alignment mode
GetAvailAlign	Gets the available alignment modes
GetMethod, SetMethod	Gets/sets the pixel value calculation method
GetAvailMethod	Gets the available pixel value calculation methods
GetWBGain, SetWBGain	Gets/sets the white balance gain coefficients
GetWBOffset, SetWBOffset	Gets/sets the white balance offset coefficients
GetGamma, SetGamma	Gets/sets the gamma correction factor for the Bayer lookup table
GetOutputFormat, SetOutputFormat	Gets/sets the data output format of Bayer conversion
IsLutEnabled	Gets the current Bayer lookup table enable value
IsAcqLut	Checks if the Bayer lookup table corresponds to the acquisition LUT
Operations	

Enable	Enables/disables Bayer conversion
Convert	Converts a Bayer-encoded image to an RGB image using software
WhiteBalance	Calculates the white balance gain coefficients for Bayer conversion
GetLut	Gets the current Bayer lookup table
EnableLut	Enables/disables the Bayer lookup table

SapBayer Member Functions

The following are members of the SapBayer Class.

SapBayer::SapBayer

SapBayer();
SapBayer(SapAcquisition* *pAcq,* SapBuffer* *pBuffer*);
SapBayer(SapBuffer* *pBuffer*);

Parameters

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

In both cases, the resulting SapBuffer object will be available through the GetBayerBuffer method.

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer::Convert

BOOL **Convert**();
BOOL **Convert**(int *srcIndex*);
BOOL **Convert**(int *srcIndex*, int *dstIndex*);

Parameters

srcIndex	Source buffer resource index
dstIndex	Destination buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

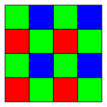
Converts a Bayer-encoded image to an RGB image using software.

The source buffer for the conversion is the current buffer resource in the main buffer object, unless you specify a source index. The GetBuffer method 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 GetBayerBuffer method 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

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 SetAcquisition method), then you can only call this method after the Create method for the acquisition object.

If there is no acquisition object, then you can only call this method after the Create method for the associated buffer object instead (specified using the SapBayer constructor or the SetBuffer method).

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

SapBayer::Destroy

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::Enable

BOOL **Enable**(BOOL *enable* = TRUE, BOOL *useHardware* = TRUE);

Parameters

enable	TRUE to enable Bayer conversion, FALSE to disable it
useHardware	TRUE to use hardware conversion, FALSE to use the software implementation

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Enables/disables conversion of 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::IsBayerAvailable method to find out if hardware correction is available in the acquisition device.

Demo/Example Usage

Bayer Demo

SapBayer::EnableLut

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

Parameters

enable	TRUE to enable the Bayer lookup table, FALSE to disable it

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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::GetAcquisition, SapBayer::SetAcquisition

SapAcquisition* **GetAcquisition**();
BOOL **SetAcquisition**(SapAcquisition* *pAcq*);

Remarks

Gets/sets the SapAcquisition object to be used for image acquisition and for Bayer conversion. You can only call SetAcquisition before the Create method.

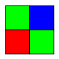
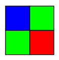
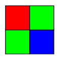
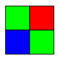
Demo/Example Usage

Not available

SapBayer::GetAlign, SapBayer::SetAlign

SapBayer::Align **GetAlign**();
BOOL **SetAlign**(SapBayer::Align *align*);

Parameters

align	Bayer alignment mode may be one of the following values	
	SapBayer::AlignGBRG	
	SapBayer::AlignBGGR	
	SapBayer::AlignRGGB	
	SapBayer::AlignGRBG	

Remarks

Gets/sets the Bayer alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of the camera.

The initial value for this attribute is SapBayer::AlignGRBG. It is then set to the acquisition device Bayer alignment value when calling the Create method (except when no acquisition device is used).

Demo/Example Usage

GigE Auto-White Balance Example

SapBayer::GetAvailAlign

SapBayer::Align **GetAvailAlign**();

Remarks

Gets the valid Bayer alignment modes, combined together using bitwise OR.

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

See the GetAlign method for a list of possible alignment modes.

Demo/Example Usage

Not available

SapBayer::GetAvailMethod

SapBayer::Method **GetAvailMethod**();

Remarks

Gets the valid Bayer pixel value calculation methods, combined together using bitwise OR.

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

See the GetMethod method for a list of possible calculation methods.

Demo/Example Usage

Not available

SapBayer::GetBayerBuffer

SapBuffer ***GetBayerBuffer**();

Remarks

Gets the buffer object used as the destination for software conversion. When using software conversion, this object is automatically created using relevant attributes from the main buffer object (the one in which images are acquired or loaded).

When Bayer conversion is performed in hardware, this method returns the same buffer object as the GetBuffer method.

You cannot call GetBayerBuffer before the Create method.

Demo/Example Usage

Bayer Demo

SapBayer::GetBayerBufferCount, SapBayer::SetBayerBufferCount

int **GetBayerBufferCount**();
BOOL **SetBayerBufferCount**(int *bayerBufferCount*);

Parameters

bayerBufferCount	Number of buffer resources

Remarks

Gets/sets the number of buffer resources used for software conversion. The initial value for this attribute is 2.

You can only call SetBayerBufferCount before the Create method.

Demo/Example Usage

Not available

SapBayer::GetBuffer, SapBayer::SetBuffer

SapBuffer ***GetBuffer**();
BOOL **SetBuffer**(SapBuffer **pBuffer*);

Remarks

Gets/sets the SapBuffer object in which images will be acquired or loaded.

For software conversion, the buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer object with the result of the conversion is then available by calling the GetBayerBuffer method.

For hardware conversion, the buffer format may be SapFormatRGB888, SapFormatRGB8888, or SapFormatRGB101010 (16-bit input image only). In this case, the buffer object returned by this method is the same as the one returned by calling the GetBayerBuffer method.

You can only call SetBuffer before the Create method.

Demo/Example Usage

Bayer Demo

SapBayer::GetGamma, SapBayer::SetGamma

float **GetGamma**();
BOOL **SetGamma**(float *gamma*);

Parameters

gamma	New gamma correction factor

Remarks

Gets/sets the Bayer gamma correction factor. If Bayer conversion is enabled, and the Bayer lookup table is also enabled (using the EnableLut method), 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::GetLut

SapLut* **GetLut**();

Remarks

Gets the 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 EnableLut method.

For hardware conversion, this is actually the acquisition lookup table, which you may also obtain through the SapAcquistion::GetLut method. If the acquisition hardware has no lookup table, then the return value is NULL.

For software conversion, the lookup table is created automatically inside the SapBayer object so that it is compatible with the buffer object on which Bayer conversion is performed.

Demo/Example Usage

Not available

SapBayer::GetMethod, SapBayer::SetMethod

SapBayer::Method **GetMethod**();
BOOL **SetMethod**(SapBayer::Method *method*);

Parameters

method Bayer pixel value calculation method may be one of the following values

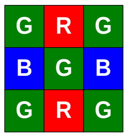
SapBayer::Method1	Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image. Based on a 3x3 neighborhood operation. See the Remarks section for more information.
SapBayer::Method2	Proprietary adaptive technique, better for preserving the edges of the image. However, it works well only when the image has a strong content in green. Otherwise, little amounts of noise may be visible within objects.
SapBayer::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.
SapBayer::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.
SapBayer::Method5	Technique based on a set of linear filters. It assumes that edges have a much stronger luminance than chrominance component.

Remarks

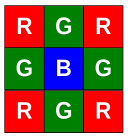
Gets/sets the Bayer pixel value calculation method.

The initial value for this attribute is SapBayer::Method1. It is then set to the acquisition device Bayer method when calling the Create method (except when no acquisition device is used).

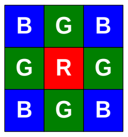
For SapBayer::Method1, four cases are possible according to window position:



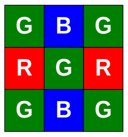
R = (R[up] + R[down]) / 2;
G = G
B = (B[left] + B[right]) / 2



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::GetOutputFormat, SapBayer::SetOutputFormat

SapFormat **GetOutputFormat**();
BOOL **SetOutputFormat** (SapFormat *format*);

Parameters

format	New Bayer conversion output format

Remarks

Gets/sets the data output format of Bayer conversion. The only two possible values for this attribute are SapFormatRGB8888 and SapFormatRGB101010.

The initial value for this attribute is SapFormatUnknown. It is then set to the appropriate value when calling the Create method.

You can only call SetOutputFormat before the Create method.

Demo/Example Usage

Bayer Demo

SapBayer::GetWBGain, SapBayer::SetWBGain

SapDataFRGB **GetWBGain**();
BOOL **SetWBGain**(SapDataFRGB *wbGain*);

Parameters

wbGain	New white balance gain coefficients

Remarks

Gets/sets the 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::GetWBOffset, SapBayer::SetWBOffset

SapDataFRGB **GetWBOffset**();
BOOL **SetWBOffset**(SapDataFRGB *wbOffset*);

Parameters

wbOffset	New white balance offset coefficients

Remarks

Gets/sets the Bayer white balance offset coefficients. These apply only for hardware conversion, that is, when the IsSoftware method returns FALSE.

The white balance offset coefficients are the red, green, and blue offsets applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all offsets to 0.0 if no white balance offset is required.

The initial value for this attribute is 0.0 for each color component.

Demo/Example Usage

Bayer Demo

SapBayer::IsAcqLut

BOOL **IsAcqLut** ();

Remarks

Checks if the Bayer lookup table corresponds to the acquisition LUT. If the return value is FALSE, then a software lookup table is used instead.

The initial value for this attribute is FALSE. It is then set according to the current acquisition device lookup table availability when calling the Create method.

Demo/Example Usage

Not available

SapBayer::IsEnabled

BOOL **IsEnabled**();

Remarks

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

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

Demo/Example Usage

Bayer Demo

SapBayer::IsLutEnabled

BOOL **IsLutEnabled**();

Remarks

Gets the current Bayer lookup table enable value. When enabled, this LUT is applied to image data after Bayer conversion has been performed.

The initial value for this attribute is FALSE. Use the EnableLut method to enable or disable the lookup table.

Demo/Example Usage

Not available

SapBayer::IsSoftware

BOOL **IsSoftware**();

Remarks

Returns TRUE if Bayer conversion is not available in the acquisition device, or if software conversion has been explicitly chosen by calling the Enable method.

Returns FALSE if Bayer conversion is available in the acquisition device, and software conversion has not been explicitly chosen by calling the Enable method.

Demo/Example Usage

Bayer Demo

SapBayer::WhiteBalance

BOOL **WhiteBalance**(int *x*, int *y*, int *width*, int *height*);
BOOL **WhiteBalance**(SapBuffer* *pBuffer*, int *x*, int *y*, int *width*, int *height*);

Parameters

x	Left coordinate of white balance region of interest
y	Top coordinate of white balance region of interest
Width	Width of white balance region of interest
Height	Height of white balance region of interest
pBuffer	Buffer object with the white balance region of interest

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Calculates the white balance gain coefficients needed for 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:

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

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

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

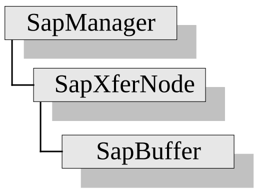
where $\overline{R}$, $\overline{G}$ and $\overline{B}$ are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer resource at the current index in the main buffer object (the one in which images are acquired or loaded) is used, unless you explicitly specify another buffer object using the *pBuffer* argument.

Demo/Example Usage

Bayer Demo, GigE Auto-White Balance Example

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, Pixel Processor, CAB, 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 or CAB resource. The array of buffers allows a transfer to cycle throughout all the buffers.

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

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

SapBuffer Class Members

Construction	
SapBuffer	Class constructor (obsolete prototypes)
Operations	
Register	Allows remote access to all buffer resources
Unregister	Removes remote access capability for all buffer resources

Member Functions

The following functions are members of the SapBuffer Class.

SapBuffer::SapBuffer

```
SapBuffer(
   int count,
   SapDisplay* pDisplay
   int width = 640,
   int height = 480,
   SapFormat format = SapFormatMono8,
   SapBuffer::Type type = SapBuffer::TypeScatterGather,
);
```

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

Parameters

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

SapBuffer:: TypeOffscreen [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.
SapBuffer:: TypeOffscreenVideo [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.
SapBuffer:: TypeOverlay [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; refer to the Sapera LT ++ Programmer's Manual for a compete description of the SapBuffer constructor.

SapBuffer::Register

virtual BOOL **Register**(const char **name*);

Parameters

name	Name to assign to the current SapBuffer object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Registers the current SapBuffer object to enable remote access. Using the supplied *name* argument, Register stores unique names for all buffer resources into an internal database. A Sapera application running on a remote server may then access these resources by creating a SapBufferRemote object with the *name* used in this method.
You may call the Unregister method after the remote server has finished accessing the buffer resources.

SapBuffer::Unregister

virtual BOOL Unregister();

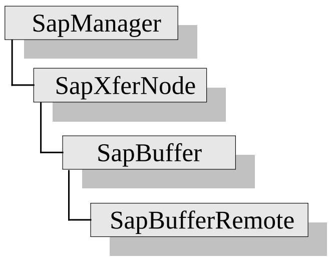
Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Disable remote access for all buffer resources in the current SapBuffer object. Unregister must be called after a Sapera application running on a remote server has finished accessing these resources. See the SapBuffer::Register method for more information.

SapBufferRemote



The SapBufferRemote Class is used to create a “wrapper” over an existing SapBuffer object on a remote server.
If a SapBuffer object exists on a remote server, and has been registered using the SapBuffer::Register method, then a SapBufferRemote object may be created locally to access the functionality of the remote object. One typical usage of this class is data exchange between two different servers.
This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassBasic.h>

SapBufferRemote Class Members

Construction	
SapBufferRemote	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources

Member Functions

The following functions are members of the SapBufferRemote Class.

SapBufferRemote::SapBufferRemote

```
SapBufferRemote(
   SapLocation loc,
   const char *name,
   int startIndex = 0,
   int count = 1
);
```

Parameters

loc	SapLocation object specifying the server where the remote buffer resources are located. The resource index of the location object is ignored.
name	The registered name of the remote object. This name must match the one used when registering the remote buffer with the SapBuffer::Register method.
startIndex	Index of the first buffer resource in the remote buffer object
count	Number of buffer resources in the remote buffer object

Remarks

The SapBufferRemote object allows creation of a “wrapper” over an existing remote buffer object. This means that the actual low-level resources are created remotely and then made available through the SapBuffer::Register method. Their values are then encapsulated in the local object. From then on, you may use the buffer object as if it had been created locally.
You may specify a subset of the remote buffer resources using the *startIndex* and *count* arguments.
The constructor does not actually access the low-level Sapera resources. To do this, you must call the Create method.
See to the SapBuffer::Register method for more details on how to register the remote buffer object.

SapBufferRemote::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created.

Remarks

Gets access to all buffer resources of the remote buffer object.

SapBufferRemote::Destroy

virtual BOOL **Destroy**();

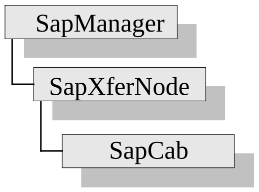
Return Value

Returns TRUE if the object was successfully destroyed.

Remarks

Release access to all the buffer resources of the remote buffer object.

SapCab



The SapCab Class includes the functionality to manipulate a CAB resource. It may be used as a destination transfer node to allow data transfer from a source node (such as acquisition buffer, Pixel Processor, or another CAB) to a CAB resource. It may also be used as a source transfer node to allow transferring data from a CAB resource to a buffer or another CAB resource.

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassBasic.h>

SapCab Class Members

Construction	
SapCab	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Operations	
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

Member Functions

The following functions are members of the SapCab Class.

SapCab::SapCab

SapCab(
  SapLocation *loc* = SapLocation::ServerSystem,
  SapXferNode **pSrcNode* = NULL
);
SapCab(
  SapLocation *loc*,
  SapXferParams *xferParams*
);

Parameters

loc	SapLocation object specifying the server where the CAB resource is located and the index of the resource on this server
pSrcNode	Source node object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
xferParams	Transfer parameters of the source node. Its attributes, to ensure transfer compatibility, must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

Remarks

The SapCab constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.
The SapCab object is used only for storing the CAB resource parameters. To transfer data to/from the CAB device, you must use the SapTransfer Class (or one of its derived classes) and specify the SapCab object as a parameter.
The data transfer is then controlled by the SapTransfer Class.

SapCab::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

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

SapCab::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

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

SapCab::GetCapability

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

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the CAB module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer. GetCapability is rarely needed. The SapCab Class already uses important capabilities internally for self-configuration and validation.
See the *CAB Programmer's Manual* for a description of all capabilities and their possible values.

SapCab::GetParameter, SapCab::SetParameter

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

Parameters

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

Return Value

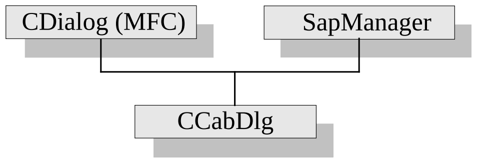
Returns TRUE if successful, FALSE otherwise

Remarks

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

Note that these methods are rarely needed. First make certain that the parameter needed is not already supported through the SapCab Class. Also, directly setting parameter values may interfere with the correct operation of the class.
See the *CAB Programmer's Manual* for a description of all parameters and their possible values.

CCabDlg



The CCabDlg Class allows you to dynamically adjust the following CAB parameters:

- Transmitter/receiver configuration
- Transfer mode
- Block size
- Source/destination channels for each port

You may create a modal version of this dialog after calling SapCab::Create, except if you rely on CCabDlg to automatically create a SapCab object.

Changes made through this dialog are saved in the current SapCab object only. Calling SapCab::Destroy causes all changes to be lost.

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassGui.h>

CCabDlg Class Members

Construction	
CCabDlg	Class constructor
Attributes	
GetCab	Gets the original or automatically creates SapCab object

Member Functions

The following functions are members of the CCabDlg Class.

CCabDlg::CCabDlg

CcabDlg(
  CWnd* *pParent*,
  SapCab **pCab*,
  SapXferNode **pSrcNode* = NULL,
  int *cabIndex* = 0
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pCab	Pointer to the related SapCab object
pSrcNode	SapXferNode object that provides CAB parameters if *pCab* is NULL
cabIndex	CAB device index to use if *pCab* is NULL

Remarks

The CCabDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
You may specify an existing SapCab object through the *pCab* argument. Alternatively, you may set *pCab* to NULL and specify an existing SapXferNode object (usually SapAcquisition) and a CAB device index. In this case, a new SapCab object is automatically created using the same SapLocation attribute as *pSrcNode* and the specified *cabIndex*.
You can only call DoModal after SapCab::Create (except of *pCab* is NULL) and before SapTransfer::Create.

Example

```
m_Xfer->Destroy();

CCabDlg dlg(this, m_Cab);
dlg.DoModal();

m_Xfer->Create();
```

CCabDlg::GetCab

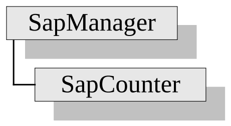
SapCab &**GetCab**();

Remarks

If you previously called the CCabDlg constructor with a non-NULL *pCab* argument, this method returns the original SapCab object.

If you called the constructor with *pCab* equal to NULL, but you specified appropriate values for the source node and the CAB device index, then this method returns a new SapCab object automatically created using the same SapLocation attribute as *pSrcNode* and the specified *cabIndex*.

SapCounter



The purpose of the SapCounter Class is to count events. These events can be external, such as a user supplied signal, or internal, such as a hardware clock The counter may then be used as a reference to control events, for example, change the state of a general I/O at a specific time (SapGio Class). It may also be used to timestamp acquired images (SapBuffer Class), or to monitor the progression of an application (by simply reading the counter value). Note that acquisition devices do not all support event counting.

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassBasic.h>

SapCounter Class Members

Construction	
SapCounter	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Gets the location where the counter resource is located
SetCallbackInfo	Sets the application callback method for counter events and the associated context
GetCallback	Gets the current application callback method for counter events
GetContext	Gets the application context associated with counter events
GetBaseUnits SetBaseUnits	Gets/sets the basic counter units
GetDetectType SetDetectType	Gets/sets the detection method of events where the counter value changes
GetDirection SetDirection	Gets/sets the counting direction (increment vs decrement)
GetCount	Gets the current counter value
GetHandle	Gets the low-level Sapera handle of the counter resource
Operations	
Start	Starts the counter device
Stop	Stops the counter device
EnableCallback	Allows an application callback function to be called at specific counter events
DisableCallback	Disables calls to the application callback function
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

Member Functions

The following functions are members of the SapCounter Class.

SapCounter::SapCounter

```
SapCounter(
   SapLocation loc = SapLocation::ServerSystem,
   SapCounterCallback pCallback = NULL,
   void *pContext = NULL
);
```

Parameters

loc	SapLocation object specifying the server where the counter resource is located and the index of the resource on this server
pCallback	Application callback function to be called each time a counter event happens. The callback function must be declared as void MyCallback(SapCounterCallbackInfo **pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Remarks

The SapCounter constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.
Specifying a callback function in the constructor does not automatically activate it after the call to the Create method.
You must subsequently call the EnableCallback method in order to be notified of counter events.

SapCounter::Create

virtual 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 counter object

SapCounter::Destroy

virtual 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 counter object

SapCounter::DisableCallback

virtual BOOL **DisableCallback**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Disables calls to the application callback function. See the SapCounter constructor and the EnableCallback method for more details.

SapCounter::EnableCallback

virtual BOOL **EnableCallback**(SapCounter::EventType *eventType*, int *elapsedTime*);

Parameters

eventType	Type of event that initiates calls to the application callback function. Only one value is currently supported:
	SapCounter::EventElapsedTime Events happen at regular time intervals
elapsedTime	Number of microseconds between each event

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Allows an application callback function to be called at specific counter events.
See the SapCounter constructor for details about the application callback function.

SapCounter::GetBaseUnits, SapCounter::SetBaseUnits

SapCounter::BaseUnits **GetBaseUnits**();
virtual BOOL **SetBaseUnits**(SapCounter::BaseUnits *baseUnits*);

Parameters

baseUnits New basic counter units that may be based on any of the following:

SapCounter::UnitsTime	An internal hardware timer
SapCounter::UnitsHSync	The horizontal sync signal from the acquisition device
SapCounter::UnitsVSync	The vertical sync signal from the acquisition device
SapCounter::UnitsExternal	A user-supplied external signal

Remarks

Gets/sets the basic counter units. The initial value for this attribute is UnitsTime.
You can only call SetBaseUnits before the Create method.

SapCounter::GetCallback

SapCounterCallback **GetCallback**();

Remarks

Gets the current application callback method for counter events. The initial value for this attribute is NULL, unless you specify another value in the constructor. See the SapCounter constructor for more details.

SapCounter::GetCapability

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

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the counter module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.
You will rarely need to use GetCapability. The SapCounter 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.

SapCounter::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with counter events. The initial value for this attribute is NULL, unless you specify another value in the constructor. See the SapCounter constructor for more details.

SapCounter::GetCount

virtual BOOL **GetCount**(int **pCount*);

Parameters

pCount	Pointer to counter value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the current counter value with units being those returned by the GetBaseUnits method.
You can only call GetCount after the Createmethod.

SapCounter::GetDetectType, SapCounter::SetDetectType

SapCounter::DetectType **GetDetectType**();
virtual BOOL **SetDetectType**(SapCounter::DetectType *detectType*);

Parameters

detectType New detection method where one of the following causes the counter value to change:

SapCounter::DetectRisingEdge	Rising edge of event triggering the counter
SapCounter::DetectFallingEdge	Falling edge of event triggering the counter

Remarks

Gets/sets the detection method of events at which the counter value changes. The initial value for this attribute is DetectRisingEdge.
You can only call SetDetectType before the Create method.

SapCounter::GetDirection, SapCounter::SetDirection

SapCounter::Direction **GetDirection**();
virtual BOOL **SetDirection**(SapCounter::Direction *direction*);

Parameters

direction New counting direction. Can be one of the following values:

SapCounter::DirectionUp	Counter increments at each triggering event
SapCounter::DirectionDown	Counter decrements at each triggering event

Remarks

Gets/sets the counting direction. The initial value for this attribute is DirectionUp.
You can only call SetDirection before the Create method.

SapCounter::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the counter resource, which you may then use from the low-level Sapera functionality.
The handle is only valid after you call the Create method.
See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapCounter::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the counter resource is located, as specified in the SapCounter constructor.

SapCounter::GetParameter, SapCounter::SetParameter

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapCounter Class. Also, directly setting parameter values may interfere with the correct operation of the class.
See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapCounter::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapCounterCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method for counter events and the associated context. You can only call SetCallbackInfo before the Create method. See the SapCounter constructor for more details.

SapCounter::Start

virtual BOOL **Start**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Starts the counter device. This is only relevant when the counter base units are not based on an internal hardware timer.

SapCounter::Stop

virtual BOOL **Stop**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Stops the counter device. This is only relevant when the counter base units are not based on an internal hardware timer.

SapCounterCallbackInfo

The SapCounterCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapCounter Class.

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassBasic.h>

SapCounterCallbackInfo Class Members

Construction	
SapCounterCallbackInfo	Class constructor
Attributes	
GetCounter	Gets the SapCounter object associated with counter events
GetEventType	Gets the counter events that triggered the call to the application callback
GetEventCount	Gets the current count of counter events
GetContext	Gets the application context associated with counter events

Member Functions

The following functions are members of the SapCounterCallbackInfo Class.

SapCounterCallbackInfo::SapCounterCallbackInfo

SapCounterCallbackInfo(
  SapCounter **pCounter*,
  void **pContext*,
  SapCounter::EventType *eventType*,
  int *eventCount*
);

Parameters

pCounter	SapCounter object that calls the callback function
pContext	Pointer to the application context
eventType	Combination of counter events. See SapCounter::EnableCallback for a list a possible values
eventCount	Current counter event count

Remarks

SapCounter objects create an instance of this class before each call to the counter callback method, in order to combine all function arguments into one container.
SapCounter uses this class for reporting of counter events. The *pContext* parameter takes the value specified in the SapCounter Class constructor, *eventType* identifies the combination of events that triggered the call to the callback function, and *eventCount* increments by one at each call, starting at 1.

SapCounterCallbackInfo::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with counter events. See the SapCounter constructor for more details.

SapCounterCallbackInfo::GetCounter

SapCounter ***GetCounter**();

Remarks

Gets the SapCounter object associated with counter events. See the SapCounter constructor for more details.

SapCounterCallbackInfo::GetEventCount

int **GetEventCount**();

Remarks

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

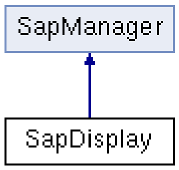
SapCounterCallbackInfo::GetEventType

SapCounter::EventType **GetEventType**();

Remarks

Gets the combination of counter events that triggered the call to the application callback.
See the SapCounter constructor for the list of possible values.

SapDisplay



The SapDisplay Class includes functionality to manipulate a display resource. There is at least one such resource for each display adapter (VGA board) in the system.

Note that SapView objects automatically manage an internal SapDisplay object for the default display resource. However, you must explicitly manage the object yourself if you need a display resource other than the default one.

#include <SapClassBasic.h>

SapDisplay Class Members

GetFormatDetection	Gets/sets automatic detection of available offscreen and overlay buffer formats
SetFormatDetection	
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 obsolete members of the SapDisplay Class.

SapDisplay::GetFormatDetection, SapDisplay::SetFormatDetection

BOOL **GetFormatDetection**();BOOL **SetFormatDetection**(BOOL *formatDetection*);

Remarks

Gets/sets automatic detection of available offscreen and overlay buffer formats.If the value of this attribute is TRUE, then all offscreen and overlay formats available for creating buffers are automatically detected when calling the Create method. It is then possible to call the IsOffscreenAvailable and IsOverlayAvaileble methods to quickly find out if creating such buffers should succeed. The drawback to this detection is that creating a SapDisplay object takes much longer, and can produce a noticeable flicker effect whenever a SapDisplay object is created explicitly by the application, or implicitly through a SapView object.While turning off auto detection solves these issues, the IsOffscreenAvailable and IsOverlayAvailable methods then become useless, and always return TRUE. In this case, trying to create a buffer of an invalid format generates an error without any possibility of prior checking.You can only call SetFormatDetection before the Create method. The initial value for this attribute is TRUE.

Demo/Example Usage

Not available

SapDisplay::IsOffscreenAvailable

BOOL **IsOffscreenAvailable**(SapFormat *format*);

Remarks

Checks if offscreen display support is available for a given buffer format. See the SapBuffer constructor for a list of possible values for *format.*

You can only call IsOffscreenAvailable after the Create method.

Demo/Example Usage

Not available

SapDisplay::IsOverlayAvailable

BOOL **IsOverlayAvailable**(SapFormat *format*);

Remarks

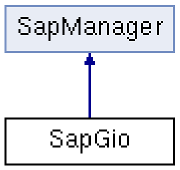
Checks if overlay display support is available for a given buffer format. See the SapBuffer constructor for a list of possible values for *format.*

You can only call IsOverlayAvailable after the Create method.

Demo/Example Usage

Not available

SapGio



The purpose of the SapGio Class is to control a block of general inputs and outputs, that is, a group of I/Os that may be read and/or written all at once. For a TTL level type I/Os, its state is considered ON or active if the measured voltage on the I/O is 5V (typical).

This class may be used together with SapCounter to associate event counting with the state of specific I/O pins.

Note that acquisition devices do not all support general I/Os.

#include <SapClassBasic.h>

SapGio Class Members

AutoTrigger	Automatically changes the state of an I/O pin for a specified duration

SapGio Obsolete Member Functions

The following are obsolete members of the SapGio Class.

SapGio::AutoTrigger

BOOL **AutoTrigger**(SapCounter* *pCounter*, int *startCount*, int *stopCount*, int *pinMask*, int *pinState*);

Parameters

pCounter	Counter object that causes I/O state transitions when reaching *startCount* and *stopCount*
startCount	Count at which the I/O pins identified by *pinMask* will change state
stopCount	Count at which the I/O pins identified by *pinMask* will go back to their original state
pinMask	Bit field specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.
pinState	Bit field representing the state of I/O pins identified by *pinMask* when the counter resource reaches *startCount*. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Bits that are set to 1 represent high, while 0 represents low.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

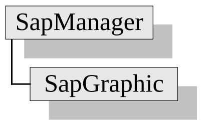
Automatically triggers the state of one or more I/O pins at a specific time.

The I/O pins identified by *pinMask* are initially set to the opposite of the values represented by *pinState*. When the counter device reaches *startCount*, their state changes to the values represented by *pinState*. When the counter device reaches *stopCount*, their state goes back to the original values. This method is not available in Sapera LT for 64-bit Windows.

Demo/Example Usage

Not available

SapGraphic



The SapGraphic Class implements the drawing of graphic primitives and text strings. It supports these operations either destructively on (using Windows GDI functions).

If you need more advanced graphic capabilities in non-destructive overlay, you will have to use GDI directly instead. You will also have to call the SapView::GetDc and SapView::ReleaseDC methods to first obtain a valid Windows display context, and then release it when you are done.

#include <SapClassBasic.h>

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	SapFormatMono8
Unsigned 16 bits/pixel	SapFormatMono16
Signed 8 bits/pixel	SapFormatInt8
Signed 16 bits/pixel	SapFormatInt16
Color, 48 bits/pixel	SapFormatRGB161616
Color, 64 bits/pixel	SapFormatRGB16161616

SapGraphic Class Members

Construction	
SapGraphic	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation, SetLocation	Gets/sets the location where the graphic resource is located
GetDrawMode SetDrawMode	Gets/sets the current foreground drawing mode
GetTransparency SetTransparency	Gets/sets the current transparency mode relative to the background
GetColor SetColor	Gets/sets the current foreground drawing color
GetBackColor SetBackColor	Gets/sets the current background drawing color
GetTextAlign SetTextAlign	Gets/sets the current horizontal text alignment mode
GetHandle	Gets the low-level Sapera handle of the graphic resource
Operations	
Clear	Clears the drawing area
Circle	Draws a circle
Ellipse	Draws an ellipse
Dot	Draws a single dot
Line	Draws a line
Rectangle	Draws a rectangle
Text	Draws a text string
SetBatchMode	Allows delayed screen update of drawing commands in non-destructive overlay
Flush	Updates non-destructive overlay with accumulated drawing commands
IsCapabilityValid	Checks for the availability of a low-level Sapera C library capability
GetCapability	Gets the value of a low-level Sapera C library capability
IsParameterValid	Checks for the availability of a low-level Sapera C library parameter
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapGraphic Obsolete Member Functions

The following are obsolete members of the SapGraphic Class

SapGraphic::SapGraphic

SapGraphic(SapLocation *loc* = SapLocation::ServerSystem);

Parameters

loc	SapLocation object specifying the server where the graphic resource is located and the index of the resource on this server

Remarks

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

Although the constructor includes an optional argument of SapLocation type, only the System server currently implements graphics support, so you should never specify another value for this argument.

SapGraphic::Circle

BOOL **Circle**(SapBuffer* *pBuffer*, int *x*, int *y*, int *radius*, BOOL *fill* = FALSE);
BOOL **Circle**(SapView* *pView*, int *x*, int *y*, int *radius*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::Clear

BOOL **Clear**(SapBuffer* *pBuffer*);
BOOL **Clear**(SapView* *pView*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::Create

BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the graphic object

SapGraphic::Destroy

BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the graphic object

SapGraphic::Dot

BOOL **Dot**(SapBuffer* *pBuffer*, int *x*, int *y*);
BOOL **Dot**(SapView* *pView*, int *x*, int *y*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::Ellipse

BOOL **Ellipse**(SapBuffer* *pBuffer*, int *x*, int *y*, int *xRadius*, int *yRadius*, BOOL *fill* = FALSE);
BOOL **Ellipse**(SapView* *pView*, int *x*, int *y*, int *xRadius*, int *yRadius*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::Flush

BOOL **Flush**(SapView* *pView*, int *x1* = 0, int *y1* = 0, int *x2* = -1, int *y2* = -1);

Parameters

pView	View 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::GetBackColor, SapGraphic::SetBackColor

SapData **GetBackColor**();
BOOL **SetBackColor**(SapData *backColor*);

Remarks

Gets/sets the current background 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 attribute is black (that is, monochrome or individual RGB components with all bits equal to 0), unless you construct this object using an existing SapGraphic object which has another background color.

The background color applies to text only, not to the drawing of graphic shapes.

SapGraphic::GetCapability

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

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Calling GetCapability has no effect when using Windows GDI graphics.

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

SapGraphic::GetColor, SapGraphic::SetColor

SapData **GetColor**();
BOOL **SetColor**(SapData *color*);

Remarks

Gets/sets the current foreground 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 attribute is white (that is, monochrome or individual RGB components with all bits equal to 1), unless you construct this object using an existing SapGraphic object which has another foreground color.

SapGraphic::GetDrawMode, SapGraphic::SetDrawMode

SapGraphic::DrawMode **GetDrawMode**();
BOOL **SetDrawMode**(SapGraphic::DrawMode *drawMode*);

Parameters

drawMode New drawing mode that specifies how the foreground color and the existing color on the drawing surface are combined together. The following values are allowed:

SapGraphic::ModeReplace	Use the foreground color only
SapGraphic::ModeAnd	Use bitwise AND between the two colors
SapGraphic::ModeOr	Use bitwise OR between the two colors
SapGraphic::ModeXor	Use bitwise XOR between the two colors

Remarks

Gets/sets the current foreground drawing mode that applies all shape drawing methods, but not text.

The initial value for this attribute is ModeReplace, unless you construct this object using an existing SapGraphic object.

You can only call SetDrawMode before the Create method.

SapGraphic::GetHandle

CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the graphic resource that you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

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

SapGraphic::GetLocation, SapGraphic::SetLocation

SapLocation **GetLocation**();
BOOL **SetLocation**(SapLocation *location*);

Remarks

Gets/sets the location where the graphic resource is located (Sapera LT currently supports the system server only). A specific server can also be specified through the SapGraphic constructor.

You can only call SetLocation before the Create method..

SapGraphic::GetParameter, SapGraphic::SetParameter

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Calling these methods has no effect when using Windows GDI graphics.

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

SapGraphic::GetTextAlign, SapGraphic::SetTextAlign

SapGraphic::TextAlign **GetTextAlign**();
BOOL **SetTextAlign**(SapGraphic::TextAlign *textAlign*);

Parameters

textAlign New text alignment mode that specifies where text strings are drawn relative to their starting (x, y) coordinates. The following values are allowed:

SapGraphic::TextLeft	Coordinates represent left side of text string
SapGraphic::TextCenter	Coordinates represent middle of text string
SapGraphic::TextRight	Coordinates represent right side of text string

Remarks

Gets/sets the current horizontal text alignment mode. This does not apply to graphic shapes.

The initial value for this attribute is TextLeft, unless you construct this object using an existing SapGraphic object.

You can only call SetDrawMode before the Create method.

SapGraphic::GetTransparency, SapGraphic::SetTransparency

BOOL **GetTransparency**();
BOOL **SetTransparency**(BOOL *isTransparent*);

Remarks

Gets/sets the current 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 attribute is FALSE.

Transparency does not apply to the drawing of graphic shapes.

You can only call SetTransparency before the Create method.

SapGraphic::IsCapabilityValid

BOOL **IsCapabilityValid**(int *cap*);

Parameters

cap	Low-level Sapera C library capability to check

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library capability for the 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::IsParameterValid

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

Parameters

param	Low-level Sapera C library parameter to check

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library parameter for the 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

BOOL **Line**(SapBuffer* *pBuffer*, int *x1*, int *y1*, int *x2*, int *y2*);
BOOL **Line**(SapView* *pView*, int *x1*, int *y1*, int *x2*, int *y2*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::Rectangle

BOOL **Rectangle**(SapBuffer* *pBuffer*, int *x1*, int *y1*, int *x2*, int *y2*, BOOL *fill* = FALSE);
BOOL **Rectangle** (SapView* *pView*, int *x1*, int *y1*, int *x2*, int *y2*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View 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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapGraphic::SetBatchMode

BOOL **SetBatchMode**(BOOL *batchMode*, SapView* *pView*);

Parameters

batchMode	TRUE to enable buffering of drawing commands, FALSE to disable it
pView	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

BOOL **Text**(SapBuffer* *pBuffer*, int *x*, int *y*, const char* *text*);
BOOL **Text**(SapView* *pView*, int *x*, int *y*, const char* *text*);

Parameters

pBuffer	Buffer object to used when drawing in image data. The current buffer index is assumed.
pView	View 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

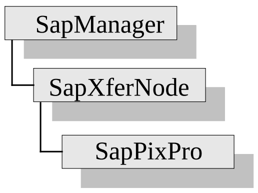
Remarks

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 *pView* has the SapBuffer::TypeOverlay type.

For a list of supported buffer formats when drawing in image data, see the "Supported Buffer Formats For Drawing" section.

SapPixPro



The SapPixPro Class includes the functionality to manipulate a pixel processor resource. It is used as an intermediate transfer node to allow data transfers from an acquisition resource to another transfer node (such as a buffer or CAB).

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassBasic.h>

SapPixPro Class Members

Construction	
SapPixPro	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetFile, SetFile	Gets/sets the name of the pixel processor file
GetParams, SetParams	Gets/sets the Pixel Processor parameters
GetOutputXferParams, SetOutputXferParams	Gets/sets the output transfer parameters
Operations	
ReadDesign	Reads a design file to allow retrieving information on functions and configurations
GetNumFunctions	Gets the number of standard functions available in the current design file
GetFunctionInfo	Gets information on a specific function in the current design file
GetNumConfigs	Gets the number of configurations available in the current design file
GetConfigInfo	Gets information on a specific configuration in the current design file
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

Member Functions

The following functions are members of the SapPixPro Class.

SapPixPro::SapPixPro

```
SapPixPro(
   SapLocation loc = SapLocation::ServerSystem,
   const char *fileName = "",
   SapXferNode *pSrcNode = NULL,
   SapPixProParams *pParams = NULL
);
SapPixPro(
   SapLocation loc,
   const char * fileName,
   SapXferParams xferParams,
   SapPixProParams *pParams = NULL
);
```

Parameters

loc	SapLocation object specifying the server where the pixel processor resource is located and the index of this resource on the server
fileName	Name of the pixel processor file
pSrcNode	Source node object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
pParams	SapPixProParams object used to initialize function and configuration parameters
xferParams	Transfer parameters of the source node. To ensure transfer compatibility, its attributes must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

Remarks

The SapPixPro constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.
Use the SapPixPro object only for loading the pixel processor file and configuring the parameters. To transfer data through the Pixel Processor, you must use the SapTransfer Class (or one of its derived classes) and specify the SapPixPro object as a parameter.
The data transfer is then controlled by the SapTransfer Class.

SapPixPro::Create

virtual 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 pixel processor object and loads the design file into the Pixel Processor.
Always call this method before SapTransfer::Create.

SapPixPro::Destroy

virtual 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 pixel processor object. Always call this method after SapTransfer::Destroy.

SapPixPro::GetCapability

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

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the Pixel Processor module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer. You will rarely need to use GetCapability. The SapPixPro 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.

SapPixPro::GetConfigInfo

BOOL **GetConfigInfo**(int *index*, char **configName*);

Parameters

index	Configuration index between 0 and the value returned by the GetNumConfigs method, minus 1
configName	Memory area large enough to receive the configuration name (at least 64 bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets information on a specific configuration in the current design file.
This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro::GetFile, SapPixPro::SetFile

const char ***GetFile**();
virtual BOOL **SetFile**(const char **fileName*);

Remarks

Gets/sets the name of the pixel processor file.
You normally set the initial value for this attribute within the SapPixPro constructor. If you do not specify a value at that time, it then defaults to an empty string. You can only call SetFile before the Create method.

SapPixPro::GetFunctionInfo

BOOL **GetFunctionInfo**(int *index*, PCORPPL_FCT_PROP *pFuncProp*, char **funcName*);

Parameters

index	Function index between 0 and the value returned by the GetNumFunctions method, minus 1
pFuncProp	Pointer to a CORPPL_FCT_PROP structure for receiving the properties of the function
funcName	Memory area large enough to receive the function name (at least 64 bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets information on a specific function in the current design file. This information is only available after a design file has been loaded into the Pixel Processor. The CORPPL_FCT_PROP structure contains an availability flag and memory bank ID. It is defined as follows:

```
typedef struct
{
  BOOLEAN fAvailable; // TRUE if function is available
  UINT32  bankId;     // Memory Bank where operation can be performed
} CORPPL_FCT_PROP, *PCORPPL_FCT_PROP;
```

SapPixPro::GetNumConfigs

int **GetNumConfigs**();

Return Value

Gets the number of configurations available in the current design file.
This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro::GetNumFunctions

int **GetNumFunctions**();

Remarks

Gets the number of standard functions available in the current design file.
This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro::GetOutputXferParams, SapPixPro::SetOutputXferParams

SapXferParams **GetOutputXferParams**();
virtual BOOL **SetOutputXferParams**(SapXferParams *params*);

Remarks

Gets/sets the output transfer parameters. Its attributes must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object. You can only call SetOutputXferParams before the Create method.

SapPixPro::GetParameter, SapPixPro::SetParameter

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapPixPro Class. Also, directly setting parameter values may interfere with the correct operation of the class.
See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapPixPro::GetParams, SapPixPro::SetParams

const SapPixProParams ***GetParams**();
virtual BOOL **SetParams**(SapPixProParams **pParams*);

Remarks

Gets/sets the pixel processor parameters.
You may set the initial value for this attribute in the SapPixPro constructor.
You can only call SetParams before the Create method.

SapPixPro::ReadDesign

BOOL **ReadDesign**(char **fileName*);

Parameters

fileName	Name of the pixel processor file

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads a design file to allow retrieving information on functions and configurations. Use the GetNumFunctions, GetFunctionInfo**,** GetNumConfigs, and GetConfigInfo methods to obtain information on functions and configurations available in the file.

SapPixProParams

SapPixProParams

The SapPixProParams Class acts as a container for storing pixel processor parameters used by the SapPixPro Class. These parameters allow you to choose a function (or configuration) in a standard design file.

This class is not available in Sapera LT for 64-bit Windows.

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

SapPixProParams Class Members

Construction	
SapPixProParams	Class constructor
Attributes	
GetFunctionIndex, SetFunctionIndex	Gets/sets the current function index in a standard design file
GetConfigIndex, SetConfigIndex	Gets/sets the current configuration index in a standard design file
GetBankId, SetBankId	Gets/sets the memory bank identifier used in the Pixel Processor
GetRefImage, SetRefImage	Gets/sets the reference image acquisition mode into the pixel processor memory bank
GetWeightingFactor, SetWeightingFactor	Gets/sets the weighting factor used by the 'Weighting Average' function
GetIntegrateCount, SetIntegrateCount	Gets/sets the integration count used by the 'Integrate and Scale' function
GetIntegrateFactor, SetIntegrateFactor	Gets/sets the integration factor used by the 'Integrate and Scale' function

Member Functions

The following functions are members of the SapPixProParams Class.

SapPixProParams::SapPixProParams

SapPixProParams(
);

Remarks

The SapPixProParams constructor initializes its members to default (but probably incorrect) values. Use the other methods in this class to properly set these values.

SapPixProParams::GetBankId, SapPixProParams::SetBankId

int **GetBankId**();
void **SetBankId**(int *bankId*);

Remarks

Gets/sets the memory bank identifier used in the Pixel Processor.

SapPixProParams::GetConfigIndex, SapPixProParams::SetConfigIndex

int **GetConfigIndex**();
void **GetConfigIndex**(int *index*);

Remarks

Gets/sets the current configuration index in a standard design file. The SapPixPro Class only uses this index if the function index is set to –1.

SapPixProParams::GetFunctionIndex, SapPixProParams::SetFunctionIndex

int **GetFunctionIndex**();
void **SetFunctionIndex**(int *index*);

Remarks

Gets/sets the current function index in a standard design file. If this index is set to –1, the SapPixPro Class uses the configuration index instead.

SapPixProParams::GetIntegrateCount, SapPixProParams::SetIntegrateCount

int **GetIntegrateCount**();
void **SetIntegrateCount**(int *count*);

Remarks

Gets/sets the integration count used by the 'Integrate and Scale' function

SapPixProParams::GetIntegrateFactor, SapPixProParams::SetIntegrateFactor

int **GetIntegrateFactor**();
void **GetIntegrateFactor**(int *factor*);

Remarks

Gets/sets the integration factor used by the 'Integrate and Scale' function

SapPixProParams::GetRefImage, SapPixProParams::SetRefImage

BOOL **GetRefImage**();
void **SetRefImage**(BOOL *isRefImage*);

Remarks

Gets/sets the reference image acquisition mode into the pixel processor memory bank

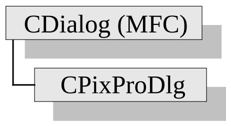
SapPixProParams::GetWeightingFactor, SapPixProParams::SetWeightingFactor

int **GetWeightingFactor**();
void **SetWeightingFactor**(int *factor*);

Remarks

Gets/sets the weighting factor used by the 'Weighting Average' function.

CPixProDlg



The CPixProDlg Class allows you to modify the parameters of a SapPixPro object.

You may create a modal version of this dialog after calling SapPixPro::Create, except if you rely on CPixProDlg to automatically create a SapPixPro object.

Depending on the pixel processor design, some additional dialog classes are automatically used to specify additional design-specific options:

- CPixProAbsSubDlg
- CPixProIntScaleDlg
- CPixProRefImageDlg
- CPixProUflowSubDlg
- CPixProWAvgDlg

This class is not available in Sapera LT for 64-bit Windows.

#include <SapClassGui.h>

CPixProDlg Class Members

Construction	
CPixProDlg	Class constructor
Attributes	
GetPixPro	Gets the original or automatically created SapPixPro object

Member Functions

The following functions are members of the CPixProDlg Class.

CPixProDlg::CPixProDlg

CPixProDlg(
  CWnd* *pParent*,
  SapPixPro **pPixPro*,
  SapXferNode **pSrcNode* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pPixPro	Pointer to the related SapPixPro object
pSrcNode	SapXferNode object that provides pixel processor parameters if *pPixPro* is NULL

Remarks

The CPixProDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
You may specify an existing SapPixPro object through the *pPixPro* argument. Alternatively, you may set *pPixPro* to NULL and specify an existing SapXferNode object (usually SapAcquisition). In this case, a new SapPixPro object is automatically created using the same SapLocation attribute as *pSrcNode*.
You can only call DoModal after SapPixPro::Create (except of *pPixPro* is NULL) and before SapTransfer::Create.

Example

```
// Both m_PixPro (SapPixPro) and m_Acq (SapAcquisition) are used.
// The first time, m_PixPro is NULL and m_Acq is used for initialization.
// The second time, m_PixPro is used for initialization.

CPixProDlg dlg(this, m_PixPro, m_Acq);
if (dlg.DoModal() == IDOK)
{
       // Destroy all objects (including m_PixPro)
       DestroyObjects();


SapPixPro pixpro;

   if (!m_PixPro)
   {
      // Allocate new object
      m_PixPro = new SapPixPro(dlg.GetPixPro());
   }
   else
   {
      // Create a backup object
      pixpro = *m_PixPro;

       // Update current object
              *m_PixPro = dlg.GetPixPro();
   }


       // Recreate all objects (including m_PixPro)
   if (!CreateObjects())
   {
      // If an error occurred, retrieve the backup copy and try again
      *m_PixPro = pixpro;
      if (!CreateObjects())
      {
          delete m_PixPro;
          m_PixPro = NULL;
          CreateObjects();
      }
   }
}
```

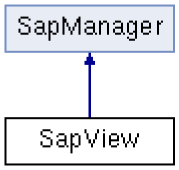
CPixProDlg::GetPixPro

SapPixPro &**GetPixPro**();

Remarks

If you previously called the CPixProDlg constructor with a non-NULL *pPixPro* argument, this method returns the original SapPixPro object.
If you called the constructor with *pPixPro* equal to NULL, but you specified an appropriate value for the source node, then this method returns a new SapPixPro object automatically created using the same SapLocation attribute as *pSrcNode*.

SapView



The SapView Class includes the functionality to display the resources of a SapBuffer object in a window. It allows you to display the current buffer resource, a specific one, or the next one not yet displayed.

An internal thread optimizes buffer display in realtime. This allows the main application thread to execute without any concerns for the display task.

An auto empty mechanism allows synchronization between SapView and SapTransfer objects to show buffers in real-time without missing any data.

#include <SapClassBasic.h>

SapView Class Members

GetOverlayMode	Gets/sets the viewing mode when dealing with buffers of overlay type
SetOverlayMode	
GetKeyColor	Gets/sets the keying color for buffers of overlay type
SetKeyColor	
Hide	Hides the currently displayed buffer

SapView Obsolete Member Functions

The following are obsolete members of the SapView Class.

SapView::GetKeyColor, SapView::SetKeyColor

SapDataRGB **GetKeyColor**();
BOOL **SetKeyColor**(SapDataRGB *keyColor*);

Remarks

Gets/sets the keying color when dealing with buffers of overlay type (SapBuffer::TypeOverlay). See the SapDataRGB class for a description of the related data type.

For an 8-bit display mode, that is, when the SapDisplay::GetPixelDepth method returns 8, then only the red color component is relevant.

The initial value for this attribute corresponds to black. When calling the Create method, if the current viewing mode is overlay, then its value will be initialized using the current low level keying color value.

You can only call SetKeyColor after the Create method.

Demo/Example Usage

Not available

SapView::GetOverlayMode, SapView::SetOverlayMode

SapView::OverlayMode **GetOverlayMode**();
BOOL **SetOverlayMode**(SapView::OverlayMode *overlayMode*);

Parameters

overlayMode Viewing mode for buffers of overlay type, can be one of the following values:

SapView::OverlayNone	Overlay mode is not initialized yet

SapView:: OverlayAlwaysOnTop	No color keying scheme is enabled. Buffer contents are displayed directly using the display adapter overlay hardware. This is the fastest method; however, other windows will not be displayed correctly if they overlap the Sapera application.
SapView:: OverlayAutoKeying	A destination color keying scheme is enabled. Source buffer pixels are displayed only if the corresponding pixel on the display has the key color. Each time a buffer is shown following calls to the Show or ShowNext methods, the current keying color is painted on the view surface. Also, the OnPaint method only repaints the keying color on the part of the view area that becomes visible again. This is usually the default mode.
SapView:: OverlayManualKeying	Similar to auto-keying mode, except that you are responsible for painting the key color in the view area. This gives you more flexibility as to where the overlay image should be displayed.

Remarks

Gets/sets the viewing mode when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

The initial value for this attribute is OverlayNone. If you do not call SetOverlayMode before the Create method, then the latter will initialize its value appropriately.

Demo/Example Usage

Not available

SapView::Hide

void **Hide**();

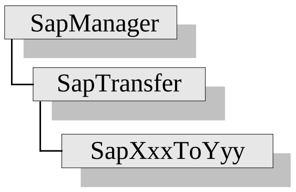
Remarks

Hides the currently displayed buffer. This is only relevant when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

Demo/Example Usage

Not available

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 SapAcqToCab Class to connect a SapAcquisition object to a SapCab object.

Each of these classes has one or more specific constructors; otherwise, they use the same methods as SapTransfer.

If you need a transfer configuration that is not supported by any of the specialized classes, then you must use SapTransfer directly instead.

#include <SapClassBasic.h>

Common Constructor Arguments

All specialized transfer classes constructors include the following two arguments:

pCallback	Application callback function to be called each time a transfer event happens. The callback function must be declared as: void MyCallback(SapXferCallbackInfo **pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

SapCabToBuf Class

SapCabToBuf(SapCab **pCab*, SapBuffer **pBuf,* SapXferCallback *pCallback* = NULL, void **pContext* = NULL);
SapCabToBuf(SapCab **pCab*, int *srcPort*, SapBuffer **pBuf,* SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pCab	Source CAB object
srcPort	Data port number on source CAB object
pBuf	Destination buffer object

Remarks

Implements a transfer from a CAB device to a buffer object. This class is not available in Sapera LT for 64-bit Windows.

SapAcqToCab Class

SapAcqToCab(SapAcquisition **pAcq*, SapCab **pCab*, SapXferCallback *pCallback* = NULL,
void **pContext* = NULL);
SapAcqToCab(SapAcquisition **pAcq*, SapCab **pCab*, int *dstPort*, SapXferCallback *pCallback* = NULL,
void **pContext* = NULL);

Parameters

pAcq	Source acquisition object
pCab	Destination CAB object
dstPort	Data port number on destination CAB object

Remarks

Implements a transfer from an acquisition device to a CAB device. This class is not available in Sapera LT for 64-bit Windows.

SapCabToCab Class

SapCabToCab(SapCab **pSrcCab*, SapCab **pDstCab*, SapXferCallback *pCallback* = NULL,
void **pContext* = NULL);

Parameters

pSrcCab	Source CAB object
pDstCab	Destination CAB object

Remarks

Implements a transfer from a CAB device to another CAB device. This class is not available in Sapera LT for 64-bit Windows.

SapBufToCab Class

SapBufToCab(SapBuffer **pBuf*, SapCab **pCab*, SapXferCallback *pCallback* = NULL,
void **pContext* = NULL);

Parameters

pBuf	Source buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from a buffer object to a CAB device. This class is not available in Sapera LT for 64-bit Windows.

SapAcqToBufThruPixPro Class

SapAcqToBufThruPixPro(SapAcquisition **pAcq*, SapPixPro **pPixPro*, SapBuffer **pBuf*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pBuf	Destination buffer object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to a buffer object.
This class is not available in Sapera LT for 64-bit Windows.

SapAcqToCabThruPixPro Class

SapAcqToCabThruPixPro(SapAcquisition **pAcq*, SapPixPro **pPixPro*, SapCab **pCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to a CAB device.
This class is not available in Sapera LT for 64-bit Windows.

SapAcqToBufCab Class

SapAcqToBufCab(SapAcquisition **pAcq*, SapBuffer **pBuf*, SapCab **pCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);
SapAcqToBufCab(SapAcquisition **pAcq*, SapBuffer **pBuf*, SapCab **pCab*, int *dstPort*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pAcq	Source acquisition object
pBuf	Destination buffer object
pCab	Destination CAB object
dstPort	Data port number on destination CAB object

Remarks

Implements a transfer from an acquisition device to both a buffer object and a CAB device (in parallel).
This class is not available in Sapera LT for 64-bit Windows.

SapCabToBufCab Class

SapCabToBufCab(SapCab **pSrcCab*, SapBuffer **pBuf*, SapCab **pDstCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);**SapCabToBufCab**(SapCab **pSrcCab*, int *srcPort*,
SapBuffer **pBuf*, SapCab **pDstCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pSrcCab	Source CAB object
srcPort	Data port number on source CAB object
pBuf	Destination buffer object
pDstCab	Destination CAB object

Remarks

Implements a transfer from a CAB device to both a buffer object and another CAB device (in parallel).
This class is not available in Sapera LT for 64-bit Windows.

SapBufToBufCab Class

SapBufToBufCab(SapBuffer **pSrcBuf*, SapBuffer **pDstBuf*, SapCab **pCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pSrcBuf	Source buffer object
pDstBuf	Destination buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from a buffer object to both another buffer object and a CAB device (in parallel).
This class is not available in Sapera LT for 64-bit Windows.

SapAcqToBufCabThruPixPro Class

SapAcqToBufCabThruPixPro(SapAcquisition **pAcq*, SapPixPro **pPixPro*, SapBuffer **pBuf*, SapCab **pCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pBuf	Destination buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to both a buffer object and a CAB device (in parallel).
This class is not available in Sapera LT for 64-bit Windows.

SapMultiAcqToCab Class

SapMultiAcqToCab(SapAcquisition **pAcq*[], SapCab **pCab*, int *numPairs*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);**SapMultiAcqToCab**(SapAcquisition **pAcq*[],
SapCab **pCab*, int *dstPort*[], int *numPairs*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pAcq	List of source acquisition objects
pCab	Destination CAB object
dstPort	List of data port numbers on destination CAB object
numPairs	Number of entries in acquisition and port lists

Remarks

Implements a transfer from a series of acquisition devices to a matching number of destination ports on a CAB device. There is a one-to-one relationship between items in the source list and items in the destination list. If no destination ports are specified, values in the range from (0, *numPairs* – 1) are automatically used. All acquisition devices must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method return TRUE. This class is not available in Sapera LT for 64-bit Windows.

SapMultiCabToBuf Class

SapMultiCabToBuf(SapCab **pCab*, SapBuffer **pBuf*[], int *numPairs*,
SapXferCallback *pCallback* = NULL,void **pContext* = NULL);**SapMultiCabToBuf**(SapCab **pCab*, int *srcPort*[],
SapBuffer **pBuf*[], int *numPairs*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pCab	Source CAB object
srcPort	List of data port numbers on source CAB object
pBuf	List of destination buffer objects
numPairs	Number of entries in port and buffer lists

Remarks

Implements a transfer from a series of source ports on a CAB device 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. If no source ports are specified, values in the range from (0, *numPairs* – 1) are automatically used. This class is not available in Sapera LT for 64-bit Windows.

SapMultiAcqToBufCab Class

SapMultiAcqToBufCab(SapAcquisition **pAcq*[], SapBuffer **pBuf*[], SapCab **pCab*, int *numPairs*,
SapXferCallback *pCallback* = NULL,void **pContext* = NULL);
SapMultiAcqToBufCab(SapAcquisition **pAcq*[], SapBuffer **pBuf*[], SapCab **pCab*, int *dstPort*[],
int *numPairs*, SapXferCallback *pCallback* = NULL,void **pContext* = NULL);

Parameters

pAcq	List of source acquisition objects
pBuf	List of destination buffer objects
pCab	Destination CAB object
dstPort	List of data port numbers on destination CAB object
numPairs	Number of entries in acquisition, buffer, and port lists

Remarks

Implements a transfer from a series of acquisition devices to a matching number of buffer objects and CAB devices (in parallel). There is a one-to-one relationship between items in the source list and items in the destination lists.

If no destination ports are specified, values in the range from (0, *numPairs* – 1) are automatically used.
All acquisition devices must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method returns TRUE. This class is not available in Sapera LT for 64-bit Windows.

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 shortcut **Start•All Programs•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 shortcut **Start•All Programs•Teledyne DALSA•<Device Name>•Device Manager**.