TELEDYNE DALSA
Everywhereyoulook™

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

Sapera++ LT™ GUI Classes Reference Manual

Part number OC-SAPM-LTGCR

Released on October 1, 2014

Document Number: OC-SAPM-LTGCR

Printed in Canada

Contents

Introduction

Overview of the Manual

The *Sapera++ LT GUI Classes Reference* Manual covers the following topics:

- Hierarchy Chart
- GUI Class Reference
- Teledyne DALSA Contact Information

Most of the GUI class references refer to other class functions described in the Sapera++ LT Programmer's Reference manual. It is recommended to have both documents open when working with the GUI classes.

About the Manual

This manual exists in Windows Help, and Adobe Acrobat (PDF) formats. The Help and PDF formats make full use of hypertext cross-references. The PDF format offers links to Teledyne DALSA's home page on the Internet, located at **www.teledynedalsa.com**.

Teledyne DALSA Web site contains documents, updates, demos, errata, utilities, and more.

Using the Manual

File names, directories, and Internet sites will be in bold text (for example, **setup.exe**, **c:\windows**, **http://www.teledynedalsa.com**). Function parameters will be in italics (for example, *xlen*).

Source code, code examples, text file listings, and text that must be entered using the keyboard will be in typewriter-style text (for example, `[timestamp]`).

Menu and dialog actions will be indicated in bold text in the order of the instructions to be executed, with each instruction separated by bullets. For example, going to the **File** menu and choosing **Save** would be written as **File•Save**.

Getting Started

What is Sapera++?

Sapera++ is the main Application Programming Interface (API) for Sapera LT. It is a C++ class library that encapsulates the low-level Sapera C Library, providing most of its functionality. It also provides some features not found in the latter, for example, Windows GDI graphics in non-destructive overlay on top of an image. Sapera++ provides high-level classes reducing application code complexity. The Sapera++ architecture reflects the underlying Sapera architecture. This provides the user with a high-level of flexibility while keeping the simplicity and compactness of object-oriented code. Sapera++ contains two sets of classes: Basic (provided as a C++ API and .NET API) and the GUI.

The ***Basic Classes*** are a set of C++ classes containing commonly used Sapera code that can be used with many imaging applications. These classes are completely user-interface independent. They address the basic concepts of imaging applications, such as Acquisition Transfers, Processing, and Display. The *Basic Classes* main purpose is to simplify application code by reducing considerably the number of calls to Sapera functions and parameters. Source code is no longer included for these classes with Sapera LT, only the header files and libraries necessary for application development are available. Refer to the Sapera++ Programmer's Reference manual and the Sapera .NET Programmer's Reference manual.

The ***GUI Classes as described in this manual***, are an additional set of C++ classes containing commonly used dialog boxes and windows used in many imaging applications. These classes are MFC™ dependent. They are intended to be included in any MFC-based imaging application in order to improve rapid-application development (RAD). GUI class source code is included with Sapera LT so that they may be freely modified. Sapera++ applications do not have to use these classes. Developers are free to implement their own dialog boxes and windows instead.

Requirements

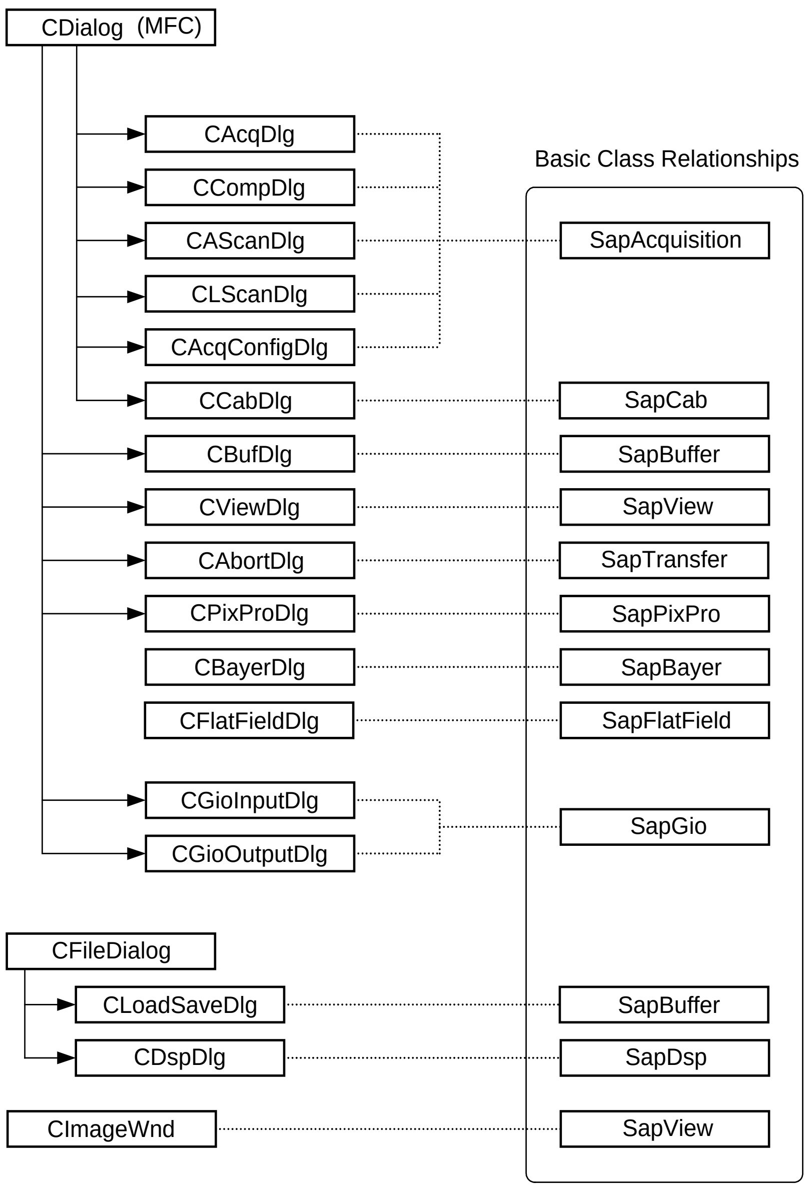
Sapera++ currently supports the following compilers:
Microsoft Visual C++ 6.0
Microsoft Visual C++ .NET 2003
Microsoft Visual C++ .2005
Borland C++ Builder 6, or C++ Builder 2006 (*Basic Classes* only)

File Locations

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

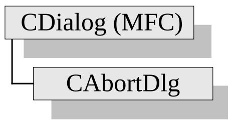
Description	Location
Basic Classes headers	**Sapera\Classes\Basic**
GUI Classes headers and source code	**Sapera\Classes\Gui**
Import librairies	**Sapera\Lib**
Import librairies (Visual C++ .NET 2003)	**Sapera\Lib\.NET_2003**
Import librairies (Visual Studio 2005)	**Sapera\Lib\.NET_2005**
Import libraries (C++ Builder 6 and C++ Builder 2006)	**Sapera\Lib\Bc**
Dynamic-link libraries (DLLs)	Windows system directory (**<windir>\System32**)

GUI Class Hierarchy Chart



GUI Class Reference

CAbortDlg



The CAbortDlg dialog class is a more evolved version of the SapTransfer::Wait method. It allows you to wait an indefinite amount of time, while giving you the possibility to abort at any time. This is useful when the time for one frame is variable (for example, when using an external trigger).

You may create a modal version of this dialog right after calling SapTransfer::Snap or SapTransfer::Freeze. If the transfer somehow does not terminate cleanly, then clicking the 'Abort' button allows you to call SapTransfer::Abort to unconditionally terminate the transfer.

#include <SapClassGui.h>

CAbortDlg Class Members

Construction	
CAbortDlg	Class constructor

Member Functions

The following functions are members of the CAbortDlg Class.

CAbortDlg::CAbortDlg

```
CAbortDlg(
   CWnd* pParent,
   SapTransfer *pXfer,
   int timeout = 1000
);
```

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pXfer	Pointer to the related SapTransfer object
timeout	Time to wait before showing the dialog (in milliseconds)

Remarks

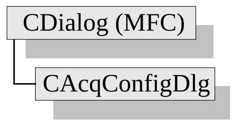
The CAbortDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
If you call DoModal, and the related transfer is inactive, then the dialog is not shown and the value IDOK is returned right away. The same happens if the transfer ends before the specified *timeout* period.

If you call DoModal, and the related transfer is still active after the *timeout* period, then the dialog appears. If the transfer ends before you click the 'Abort' button, then the dialog automatically closes with a return value equal to IDOK. If you click 'Abort', then the dialog closes with a return value equal to IDCANCEL.
There is usually nothing else to do after the dialog returns IDOK. You, however, must call SapTransfer::Wait if IDCANCEL is returned.

Example

```
m_Xfer->Snap();
if (CAbortDlg(this, m_Xfer).DoModal() != IDOK)
   m_Xfer->Abort();
```

CAcqConfigDlg



The CAcqConfigDlg Class allows you to load an acquisition configuration file (CCF) to create a new SapAcquisition object. You may then retrieve this new object using the CAcqConfigDlg:: GetAcquisition method.

Alternatively, you may specify an existing SapAcquisition object. In this case, a copy of the object is automatically created for loading the CCF file. Calling CAcqConfigDlg:: GetAcquisition then returns this object.

#include <SapClassGui.h>

CAcqConfigDlg Class Members

Construction

CAcqConfigDlg	Class constructor

Attributes

GetAcquisition	Gets the automatically created SapAcquisition object

Member Functions

The following functions are members of the CAcqConfigDlg Class.

CAcqConfigDlg::CAcqConfigDlg

CAcqConfigDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq* = NULL,
  const char **productDir* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object
productDir	Environment variable name that specifies a product directory other than Sapera for CCF files

Remarks

The CAcqConfigDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
If you specify an existing SapAcquisition object through the *pAcq* argument, then a new object is automatically created as a working copy. The existing object is not modified.

If *pAcq* is NULL, then the latest acquisition server and resource indices are retrieved from Windows Registry, as well as the name of the CCF file. A new SapAcquisition object is then automatically created from these values. If no such registry entry exists, then the default directory for finding the CCF files is set according to the environment variable you specify with the *productDir* argument (for example, “X64DIR”).
When you are finished with the dialog, you can get the new object by calling GetAcquisition.

Example 1: Creating a SapAcquisition object the first time

```
CAcqConfigDlg dlg(this, NULL);
if (dlg.DoModal() == IDOK)
{
   m_Acq = new SapAcquisition(dlg.GetAcquisition());
}
```

Example 2: Modifying an existing SapAcquisition object.

```
CAcqConfigDlg dlg(this, m_Acq);
if (dlg.DoModal() == IDOK)
{
   // Destroys all objects (including m_Acq)
   DestroyObjects();

   // Creates a backup object
   SapAcquisition acq = *m_Acq;


   // Updates acquisition object
   *m_Acq = dlg.GetAcquisition();

   // Recreates all objects (including m_Acq)
   if (!CreateObjects())
   {
      // If an error occurred, retrieves the backup copy and tries again
      *m_Acq = acq;
      CreateObjects();
   }
}
```

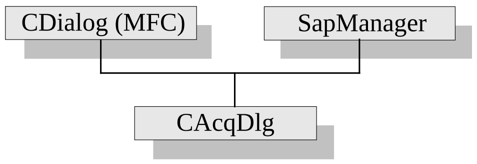
CAcqConfigDlg:: GetAcquisition

SapAcquisition &**GetAcquisition**();

Remarks

Gets the automatically created SapAcquisition object with the new CCF parameters. This object is not the same as the one specified as a constructor argument, if any.

CAcqDlg



The CAcqDlg Class allows you to dynamically adjust the following acquisition parameters:

- Camera selector
- Digital input connect bit ordering
- External trigger options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CAcqDlg Class Members

Construction

CAcqDlg	Class constructor

Member Functions

The following functions are members of the CAcqDlg Class.

CAcqDlg::CAcqDlg

CAcqDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

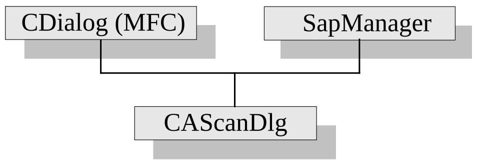
Remarks

The CAcqDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::GetParameter, SapAcquisition::SetParameter after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CAcqDlg dlg(this, m_Acq);
dlg.DoModal();
```

CAScanDlg



The CAScanDlg Class allows you to dynamically adjust the following acquisition parameters related to area scan cameras:

- Trigger/reset/integrate options
- Master mode (camera slave) options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CAScanDlg Class Members

Construction

CAScanDlg	Class constructor

Member Functions

The following functions are members of the CAScanDlg Class.

CAScanDlg::CAScanDlg

CAScanDlg(
  CWnd **pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

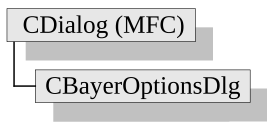
Remarks

The CAScanDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CAScanDlg dlg(this, m_Acq);
dlg.DoModal();
```

CBayerOptionsDlg



The CBayerOptionsDlg Class allows you to adjust the following Bayer conversion parameters:

- Alignment mode
- Pixel value calculation method
- Gamma factor for Bayer lookup table
- White balance gain coefficients (manually or automatically)

As opposed to other GUI Classes, the Bayer dialog is modeless, so that you may define a region of interest (for white balance calculation) on the currently displayed image while keeping the dialog visible. You may create this dialog after calling SapBayer::Create.

#include <SapClassGui.h>

CBayerOptionsDlg Class Members

Construction

CBayerOptionsDlg	Class constructor

Member Functions

The following functions are members of the CBayerOptionsDlg Class.

CBayerOptionsDlg::CBayerOptionsDlg

```
CBayerOptionsDlg(
   CWnd* pParent,
   SapBayer *pBayer,
   SapTransfer *pXfer = NULL,
   CImageWnd *pImageWnd = NULL,
   SapProcessing *pPro = NULL
);
```

Parameters

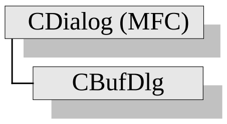
pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBayer	Pointer to the related SapBayer object (cannot be NULL)
pXfer	Pointer to the related SapTransfer object
pImageWnd	Pointer to the related CImageWnd object
pPro	Pointer to the related SapProcessing object

Remarks

The CBayerOptionsDlg constructor does not immediately show the dialog. This happens only when you call its ShowWindow method.
If *pXfer* is NULL, then a buffer with valid Bayer data must be available through the object pointed to by the *pBayer* argument. This is necessary since a software conversion is performed whenever you perform an automatic white balance calculation, or when you click on the Apply button.

If *pXfer* points to a valid SapTransfer object, then the acquisition device is used to acquire Bayer images.
If *pImageWnd* is NULL, then automatic white balance calculation is not available, as it requires a valid region of interest to be drawn on the image display area.
If you specify a non-NULL *pPro* argument, then the execution time becomes available whenever a software Bayer conversion is performed within the dialog.

CBufDlg



The CBufDlg Class allows you to modify the parameters of a SapBuffer object. It actually works on a copy of the object, so that modifications do not affect the original object right away.

You may create a modal version of this dialog after calling SapBayer::Create. When you are finished using the dialog, you may retrieve the modified copy of the buffer object using the GetBuffer method.

#include <SapClassGui.h>

CBufDlg Class Members

Construction	
CBufDlg	Class constructor
Attributes	
GetBuffer	Gets the modified copy of the original SapBuffer object, as modified from the dialog

Member Functions

The following functions are members of the CBufDlg Class.

CBufDlg::CBufDlg

CBufDlg(
  CWnd* *pParent*,
  SapBuffer **pBuffer*,
  SapDisplay **pDisplay* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBuffer	Pointer to the related SapBuffer object
pDisplay	Pointer to an optional SapDisplay object used for displaying the buffer contents

Remarks

The CBufDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
The specified *pBuffer* is not modified when you use the dialog. Instead, an additional SapBuffer object is used internally to keep track of all changes. You may then get the modified object by calling GetBuffer.
If you specify a non-NULL *pDisplay* argument, then this SapDisplay object will be used to automatically determine the availability of offscreen and overlay buffer types for the current buffer format.

You can only call DoModal after SapBuffer::Create.

Example: Modifying an existing SapBuffer object

```
CBufDlg dlg(this, m_Buffers);
if (dlg.DoModal() == IDOK)
{
   // Destroy all objects (including m_Buffers)
   DestroyObjects();

   // Create a backup object
   SapBuffer buf = *m_Buffers;


// Update buffer object
   *m_Buffers = dlg.GetBuffer();

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

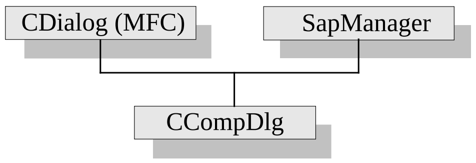
CBufDlg::GetBuffer

SapBuffer &**GetBuffer**();

Remarks

Gets the modified copy of the original SapBuffer object, as modified from the dialog. This object is not the same as the one used in the CBufDlg constructor, it is a completely different instance.

CCompDlg



The CCompDlg Class allows you to dynamically adjust the following acquisition parameters related to composite video input signals:

- Brightness and contrast options
- Hue and saturation options
- Sharpness options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CCompDlg Class Members

Construction

CCompDlg	Class constructor

Member Functions

The following functions are members of the CCompDlg Class.

CCompDlg::CCompDlg

CCompDlg(
   CWnd **pParent*,
   SapAcquisition **pAcq*,
   SapTransfer **pXfer* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object
pXfer	Pointer to a SapTransfer for controlling visual feedback

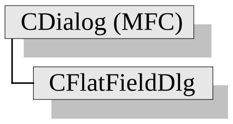
Remarks

The CCompDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
If you supply a non-NULL *pXfer* argument, then a new image is automatically acquired through this SapTransfer object every time you modify one of the values in the dialog. This gives you immediate visual feedback.
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
m_Xfer->Snap();
CCompDlg dlg(this, m_Acq, m_Xfer);
dlg.DoModal();
```

CFlatFieldDlg



The CFlatFieldDlg Class allows you to perform flat-field calibration by either acquiring or loading from files black and white reference images.

You may create a modal version of this dialog after calling SapFlatField::Create.

#include <SapClassGui.h>

CFlatFieldDlg Class Members

Construction

CFlatFieldDlg	Class constructor

Member Functions

The following functions are members of the CFlatFieldDlg Class.

CFlatFieldDlg::CFlatFieldDlg

CFlatFieldDlg(
   CWnd* *pParent*,
   SapFlatField **pFlatField*,
   SapTransfer **pXfer*,
   SapBuffer **pBuffer*
);

Parameters

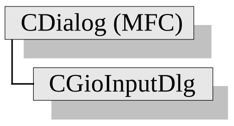
pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pFlatField	Pointer to the related SapFlatField object
pXfer	Pointer to the related SapTransfer object
pBuffer	Pointer to the related SapBuffer object

Remarks

The CFlatFieldDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
If *pXfer* is NULL, then the black and the white images needed for computing the flat-field correction coefficients have to be loaded from files.

If *pXfer* points to a valid SapTransfer object, then these images will be acquired from the acquisition device.

CGioInputDlg



The CGioInputDlg Class includes the following functionality:

- Gets the current pin state (low or high)
- Gets read/write access to the low-level Sapera C library parameter for the I/O module
- Sets the input level type I/O
- Enables/disables a callback function for specific I/O events (rising edge or falling edge)
- Event count for each occuring I/O event

You may create a modal version of this dialog after calling SapGio::Create except for when you use CGioInputDlg to automatically create a SapGio object.

#include <SapClassGui.h>

CGioInputDlg Class Members

Construction

CgioInputDlg	Class constructor

Member Functions

The following functions are members of the CGioInputDlg Class.

CGioInputDlg:: CGioInputDlg

CGioInputDlg(
 CWnd* *pParent*,
 UINT *iDevice*,
 SapGio **pGio*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
iDevice	Input GIO resource index
pGio	Pointer to the related SapGio object

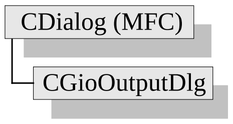
Remarks

The CGioInputDlg constructor does not immediately show the dialog. This occurs only after you call its DoModal method.
You may specify an existing SapGio object through the pGio argument.
You can only call DoModal after SapGio::Create.

Example: Constructor calls for acquisition devices supporting general I/O

```
CGioInputDlg   *m_pDlgInput[MAX_GIO_DEVICE];
//Loop for all resources
for (UINT32 iDevice = 0; (iDevice < MAX_GIO_DEVICE) && (iDevice < m_gioCount); iDevice++) {
m_pDlgInput[iDevice] = new CGioInputDlg(this, iDevice, m_pGio[iDevice]);
if (m_pDlgInput[iDevice] != NULL)
               m_pDlgInput[iDevice]->Create();}
```

CGioOutputDlg



The CGioOutputDlg Class includes the following functionality:

- Gets the number of pins present on the output I/O resource
- Gets/sets the current pin state (low or high)
- Gets/sets the current output type parameter value

You may create a modal version of this dialog after calling SapGio::Create except for when you use CGioOutputDlg to automatically create a SapGio object.

#include <SapClassGui.h>

CGioOutputDlg Class Members

Construction

CGioOutputDlg	Class constructor

Member Functions

The following functions are members of the CGioOutputDlg Class.

CGioOutputDlg::CGioOutputDlg

CGioOutputDlg(
  CWnd* *pParent*,
  UINT *iDevice*,
  SapGio **pGio*);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
iDevice	Output GIO resource index
pGio	Pointer to the related SapGio object

Remarks

The CGioOutputDlg constructor does not immediately show the dialog. This occurs only after you call its DoModal method.
You may specify an existing SapGio object through the pGio argument.
You can only call DoModal after SapGio::Create.

Example: Constructor calls for acquisition devices supporting general I/O

```
CGioOutputDlg  *m_pDlgOutput[MAX_GIO_DEVICE];
//Loop for all resources
for (UINT32 iDevice = 0; (iDevice < MAX_GIO_DEVICE) && (iDevice < m_gioCount); iDevice++) {
m_pDlgOutput[iDevice] = new CGioOutputDlg(this, iDevice, m_pGio[iDevice]);
if (m_pDlgOutput[iDevice] != NULL)
       m_pDlgOutput[iDevice]->Create();}
```

CImageWnd

CImageWnd

The CImageWnd utility class includes the following functionality:

- Manipulates an image window (view window)
- Manipulates scroll bars attached to the image window
- Interactively draws a region-of-interest (ROI) in the image window
- Gets a description of the pixel value at the current mouse position in the image window

Since CImageWnd uses a SapView object, you may use it to simplify the processing of application messages related to the view window such as WM_MOVE, WM_PAINT, WM_SIZE, etc.

There are two ways of using CImageWnd:

- The view window and the scroll bars already exist in a dialog-based application. These controls are specified as arguments to the class constructor.
- In non dialog-based application, the view window is the main application window itself. The scroll bars are then accessed through this window.

#include <SapClassGui.h>

CImageWnd Class Members

Construction	
CImageWnd	Class constructor
Operations (message handling)	
OnMove	Adjusts the position of the view window following a WM_MOVE message
OnPaint	Repaints uncovered areas of the View window following a WM_PAINT message
OnSize	Adjusts the size of the view window following a WM_SIZE message
OnHScroll	Adjusts the horizontal scroll bar following a WM_HSCROLL message
OnVScroll	Adjusts the vertical scroll bar following a WM_VSCROLL message
OnLButtonDown	Begins or ends drawing of the ROI tracker following a WM_LBUTTONDOWN message
OnSetCursor	Sets the cursor shape according to the position of the mouse over the ROI tracker following a WM_SETCURSOR message
Operations (ROI tracker)	
IsRoiTrackerActive	Checks if the ROI tracker is currently displayed
DisplayRoiTracker	Redraws the ROI tracker
HideRoiTracker	Hides the ROI tracker
GetSelectedRoi	Gets the rectangle corresponding to the ROI tracker
SelectRoi	Sets the ROI tracker rectangle
Operations (other)	
IsPointInside	Checks if the specified point is inside the view window
TranslateMousePos	Converts mouse position coordinates to the view window coordinate system
Invalidate	Invalidates the view window client area
UpdateClientArea	Repaints unused regions of the view window client area
GetPixelString	Gets a text description of the pixel value at a specific position in the view window

Member Functions

CImageWnd::CimageWnd

CImageWnd(
  SapView **pView*,
  CWnd **pViewWnd*,
  CScrollBar **pHorzScr*,
  CScrollBar **pVertScr*,
  CWnd **pAppWnd* = NULL
);
CImageWnd(
  SapView **pView*,
  CWnd **pAppWnd*
);

Parameters

pView	SapView object to control
pViewWnd	Existing view window object in a dialog-based application (CWnd is defined in MFC)
pHorzScr	Existing horizontal scroll bar object in a dialog-based application (CScrollBar is defined in MFC)
pVertScr	Existing vertical scroll bar object in a dialog-based application (CScrollBar is defined in MFC)
pAppWnd	Application main window object (CWnd is defined in MFC)

Remarks

If your application is dialog-based, you should use the first form of the CImageWnd constructor. The application must already include view window and scroll bar objects, since they are needed by the constructor. If you supply a NULL value for the *pAppWnd* argument, then the value of *pViewWnd* will be used as the main window.
If your application is Single Document Interface (SDI) or Multiple Document Interface (MDI) based, then you should use the second form of the constructor. In this case, the view window is the main application window itself. The scroll bars are then accessed through this window.

CImageWnd::DisplayRoiTracker

void **DisplayRoiTracker**();

Remarks

Redraws the ROI tracker (clipped to the view window) if it is currently active

CImageWnd::GetPixelString

CString **GetPixelString**(CPoint *point*);

Parameters

point	Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Remarks

Gets a text description of the pixel value a specific position in the view window (CString is defined in MFC). If the point is outside the view window, the string is empty.

CImageWnd::GetSelectedRoi

CRect **GetSelectedRoi**();

Remarks

Gets the rectangle corresponding to the ROI tracker (CRect is defined in MFC)

CImageWnd::HideRoiTracker

void **HideRoiTracker**();

Remarks

Hides the ROI tracker if it is currently active

CImageWnd::Invalidate

void **Invalidate**(BOOL *bErase* = TRUE);

Parameters

bErase	TRUE to also erase the view window background, FALSE otherwise

Remarks

Invalidates the view window client area so that it will be completely redrawn through the OnPaint method

CImageWnd::IsPointInside

BOOL **IsPointInside**(CPoint *point*);

Parameters

point	Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Remarks

Returns TRUE if the specified point is inside the view window, FALSE otherwise

CImageWnd:: IsRoiTrackerActive

BOOL **IsRoiTrackerActive**();

Remarks

Returns TRUE if the ROI tracker is currently displayed, FALSE otherwise.
To activate the ROI tracker, click once in the view window, then keep the left mouse button down while dragging. Release the left button when done.

CImageWnd::OnHScroll

void **OnHScroll**(UINT *nSBCode*, UINT *nPos*);

Parameters

nSBCode	Scroll-bar code that indicates the scrolling request
nPos	Horizontal scroll-bar position

Remarks

Adjusts the horizontal scroll bar following a WM_HSCROLL message. You must call OnHScroll from within your application's WM_HSCROLL message handler, using the *nSBCode* and *nPos* arguments you received in this handler.

CImageWnd::OnLButtonDown

BOOL **OnLButtonDown**(CPoint *point*);

Parameters

point	Mouse position coordinate (CPoint is defined in MFC)

Return Value

Returns TRUE if *point* is inside the view window, FALSE otherwise

Remarks

Begins or ends drawing of the ROI tracker following a WM_LBUTTONDOWN message. You must call OnLButtonDown from within your application's WM_LBUTTONDOWN message handler.
If the mouse position is outside the view window, then the ROI tracker rectangle is clipped to the window.

CImageWnd::OnMove

void **OnMove**();

Remarks

Adjusts the position of the view window following a WM_MOVE message. You must call OnMove from within your application's WM_MOVE message handler.

CImageWnd::OnPaint

void **OnPaint**();

Remarks

Repaints uncovered areas of the view window following a WM_PAINT message. You must call OnPaint from within your application's WM_PAINT message handler.

CImageWnd::OnSetCursor

BOOL **OnSetCursor**(UINT *nHitTest*);

Parameters

nHitTest	Hit test code

Return Value

Returns TRUE if the ROI tracker is active and the cursor position is inside its rectangle, FALSE otherwise

Remarks

Sets the cursor shape according to the position of the mouse over the ROI tracker following a WM_SETCURSOR message. You must call OnSetCursor from within your application's WM_SETCURSOR message handler, using the *nHitTest* argument you received in this handler.
The cursor shape indicates whether you can resize or move the rectangle.

CImageWnd::OnSize

void **OnSize**();

Remarks

Adjusts the size of the view window following a WM_SIZE message. You must call OnSize from within your application's WM_SIZE message handler.
In a dialog-based application, the view window size and the scroll bars are adjusted to reach the bottom right corner of the main application window. Therefore, do not place any controls at the right or at the bottom of the view window, as they will be hidden when running the application.

CImageWnd::OnVScroll

void **OnVScroll**(UINT *nSBCode*, UINT *nPos*);

Parameters

nSBCode	Scroll-bar code that indicates the scrolling request
nPos	Vertical scroll-bar position

Remarks

Adjusts the vertical scroll bar following a WM_VSCROLL message. You must call OnVScroll from within your application's WM_VSCROLL message handler, using the *nSBCode* and *nPos* arguments you received in this handler.

CImageWnd::SelectRoi

void **SelectRoi**(CRect *rect*);

Parameters

rect	New tracker ROI rectangle (CRect is defined in MFC)

Remarks

Sets the ROI tracker rectangle. You must call DisplayRoiTracker after SelectRoi in order to draw the ROI.

CImageWnd::TranslateMousePos

CPoint **TranslateMousePos**(CPoint *point*);

Parameters

point	Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Return Value

Translated point coordinate

Remarks

Converts mouse position coordinates to the view window coordinate system. The current scrolling position is taken into account.

CImageWnd::UpdateClientArea

void **UpdateClientArea**();
void **UpdateClientArea**(COLORREF *color*);

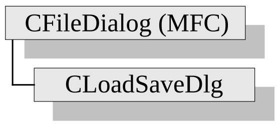
Parameters

color	New view window background color (COLORREF is defined in GDI)

Remarks

Repaints unused regions of the view window client area using either the current background color, or the specified *color* argument. This is useful when the displayed image is smaller than the view window.

CloadSaveDlg



The CLoadSaveDlg Class allows you to load/save an image file from/to a SapBuffer object. You may create a modal version of this dialog after calling SapBuffer::Create, except if you do not supply a SapBuffer object in the constructor.

Depending on the file type, some additional dialog classes are automatically used to specify additional type-specific options:

- CAviFileDlg (AVI files)
- CJpegFileDlg (JPEG files)
- CTiffFileDlg (TIFF files)
- CRawFileDlg (RAW files)

#include <SapClassGui.h>

CLoadSaveDlg Class Members

Construction	
CLoadSaveDlg	Class constructor
Attributes	
GetPathName	Get the full path name of the file to load from or save to
GetOptions	Gets a text string with the load/save options
GetStartFrame	Gets the first frame number to be loaded in a sequenced image file

Member Functions

The following functions are members of the CLoadSaveDlg Class.

CLoadSaveDlg::CLoadSaveDlg

CLoadSaveDlg(
  CWnd* *pParent*,
  SapBuffer **pBuffer*,
  BOOL *isFileOpen* = TRUE,
  BOOL *isSequence* = FALSE
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBuffer	Pointer to a SapBuffer object used for loading/saving the image file (can be NULL)
isFileOpen	TRUE to create a 'File | Load' dialog box, FALSE to create a 'File | Save' dialog box
isSequence	TRUE to include sequenced file types in the filter list (for example, AVI), FALSE to include basic file types (for example, BMP and JPEG).

Remarks

The CLoadSaveDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
There are two ways to use this dialog. The first is to set the *pBuffer* argument to a non-NULL value. The selected image file is then automatically loaded from / saved to the SapBuffer object. This means that you must have called SapBuffer::Create first. This usage is the simpler of the two. However, when loading an image the buffer may not necessarily be compatible with the selected image file causing an automatic data conversion to be performed (for example, trying to load a color image file into a monochrome buffer).

The second way is to set *pBuffer* to NULL. You must then explicitly load/save the image file by calling the SapBuffer::Load and SapBuffer::Save methods after the dialog is closed. You may obtain the necessary arguments from the GetPathName, GetOptions, and GetStartFrame methods. When loading, this allows you to make your buffer compatible with the image file by using either the SapBuffer::SapBuffer(**filename*) constructor or the SapBuffer::SetParametersFromFile method before calling SapBuffer::Create.

Example 1: Loading a non-sequenced image file (in an existing buffer)

```
CLoadSaveDlg dlg(this, m_Buf, TRUE);
dlg.DoModal();
```

Example 2: Loading a non-sequenced image file (in a non-existing buffer)

```
CLoadSaveDlg dlg(this, NULL, TRUE);
dlg.DoModal();

SapBuffer buf(dlg.GetPathName());   // Constructs a compatible buffer
buf.Create();
buf.Load(dlg.GetPathName(), dlg.GetOptions());
```

Example 3: Loading a sequenced image file (in an existing buffer)

```
CLoadSaveDlg dlg(this, m_Buf, TRUE, TRUE);
dlg.DoModal();
```

Example 4: Loading a sequenced image file (in a non-existing buffer)

```
CLoadSaveDlg dlg(this, NULL, TRUE, TRUE);
dlg.DoModal();

SapBuffer buf(dlg.GetPathName());   // Constructs a compatible buffer
buf.Create();
buf.Load(dlg.GetPathName(), dlg.GetOptions(),dlg.GetStartFrame());
```

Example 5: Saving a non-sequenced image file

```
CLoadSaveDlg dlg(this, m_Buf, FALSE);
dlg.DoModal();
```

Example 6: Saving a sequenced image file

```
CLoadSaveDlg dlg(this, m_Buf, FALSE, TRUE);
dlg.DoModal();
```

CLoadSaveDlg::GetOptions

CString **GetOptions**();

Remarks

Gets a text string with the load/save options. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetOptions before SapBuffer::Load or SapBuffer::Save.

CLoadSaveDlg::GetPathName

CString **GetPathName**();

Remarks

Get the full path name of the file to load from / save to. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetPathName before SapBuffer::Load or SapBuffer::Save.

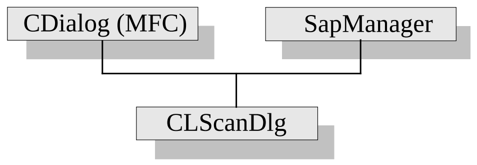
CLoadSaveDlg::GetStartFrame

int **GetStartFrame**();

Remarks

Gets the first frame number to be loaded in a sequenced image file. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetOptions before SapBuffer::Load or SapBuffer::Save.

CLScanDlg



The CLScanDlg Class allows you to dynamically adjust the following acquisition parameters related to linescan cameras:

- Line triggering options
- Frame triggering options
- Shaft encoder options
- Linescan direction options

You may create a modal version of this dialog after calling SapAcquisition::Create. Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CLScanDlg Class Members

Construction

CLScanDlg	Class constructor

Member Functions

The following functions are members of the CLScanDlg Class.

CLScanDlg::CLScanDlg

CLScanDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

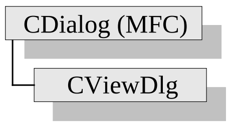
Remarks

The CLScanDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CLScanDlg dlg(this, m_Acq);
dlg.DoModal();
```

CViewDlg



The CViewDlg Class allows you to dynamically adjust the following view parameters:

- Viewing range value

You may create a modal version of this dialog after calling SapView::Create.

#include <SapClassGui.h>

CViewDlg Class Members

Construction

CViewDlg	Class constructor

Member Functions

The following functions are members of the CViewDlg Class.

CViewDlg::CViewDlg

CViewDlg(
   CWnd **pParent*,
   SapView **pView*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pView	Pointer to the related SapView object

Remarks

The CViewDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method. You can only call DoModal after SapView::Create.

Example

```
CViewDlg dlg(this, m_View);
dlg.DoModal();
```

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**.