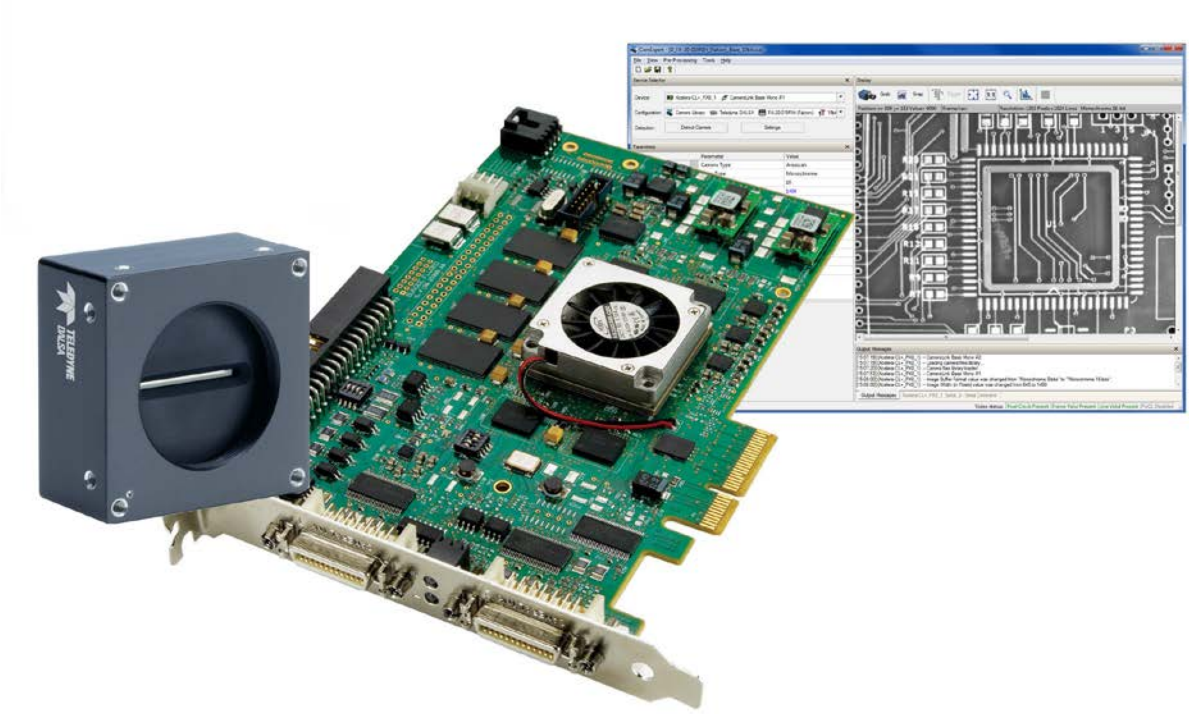
Sapera LT™ 8.20

Camera SDK User's Manual

For GenCP CL Cameras

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



P/N: OC-SAPM-CSDK0

www.teledynedalsa.com

TELEDYNE DALSA
Everywhereyoulook™

NOTICE

Printed on October 20, 2016

Document Number: OC-SAPM-CSDK0

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

Overview

About Sapera Camera SDK

The Sapera Camera SDK is an application programming interface (API) for Teledyne DALSA cameras that support the GenICam (GENeric programming Interface for CAMeras) standard, when those cameras are connected to 3rd party frame grabber boards.

The Sapera Camera SDK allows camera configuration and control to be integrated with imaging applications written using 3rd party APIs. The camera feature set is described using an XML file defined by the GenICam standard.

The Sapera Camera SDK supports current and future GenCP (Generic Control Protocol) devices. GenCP provides a common serial command protocol, currently implemented for Camera Link cameras.

About Sapera++

Sapera++ is a high-level C++ class library dedicated to control and configuration of Teledyne DALSA cameras. The available classes may be subdivided as follows:

- Manager Classes provide image device management functionality.
- Hardware Independent Classes provide everything you need for camera control without worrying about specific camera models.

Hardware independent classes allow one application to control different Teledyne DALSA cameras through the same API. It also guarantees seamless migration to any future Teledyne DALSA camera supported by Sapera LT. The modular architecture provides the user with high programming flexibility and readability.

About Sapera .NET

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

Hardware independent classes allow one application to control different Teledyne DALSA cameras through the same API. It also guarantees seamless migration to any future Teledyne DALSA camera supported by Sapera LT. The modular architecture provides the user with high programming flexibility and readability.

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

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

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

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

System Requirements for Sapera++ and Sapera .NET

- Windows 7, Windows 8 , Windows 8.1 or Windows 10 (32-bit or 64-bit versions)
- **For both 32-bit and 64-bit C++ development**, use one of the following compilers:
  Visual C++ 2005 (with Service Pack 1)
  Visual C++ 2008 (with Service Pack 1)
  Visual C++ 2010
  Visual C++ 2012
  Visual C++ 2013
  Visual C++ 2015
- **For .NET development**, use one of the following development environments:
  Visual Studio 2005 (with Service Pack 1)
  Visual Studio 2008 (with Service Pack 1)
  Visual Studio 2010
  Visual Studio 2012
  Visual Studio 2013
  Visual Studio 2015

If you are using Visual Studio 2005 or 2008 to develop Sapera LT applications, then you should have the latest Service Pack plus all security patches installed. You can check if these are installed through the Help->About menu item in Visual Studio.

Hardware Requirements (optional)

- Any Sapera Camera SDK compatible cameras.

Installation Procedure

For installation notes on tested 3rd party frame grabbers, refer to the Tested 3rd Party Frame Grabber Compatability section.

Install the Sapera Camera SDK as follows:

- If the Sapera Camera SDK was distributed as an executable file, just run this file from your desktop or a temporary local folder.
- If the Sapera Camera SDK was distributed on removable media, insert the installation media and the installer program should run automatically.
- If the installer program does not automatically run, using Windows Explorer execute the **launch.exe** application located in the root directory.
- Select the Sapera Camera SDK Installation and follow the on-screen instructions.
- Reboot the computer when prompted to complete the software installation.
- Also install any 3rd party frame grabber hardware drivers, if not already installed.

Note that installation information to configure a silent install is available in Appendix B: Sapera Camera SDK Installations.

Disk Contents and File Locations

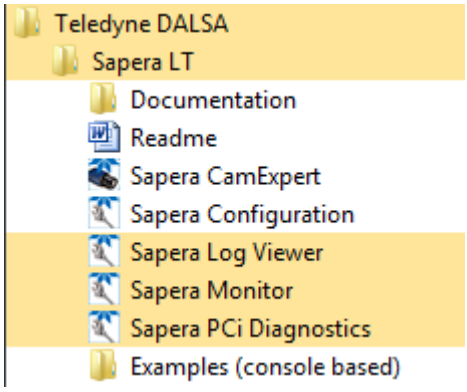
The table below lists the different file groups and locations, where **<SAPERADIR>** refers to the Camera SDK installation directory, usually **C:\Program Files\Teledyne DALSA\Sapera**.

Description	Location
Sapera Camera SDK Readme and version history documents Third-party software licenses (for example, GenICam)	**\<SAPERADIR>**
Utility programs	**\<SAPERADIR>\Bin**
CamExpert camera configuration utility	**\<SAPERADIR>\CamExpert**
Acquisition configuration files	**\<SAPERADIR>\CamFiles** (and device-specific subdirectories)
<SAPERADIR>++ header files	**\<SAPERADIR>\Classes**
.NET classes	**\<SAPERADIR>\Components\NET**
.NET dynamic-link libraries (DLLs)	**\<SAPERADIR>\Components\NET\Bin**
Source code for console-based demo applications	**\<SAPERADIR>\Examples**
Executable files for console-based demo applications	**\<SAPERADIR>\Examples\Binaries**
GenICam runtime support for Teledyne DALSA CameraLink cameras	**\<SAPERADIR>\GenICam_2_3**
Help files	**\<SAPERADIR>\Help**
Header files for C libraries	**\<SAPERADIR>\Include**
Import libraries for Microsoft compilers	**\<SAPERADIR>\Lib**
Dynamic-link libraries (DLLs)	Windows system directory (**windows\system32**)
Device driver files	Windows driver directory (**windows\system32\drivers**)

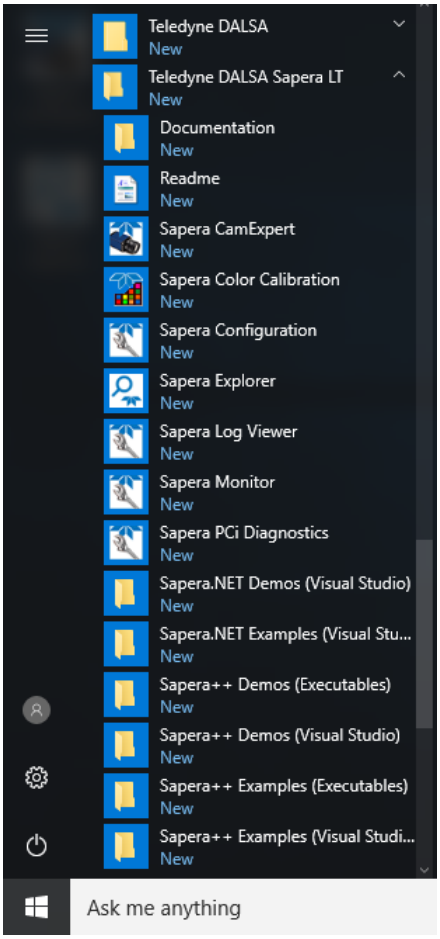
Start Menu Shortcuts

For Windows 7, Start menu shortcuts for Sapera LT are available under **All Programs > Teledyne DALSA > Sapera LT** and **>Sapera Network Imaging Package.** For Windows 10, similar shortcuts are available from the Start menu.

The following screenshots display the menus for the full Sapera LT SDK.

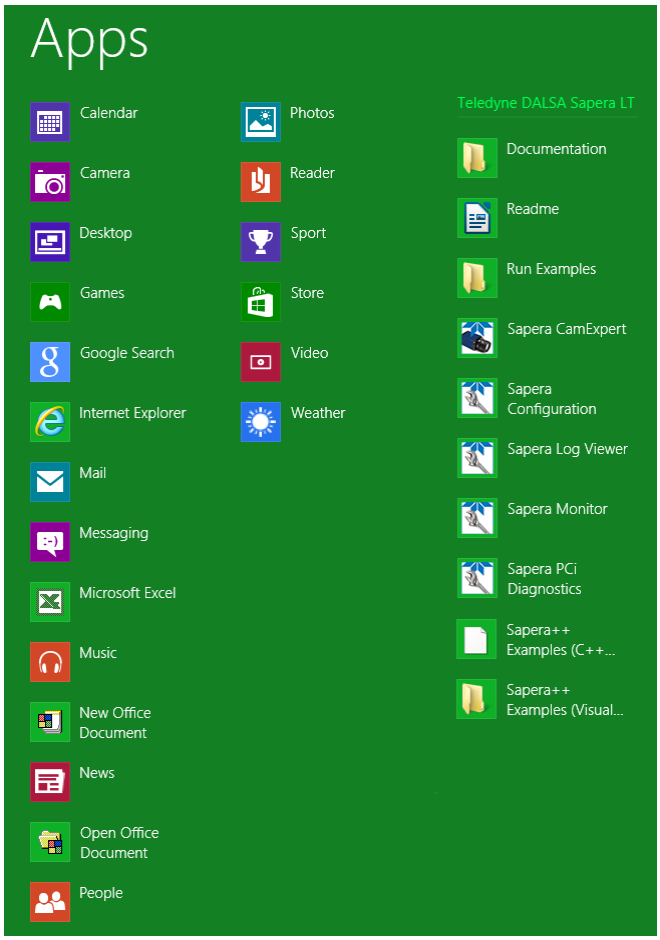


Windows 7



Windows 10

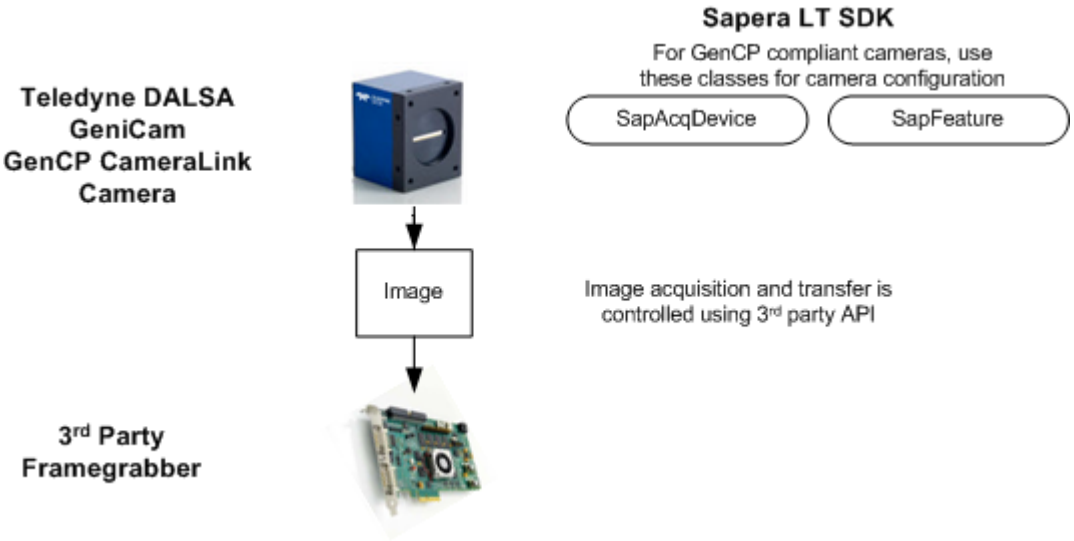
For Windows 8, shortcuts are presented as tiles on the desktop through the Apps menu.



Using Sapera Camera SDK

The Sapera Camera SDK is a subset of the Sapera LT SDK and only includes those classes required for configuring Teledyne DALSA GenICam GenCP protocol compliant cameras and error management. For cameras that support the GenCP protocol, the Sapera LT SDK can be used to configure features dynamically within your application when used in conjunction with a fully compliant Camera Link standard frame grabber.

Acquisition and image transfer is controlled by a 3rd party API.



The Sapera CamExpert application, included with Sapera Camera SDK, can be used to configure Teledyne DALSA Camera Link cameras that support the GenCP protocol. CamExpert allows you to set parameters interactively in the application window. The configuration can then be saved directly on the camera.

	Note: for cameras that use the Teledyne DALSA text-based protocol, the CamExpert serial port window can be used to send user input three letter commands. However, this type of camera is not supported by the Sapera Camera SDK.

What is GenICam?

GenICam™ is an international standard that allows a single application programming interface (API) to control any compliant video source, regardless of its vendor, feature set, or interface technology (GigE Vision®, Camera Link®, etc.).

GenICam consists of four modules:

- GenApi: This module defines the format of an XML file that captures the features of a device. GenApi also specifies how to access and control the features. All GenICam-compliant devices must contain an XML file that conforms to this format.
- Standard Features Naming Convention (SFNC): This module standardizes the names of more than 220 commonly used camera features. To comply with GigE Vision, seven of the features are mandatory. The rest are either recommended or optional. Compliance with the naming convention is important for interoperability, as it frees application software from the complexity of situations where vendors call the same feature by different names, such as, 'Brightness' and 'Gain'.
- GenTL: This module defines a software interface for accessing image data from a generic transport layer.
- GenCP: This module provides a standardized packet based protocol to read and write device registers, and gives access to a GenApi compliant XML file present on the device.

There are two levels of compliance to GenICam:

- GenICam-compliance: where a product either provides or interprets a compliant XML file.
- GenICam TL-compliance: where a product exposes a transport layer compatible with GenTL.

Currently, Teledyne DALSA offers several cameras with GenICam and GigEVision compliance.

Detecting Cameras

Teledyne DALSA GigE Vision cameras are automatically detected after their Framework package is installed. Refer to the camera's manual for installation and configuration instructions along with details to the supported GenICam feature set.

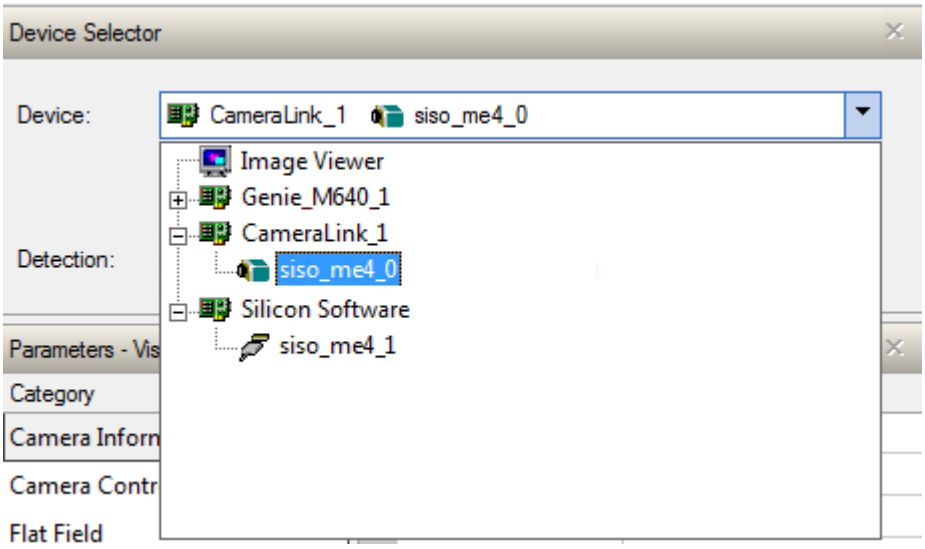
Detection of Teledyne DALSA Camera Link cameras can be enabled with the Sapera Configuration utility or within CamExpert. Detection is dependent on the vendor-specific CameraLink dll (clserxxx.dll (where xxx is the vendor ID)). This file must be available in the C:\CameraLink\Serial directory for Sapera LT to detect the CameraLink serial port.

	Note: Some frame grabber vendors do not supply the required *clserxxx.dll*. Instead the frame grabber offers serial communication ports as Windows COM ports. To configure Sapera CamExpert and the Sapera Camera SDK using a Windows COM port as a Camera Link serial port, refer to the following section Configuring Frame Grabber Windows COM port for Camera Link.

Refer to the camera documentation to see if it supports the GenCP protocol or the Teledyne DALSA text-based protocol.

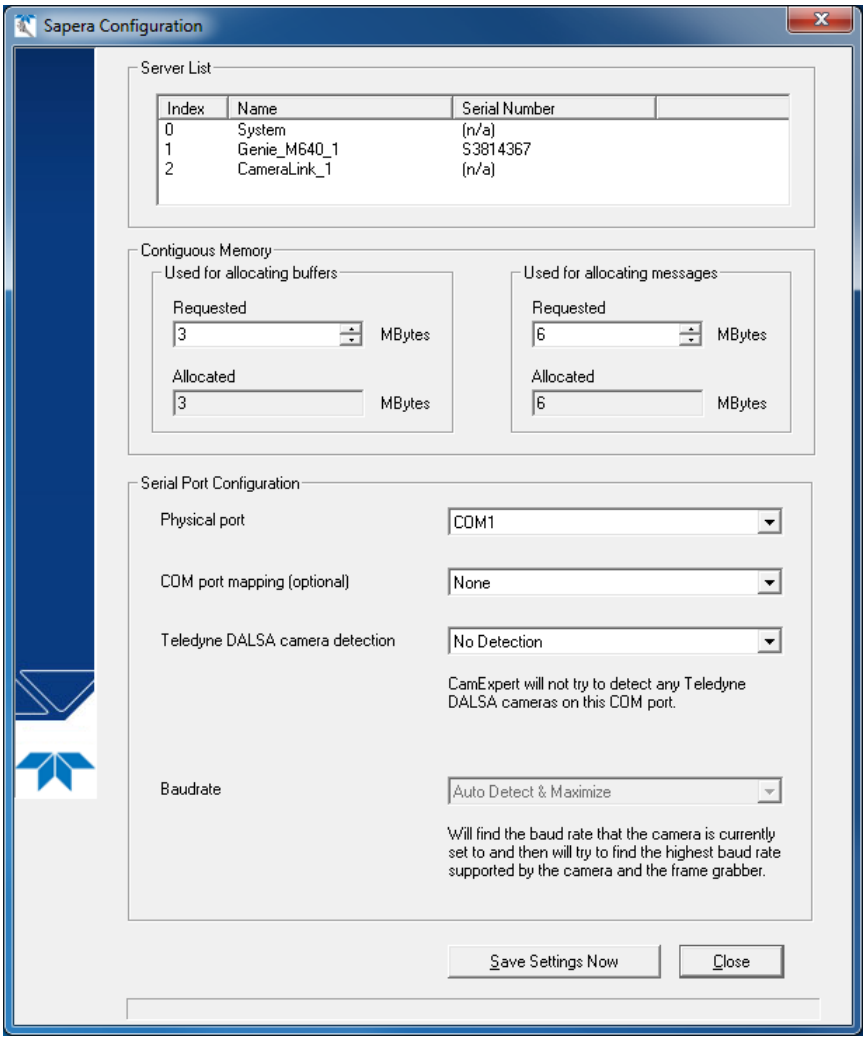
Cameras configured in this manner are then available through the 'CameraLink_x' (where x = 1 , 2, 3, ...) server names through the SapAcqDevice class in Sapera LT SDK applications. They are also available in CamExpert under the 'CameraLink_x' device category.

The following screen capture shows the CamExpert Device Selection menu of a system with both a Teledyne DALSA Genie TS C3500 camera and a third-party frame grabber.



Configuring Sapera Camera SDK

The **Sapera Camera SDK Configuration** program (**SapConf.exe)** allows you to see all the Sapera Camera SDK-compatible devices present within your system, together with their respective serial numbers. After activating this program, it displays all the Sapera servers related to the installed devices as shown in the figure below.



- The **System** entry represents the system server. It corresponds to the host machine (your computer), and is the only server that should always be present. Any other servers correspond to the camera devices detected on the system.
- The **Serial Port Configuration** section allows you to select the frame grabber's serial port and specify the type of Teledyne DALSA camera detection, as well as configure the baud rate.

Configuring Frame Grabber Board Serial Ports

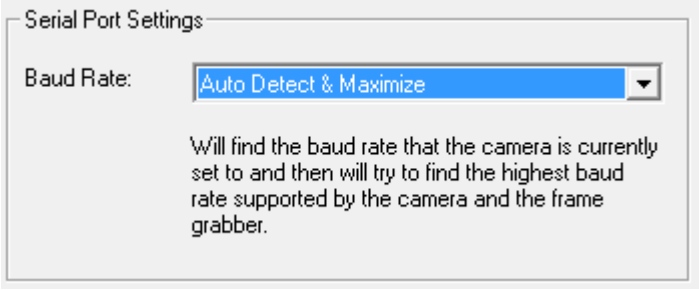
The Sapera Camera SDK and the CamExpert tool send commands to Teledyne DALSA cameras through the 3rd party frame grabber's serial port.

For Camera Link compliant frame grabbers, all the required software support should be installed automatically with the frame grabber's software driver.

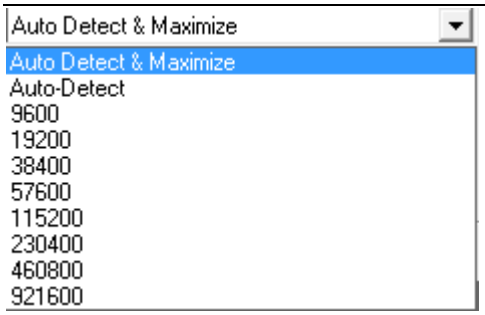
	Note: Some frame grabber vendors do not supply the required *clserxxx.dll*. Instead the frame grabber offers serial communication ports as Windows COM ports. To configure Sapera CamExpert and the Sapera Camera SDK using a Windows COM port as a Camera Link serial port, refer to the following section Configuring Frame Grabber Windows COM port for Camera Link.

Baud Rate

By default, the baud rate is set to **Auto Detect & Maximize**, which will find the baud rate that the camera is currently set to and then find the highest baud rate supported by both the camera and the frame grabber.



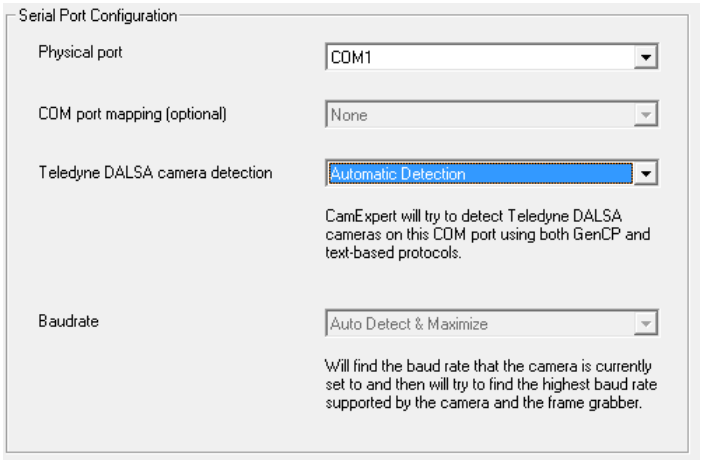
Otherwise, the baud rate can be specifically chosen from among the options available in the drop-down list.



Configuring Frame Grabber Windows COM port for Camera Link

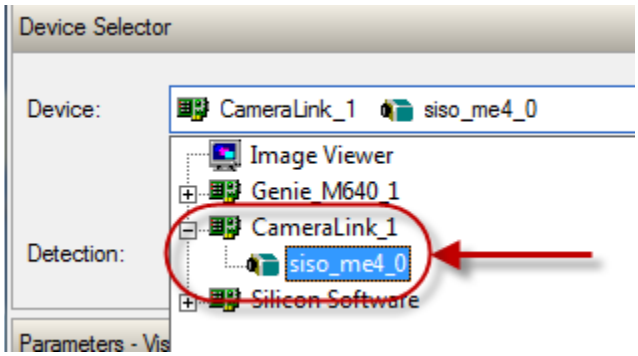
To enable camera detection, and use CamExpert or Sapera Camera SDK to control the camera through a Windows COM port:

- Run the Sapera Configuration utility, select the frame grabber serial port connected to the camera, and set the **Teledyne DALSA camera detection**. Possible values are:
  - Automatic Detection (detects both types of camera)
  - Genicam GenCP for Camera Link
  - Teledyne DALSA Text Based



	Note: If the host computer also has a Teledyne DALSA frame grabber installed, this may prevent the display of other available COM ports.

The camera is now available in CamExpert to adjust parameters; select the appropriate COM port under 'CameraLink_1'.



	Note: If the host computer also has a Teledyne DALSA frame grabber installed, this may prevent the display of other available COM ports.

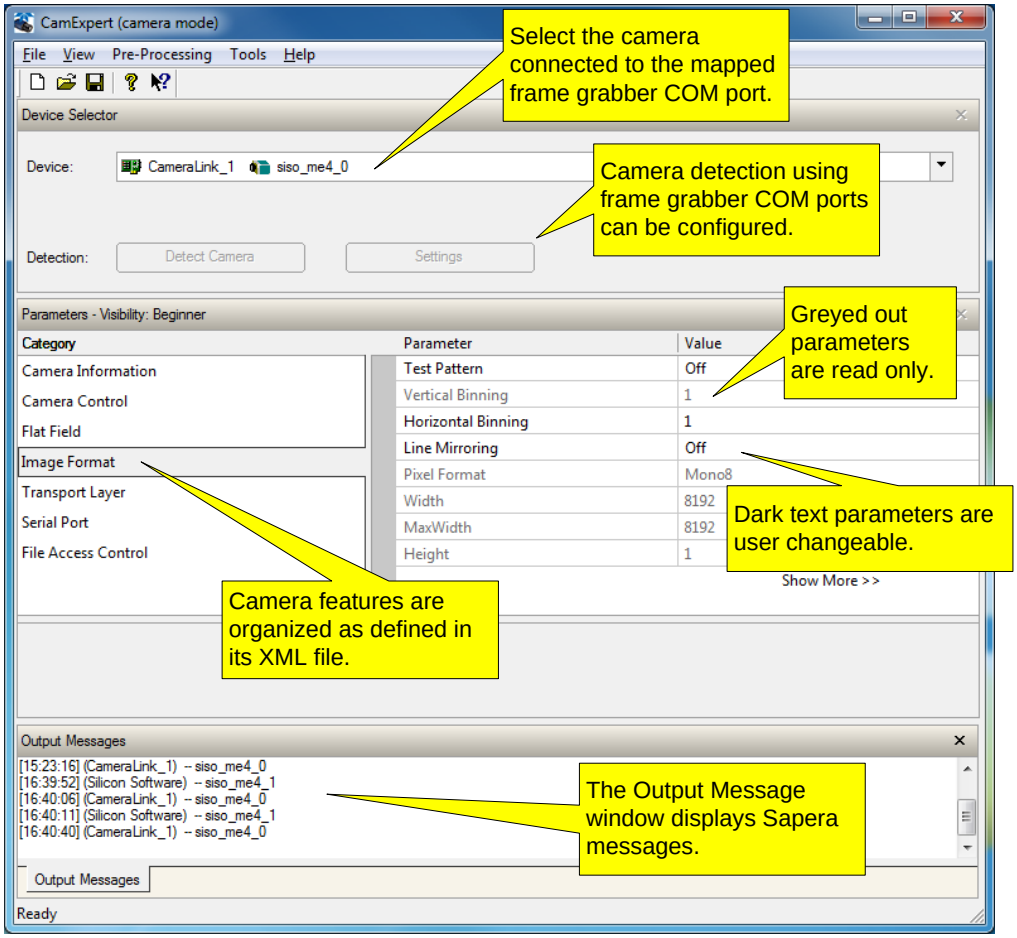
Using the CamExpert Tool with 3rd Party Frame Grabbers

Overview

The Sapera Camera SDK CamExpert tool controls Teledyne DALSA camera link cameras via serial commands passed through the 3rd party frame grabber. The mapping of the frame grabber serial port to a system COM port was described in the previous section (see Configuring Frame Grabber Board Serial Ports).

CamExpert User Interface based on Camera Features

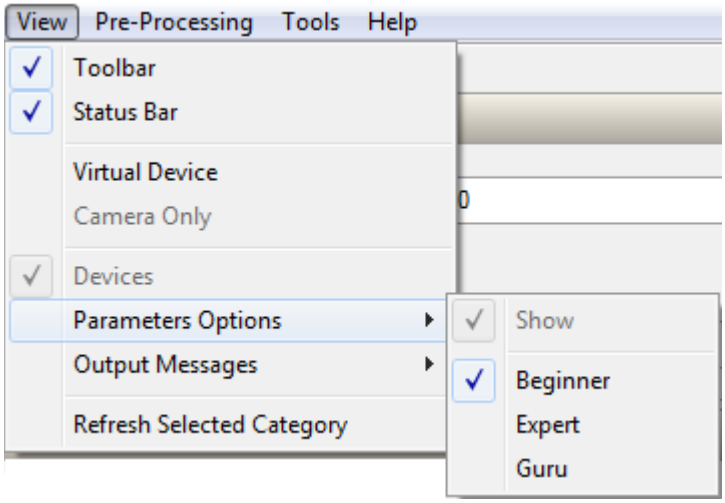
The following screen capture defines each major section of the CamExpert tool. After a camera's parameters are configured, use the 3rd party frame grabber image acquisition application to match the camera timing settings and test an image grab. Additional details about the Sapera CamExpert tool follow in this section.



CamExpert View Parameters Option

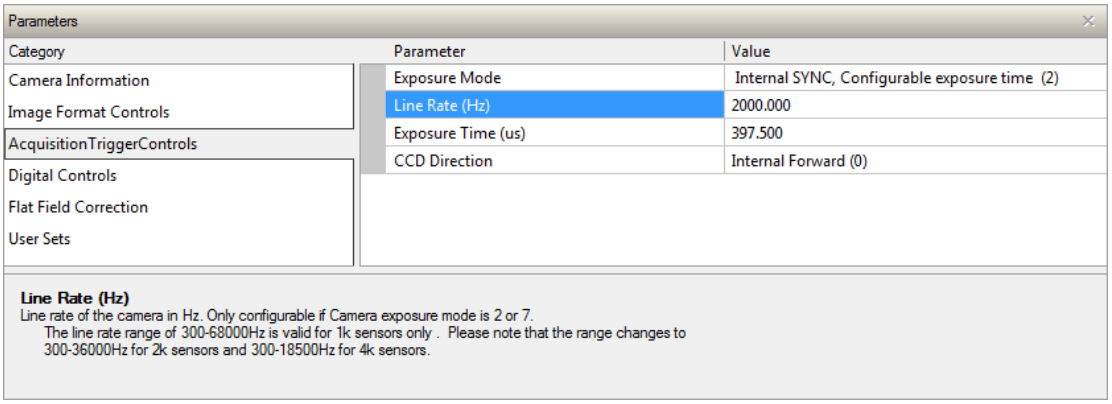
All camera features have a **Visibility** attribute which defines its requirement or complexity. The states vary from **Beginner** (features required for basic operation of the device) to **Guru** (optional features required only for complex operations).

CamExpert presents camera features based on their visibility attribute. The user chooses the Visibility level from the **View · Parameters Options** menu.



CamExpert Parameter Categories

CamExpert displays parameters, grouped in categories, as defined in the camera XML description file. Which of the parameters exposed is dependent on the user's choice of parameter visibility, such as Beginner for basic camera settings to Guru for optional settings. When selecting a parameter, a short description displays in the lower window pane.

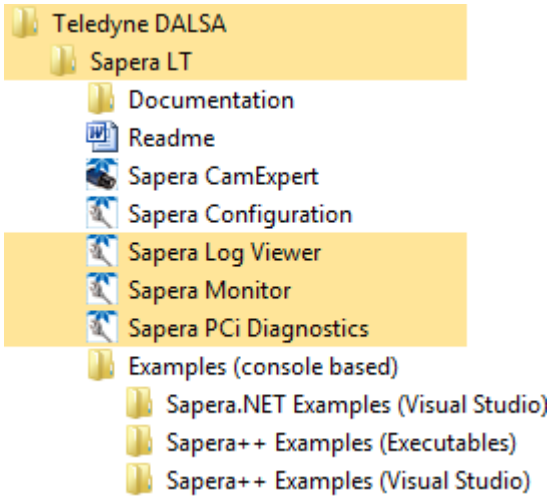


Example Programs

Several generic examples come with Sapera Camera SDK. A description is provided for each in this section. The following image shows the Windows Start menu organization for these samples.

The **Examples** links point to various versions of Microsoft Visual Studio project solution files, where the user then can explore the source code provided. The same examples are ported to C++, C#, and VB.

Note that compiled executables from the C# or VB code examples are not provided.



Description of Programming Examples

Sapera Camera SDK provides several example programs. They are basic applications demonstrating simple tasks Although simple, the purpose of the examples is to provide the user with code samples that can be easily extracted and integrated into an application.

The **Sapera++ examples** are included in the following project solution files, (default installation folder — `c:\Program Files\Teledyne DALSA\Sapera\...`):

Visual Studio 2005	`Examples\Classes\SapExamples_2005.sln`
Visual Studio 2008	`Examples\Classes\SapExamples_2008.sln`
Visual Studio 2010	`Examples\Classes\SapExamples_2010.sln`
Visual Studio 2012	`Examples\Classes\SapExamples_2012.sln`
Visual Studio 2013	`Examples\Classes\SapExamples_2013.sln`
Visual Studio 2015	`Examples\Classes\SapExamples_2015.sln`

The **Sapera .NET** examples are included in the following project solution files:

Visual Studio 2005	C#	`Examples\NET\SapNETC#Examples_2005.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2005.sln`
Visual Studio 2008	C#	`Examples\NET\SapNETC#Examples_2008.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2008.sln`
Visual Studio 2010	C#	`Examples\NET\SapNETC#Examples_2010.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2010.sln`
Visual Studio 2012	C#	`Examples\NET\SapNETCSExamples_2012.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2012.sln`
Visual Studio 2013	C#	`Examples\NET\SapNETCSExamples_2013.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2013.sln`
Visual Studio 2015	C#	`Examples\NET\SapNETCSExamples_2015.sln`
	VB.NET	`Examples\NET\SapNETVBExamples_2015.sln`

These projects allow you to recompile all the examples in a batch.

The Sapera Camera SDK Example Programs

CameraEvents	This example shows using SapAcqDevice to get all available camera events. In addition by using the registering and unregistering callback mechanism, the application tracks when a specific event occurs.
CameraFeatures	This example shows the method to access camera features. The program retrieves the camera features and displays their access mode, data type and value. Additionally there is an example of how to change the value of a feature as well as an example of how to trigger a callback once a value has been changed.
CameraFirmwareUpdate	This example shows how to update firmware for GenCP cameras that support file access, allowing automatic firmware updates at the application level. (GigE and CLHS cameras are also supported by this example.)
FindCamera	This example shows how to detect and list all cameras (both GigE Vision and Teledyne DALSA GenCP cameras) when more than one camera is present. The cameras are listed by user name, serial number, model name or server name.
CameraFiles	This example shows how to upload/download files for GigE or GenCP cameras that support file access.

Sapera Camera SDK Architecture

Definition of Terms

What is a server?

A Sapera server is an abstract representation of a physical device like a camera. A server allows Sapera LT applications to interact with the server's resources.

What is a static resource?

Resources attached to a physical device are called static resources. For example, a camera can have an acquisition device resource. These resources can be manipulated to control a physical device through a Sapera LT server.

What is a dynamic resource?

A dynamic resource is an abstract representation of data storage (such as a lookup table). Unlike static resources, dynamic resources are not dependent on physical devices; therefore, users on a specified server can freely create dynamic resources.

What is a module?

A module is a set of functions used to access and/or control a static or a dynamic resource.

Sapera++ (as a series of C++ classes) or Sapera .NET (as a series of .NET classes) offer the following benefits:

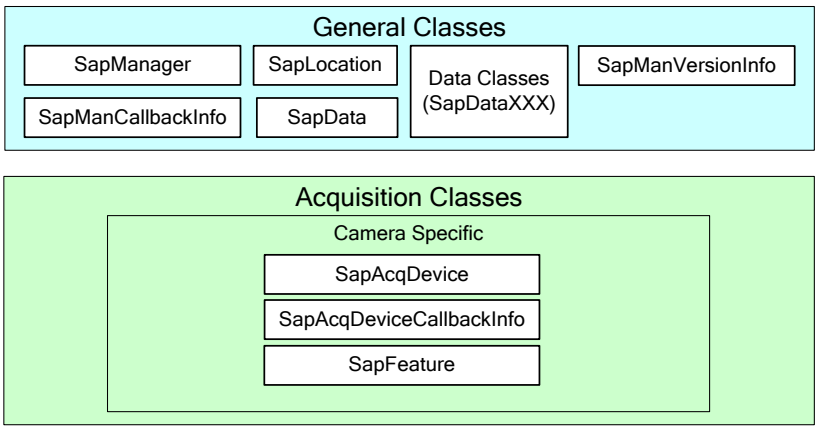
- Easy server management
- Consistent programming interface for static and dynamic resources
- Grouping of modules inside one class whenever appropriate

Sapera Classes

This section provides information on Sapera++ classes applicable to the Camera SDK. Class grouping diagrams are presented for the API followed by class descriptions.

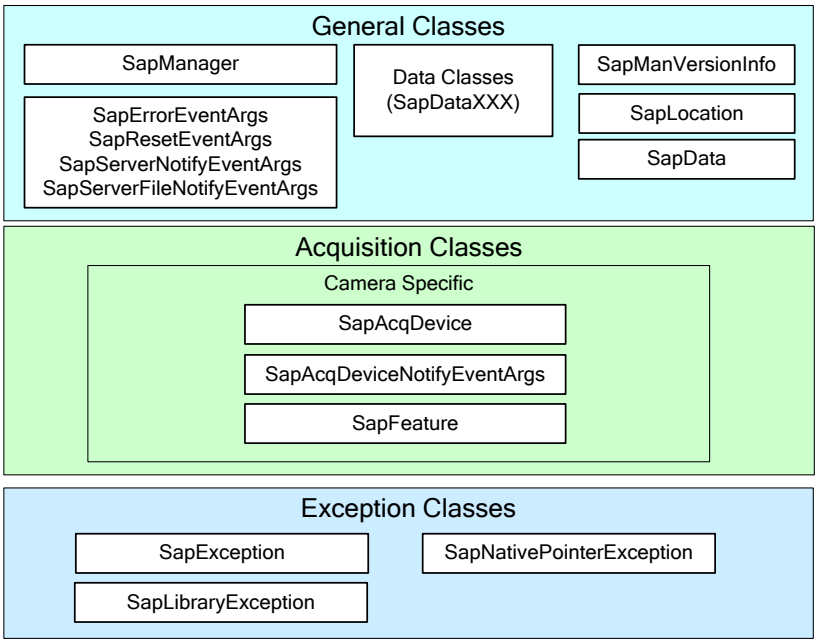
Sapera++ Classes by Subject

The following figure shows the main Sapera++ classes as well as their relationship to other classes.



Sapera .NET Classes by Subject

The following figure shows the main Sapera .NET classes as well as their relationship to other classes.



Sapera++ and Sapera .NET Class Descriptions

Sapera Camera SDK class descriptions cover the purpose of each class and mention any associated Sapera class used. Classes specific to Sapera++ or Sapera .NET are identified as such, otherwise the class applies to both.

SapAcqDevice Class

The SapAcqDevice class includes the functionality to control an acquisition device on any Teledyne DALSA camera.

SapAcqDeviceCallbackInfo Class (Sapera++)

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

SapAcqDeviceNotifyEventArgs Class (Sapera .NET)

The SapAcqDeviceNotifyEventArgs class stores arguments for the AcqDeviceNotify event of the SapAcqDevice class.

SapData and SapDataXxx Classes

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

SapErrorEventArgs Class (Sapera .NET)

The SapErrorEventArgs class stores arguments for the Error event of the SapManager class.

SapException Class (Sapera .NET)

The SapException class is the base class common to the SapLibraryException and SapNativePointerException classes.

SapFeature Class

The SapFeature class includes the functionality to retrieve the feature information from the SapAcqDevice class. Each feature supported by the SapAcqDevice class provides a set of properties such as name, type, access mode, and so forth, that can be obtained through the feature module.

SapLibraryException Class (Sapera .NET)

The SapLibraryException class is thrown when error conditions reported as exceptions occur in the Sapera LT libraries.

SapLocation Class

The SapLocation class identifies a Sapera server/resource pair.

SapManager Class

The SapManager class includes methods for describing the Sapera resources present on the system. It also includes error management capabilities.

SapManCallbackInfo Class (Sapera++)

The SapManCallbackInfo class acts as a container for storing all arguments to callback functions for the SapManager class.

SapManVersionInfo Class

The SapManVersionInfo class includes version information corresponding to the currently installed copy of Sapera LT.

SapNativePointerException Class (Sapera .NET)

The SapNativePointerException class is thrown when internal pointer error conditions occur.

SapResetEventArgs Class (Sapera .NET)

The SapResetEventArgs class stores arguments for the Reset event of the SapManager class.

SapServerFileNotifyEventArgs Class (Sapera .NET)

The SapServerFileNotifyEventArgs contains the arguments to the application handler method for the ServerFileNotify event of the SapManager class.

SapServerNotifyEventArgs Class (Sapera .NET)

The SapServerINotifyEventArgs class stores arguments for the ServerNotify event of the SapManager class.

Using Sapera LT with GenICam-compliant Devices

Sapera LT uses the SapAcqDevice and SapFeature classes to access the GenICam features of a device.

A SapAcqDevice object is created for each acquisition device and provides access to the list of features, events and files that are supported on the device. SapAcqDevice also allows the registering and unregistering of callback functions on an event.

Features

A SapFeature object can be accessed for each feature on the device and provides more detailed information on the actual feature, such as its access mode, minimum and maximum values, enumerations, and so forth, as well as information used for integrating feature access into graphical user interfaces, such as the feature category.

Feature values can be read and written to using the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue functions (C++) or SapAcqDevice.GetFeatureValue Method (and SapAcqDevice.SetFeatureValue Method (.NET). To get more information on a feature, retrieve the SapFeature object for this specific feature using the SapAcqDevice::GetFeatureInfo function SapAcqDevice.GetFeatureInfo Method . See the Sapera++ – Modifying Camera Features and Sapera .NET – Modifying Camera Features for more information and examples on how to access and modify features.

Selectors

A selector is a fundamental concept of GenICamSFNC; it allows using a single feature to control multiple components of the same feature. For example the Gain feature might have three components: Red, Green, and Blue. The SapFeature::IsSelector, SapFeature::GetSelectedFeatureCount, SapFeature::GetSelectedFeatureName, SapFeature::GetSelectingFeatureIndex functions (C++) or SapFeature.IsSelector, SapFeature.SelectedFeatureCount Property, SapFeature.SelectedFeatureNames Property, SapFeature.SelectedFeatureIndexes Property (.NET) and corresponding GetSelecting functions allow the user to query information about the selector.

File Transfer

Sapera LT simplifies the transfer of files to and from devices with the SapAcqDevice::GetFileCount and SapAcqDevice::GetFileNameByIndex functions (C++) or SapAcqDevice.FileCount Property and SapAcqDevice.FileNames Property (.NET), which allow for the enumeration of the available device files. The SapAcqDevice::WriteFile and SapAcqDevice::ReadFile functions (C++) SapAcqDevice.WriteFile Method and SapAcqDevice.ReadFile Method (.NET) are used to transfer the file in and out of the device.

Sapera Supported Features List

The Teledyne DALSA camera features and their currently set values can be read by a Sapera application. The Sapera tool CamExpert is an example of an application that reads and writes GigE Vision or GenCP camera parameters to control its operation.

Accessing Features with Sapera++ and Sapera .NET

When working with camera features, the **SapAcqDevice** class provides functions for accessing those acquisition device features. The class also contains the functionality for sending commands and registering events to devices.

- **GetFeatureInfo** returns information on a feature associated with a specified name or index.
- **SetFeatureValue** writes a new value for a feature.
- All information about the feature is stored in a **SapFeature** class object.

Sapera++ – Object Initialization and Cleanup

Sapera++ objects that encapsulate management of Sapera resources are initialized and refreshed in a uniform way, which consists of the following steps:

- Allocate memory for the object
- Create the resources needed by the object through the Create method
- Destroy the resources for the object through the Destroy method
- Release the memory for the object

Notes on Using the Sapera Camera SDK API

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

```
SapFeature staticFeature;           // Wrong !!
      SapFeature *pFeature;         // OK

      DllMain()
      {
            pFeature = new SapFeature();  // Wrong !!
            delete pFeature;              // Wrong !!
      }

      MyFunction()
{
            pFeature = new SapFeature();  // OK
            delete pFeature;              // OK
      }
```

Example with SapAcqDevice Class Objects

There is more than one way to do this, as shown next for SapAcqDevice class objects:

```
// The usual way to create the object is through a pointer
SapAcqDevice *pAcqDevice = new SapAcqDevice(location);

if (pAcqDevice->Create())
{
   // Acquisition object is correctly initialized
}

// Destroy the acquisition resources after checking if it is still initialized
// through the 'operator BOOL' for the SapAcqDevice class
if (*pAcqDevice)
{
   pAcqDevice->Destroy();
}

// Release the object memory
delete pAcqDevice;
pAcqDevice = NULL;
```

```
// Create the object on the stack
SapAcqDevice acqDevice(location);

if (acqDevice.Create())
{
   // Acquisition object is correctly initialized
   // Destroy the acquisition resources
   acqDevice.Destroy();
}

// The object memory is automatically released when it goes out of scope
```

Sapera++ objects that do not encapsulate management of Sapera resources are correctly initialized as soon as their constructor has been called.

```
SapDataMono data(123);

// The object memory is automatically released when it goes out of scope
```

Sapera .NET – Object Initialization and Cleanup

Sapera .NET objects are initialized and cleaned up in a uniform way, which is described by the following steps:

- Allocate memory for the object
- Create the resources needed by the object through the Create method
- Destroy the resources for the object through the Destroy method
- Release unmanaged memory used internally through the Dispose method

Example with SapAcqDevice Class Objects in C#:

```
SapAcqDevice acqDevice = new SapAcqDevice(location);

if (acqDevice.Create())
{
   // Acquisition object is correctly initialized
}

// Destroy the acquisition resources after checking if it is still initialized
if (acqDevice.Initialized)
{
   acqDevice.Destroy();
}

// Release unmanaged memory used internally
acqDevice.Dispose();
```

Equivalent Code for Visual Basic .NET:

```
Dim AcqDevice As SapAcqDevice = New SapAcqDevice(location)

If AcqDevice.Create() Then
   ' Acquisition object is correctly initialized
End If

' Destroy the acquisition resources after checking if it is still initialized
If AcqDevice.Initialized Then
   AcqDevice.Destroy()
End If

' Release unmanaged memory used internally
AcqDevice.Dispose()
```

Sapera++ – Error Management

Most Sapera++ methods return a Boolean TRUE/FALSE result to indicate success or failure. However, the actual errors conditions are still reported as soon as they happen using one of five predefined reporting modes:

- Error messages are sent to a popup window (the default)
- Error messages are sent to the Sapera Log Server (displayed using the Sapera Log Viewer)
- Error messages are sent to the active debugger if any
- Error messages are generated internally
- Error messages are sent to the application through a call back function

Setting the Current Reporting Mode

Use the SapManager::SetDisplayStatusMode method to set the current reporting mode, as follows:

```
// Send error messages to the Sapera Log Server
SapManager::SetDisplayStatusMode(SapManager::StatusLog);

// Send error messages to the debugger
SapManager::SetDisplayStatusMode(SapManager::StatusDebug);

// Simply generate error messages
SapManager::SetDisplayStatusMode(SapManager::StatusCustom);

// Send errors to application using a callback function
SapManager::SetDisplayStatusMode(SapManager::StatusCallback);

// Restore default reporting mode
SapManager::SetDisplayStatusMode(SapManager::StatusNotify);
```

Monitoring Errors

No matter which reporting mode is currently active, it is always possible to retrieve the latest error message. If the error happened when Sapera++ called a low level Sapera function, then a related numeric code is also available. In order the retrieve this information, call the SapManager::GetLastStatus method as follows:

```
// Get the latest error message
char errorDescr[256];
strcpy(errorDescr, SapManager::GetLastStatus());

// Get the latest error code
// See the Sapera Basic Modules Reference Manual for details
SAPSTATUS lastError;
SapManager::GetLastStatus(&lastError);
```

In addition, the Sapera Log Viewer utility program, included with Sapera Camera SDK, provides an easy way to view error messages. It includes a list box that stores these messages as soon as the errors happen. Available options allow you to modify the different fields for display.

During development it is recommended to start the Log Viewer before your application and then let it run so it can be referred to any time a detailed error description is required. However, errors are actually stored by the Sapera Log Server (running in the background), even if the utility is not running. Therefore it is possible to start the Log Viewer only when a problem occurs with your application.

Sapera .NET – Error Management

Most Sapera .NET methods return a Boolean result to indicate success or failure. However, the actual errors conditions are still reported as soon as they happen using one of five predefined reporting modes:

- Error messages are sent to a popup window (the default)
- Error messages are sent to the Sapera Log Server (displayed using the Sapera Log Viewer)
- Error messages are sent to the application through an event
- Error messages are sent to the application through an exception
- Error messages are generated internally, but not reported immediately

Setting the Current Reporting Mode

Use the DisplayStatusMode property of the SapManager class to set the current reporting mode as follows:

Example of Error Management with C#:

```
// Send error messages to the Sapera Log Server
SapManager.DisplayStatusMode = SapManager.StatusMode.Log;

// Send errors to application through the Error event
SapManager.DisplayStatusMode = SapManager.StatusMode.Event;
SapManager.Error += new SapErrorHandler(SapManager_Error);

SapManager.Error -= new SapErrorHandler(SapManager_Error);

// Send errors to application through an exception
SapManager.DisplayStatusMode = SapManager.StatusMode.Exception;

// try
{
   // Code that possibly generates an error
}
catch (SapLibraryException exception)
{
   // Exception handling code
}

// Just generate error messages
SapManager.DisplayStatusMode = SapManager.StatusMode.Custom;

// Restore default reporting mode
SapManager.DisplayStatusMode = SapManager.StatusMode.Popup;
```

Equivalent Code for Visual Basic .NET:

```
' Send error messages to the Log Viewer
SapManager.DisplayStatusMode = SapManager.StatusMode.Log

' Send errors to application through the Error event
SapManager.DisplayStatusMode = SapManager.StatusMode.Event
AddHandler SapManager.Error, AddressOf SapManager_Error

RemoveHandler SapManager.Error, AddressOf SapManager_Error

' Send errors to application through an exception
SapManager.DisplayStatusMode = SapManager.StatusMode.Exception

Try
   ' Code that possibly generates an error
Catch exception As SapLibraryException
   ' Exception handling code
End Try

' Just generate error messages
SapManager.DisplayStatusMode = SapManager.StatusMode.Custom

' Restore default reporting mode
SapManager.DisplayStatusMode = SapManager.StatusMode.Popup
```

Event Handling Method Definition for C#:

```
public static void SapManager_Error(Object sender, SapErrorEventArgs args)
{
   // Code to handle the Error event of the SapManager class
}
```

Equivalent Code for Visual Basic .NET:

```
Sub SapManager_Error(ByVal sender As Object, ByVal args As SapErrorEventArgs)
    ' Code to handle the Error event of the SapManager class
End Sub
```

Monitoring Errors

No matter which reporting mode is currently active, it is always possible to retrieve the latest error message. If the error happened when Sapera .NET called a Standard API function, then a related numeric code is also available. In order the retrieve this information use the LastStatusMessage and LastStatusCode properties of the SapManager class.

Example to Monitor Errors in C#:

```
// Get the latest error message
string lastMessage = SapManager.LastStatusMessage;

// Get the latest error code
// See the Sapera Basic Modules Reference Manual for details
SapStatus lastCode = SapManager.LastStatusCode;
```

Equivalent Code for Visual Basic .NET:

```
' Get the latest error message
Dim lastMessage As String = SapManager.LastStatusMessage

' Get the latest error code
' See the Sapera Basic Modules Reference Manual for details
Dim lastCode As SapStatus = SapManager.LastStatusCode
```

In addition, the Sapera Log Viewer utility program included with Sapera LT provides an easy way to view error messages. It includes a list box that stores these messages as soon as the errors happen. Available options allow you to modify the different fields for display.

During development it is recommended to start the Log Viewer before your application and then let it run so it can be referred to any time a detailed error description is required. However, errors are actually stored by the Sapera Log Server (running in the background), even if the utility is not running. Therefore it is possible to start the Log Viewer only when a problem occurs with your application.

Sapera++ – Modifying Camera Features

The following section describes how to modify camera features individually or by group. These features are defined in a camera XML file, which is either host-based or stored directly in the on-camera firmware. They are then accessible through the SapAcqDevice and SapFeature classes.

Accessing Feature Information and Values

The following example shows how Teledyne DALSA camera features are accessed. Information such as type, range and access mode can be retrieved for each supported feature. The *SapAcqDevice* class also allows modifying the feature values by directly writing to the camera. In some circumstances, a set of feature values are tightly coupled together and must be written to the camera at the same time. The next section shows how to proceed in such a case.

Sample Code for Sapera++

```
//
// Callback Function
//
void CameraCallback(SapAcqDeviceCallbackInfo* pInfo)
{
   BOOL status;
   int eventCount;
   int eventIndex;
   char eventName[64];

   // Retrieve count, index and name of the received event
   status = pInfo->GetEventCount(&eventCount);
   status = pInfo->GetEventIndex(&eventIndex);
   status = pInfo->GetAcqDevice()->GetEventNameByIndex(eventIndex, eventName,
            sizeof(eventName));

   // Check for "Feature Value Changed" event
   if (strcmp(eventName, "Feature Value Changed") == 0)
   {
      // Retrieve index and name of the feature that has changed
      int featureIndex;
      char featureName[64];
      status = pInfo->GetFeatureIndex(&featureIndex);
      status = pInfo->GetAcqDevice()->GetFeatureNameByIndex(featureIndex, featureName,
               sizeof(featureName));
   }
}

//
// Main Program
//
main()
{
   BOOL status;

   // Create a camera object
   SapAcqDevice camera("CameraLink_1");
   status = camera.Create();

   // Get the number of features provided by the camera
   int featureCount;
   status = camera.GetFeatureCount(&featureCount);

   // Create an empty feature object (to receive information)
   SapFeature feature("CameraLink_1");
   status = feature.Create();

   //
   // Example 1 : Browse through the feature list
   //
   int featureIndex;
   for (featureIndex = 0; featureIndex < featureCount; featureIndex++)
   {
      char featureName[64];
      SapFeature::Type featureType;
      SapFeature::AccessMode featureAccessMode;
```

```
   // Get information from current feature
   // Get feature object
   status = camera.GetFeatureInfo(featureIndex, &feature);

   // Extract name and type from object
   status = feature.GetName(featureName, sizeof(featureName));
   status = feature.GetType(&featureType);
   status = feature.GetAccessMode(&featureAccessMode);

   // Get/set value from/to current feature
   switch (featureType)
   {
      // Feature is a 64-bit integer
   case SapFeature::TypeInt64:
      {
         UINT64 value;
         if (featureAccessMode == SapFeature::AccessRW)
         {
            status = camera.GetFeatureValue(featureIndex, &value);
            value += 10;
            status = camera.SetFeatureValue(featureIndex, value);
         }
      }
      break;

      // Feature is a boolean
   case SapFeature::TypeBool:
      {
         BOOL value;
         if (featureAccessMode == SapFeature::AccessRW)
         {
            status = camera.GetFeatureValue(featureIndex, &value);
            value = !value;
            status = camera.SetFeatureValue(featureIndex, value);
         }
      }
      break;

      // Other feature types
      // ...
   }
}

//
// Example 2 : Access specific feature (integer example)
//
// Get feature object
status = camera.GetFeatureInfo("Gain", &feature);

// Extract minimum, maximum and increment values
UINT32 min, max, inc;
status = feature.GetMin(&min);
status = feature.GetMax(&max);
status = feature.GetInc(&inc);

// Read, modify and write value
UINT32 value;
status = camera.GetFeatureValue("Gain", &value);
value += 10;
status = camera.SetFeatureValue("Gain", value);

//
// Example 3 : Access specific feature (enumeration example)
//
// Get feature object
status = camera.GetFeatureInfo("DeviceScanType", &feature);

// Get number of items in enumeration
int enumCount;
status = feature.GetEnumCount(&enumCount);

int enumIndex, enumValue;
char enumStr[64];
for (enumIndex = 0; enumIndex < enumCount; enumIndex++)
{
   // Get item string and value
   status = feature.GetEnumString(enumIndex, enumStr, sizeof(enumStr));
```

```
      status = feature.GetEnumValue(enumIndex, &enumValue);
   }

   // Read a value and get its associated string
   status = camera.GetFeatureValue("DeviceScanType", &enumValue);
   status = feature.GetEnumStringFromValue(enumValue, enumStr, sizeof(enumStr));

   // Write a value corresponding to known string
   status = feature.GetEnumValueFromString("Linescan", &enumValue);
   status = camera.SetFeatureValue("DeviceScanType", enumValue);

   //
   // Example 4 : Callback management
   //
   // Browse event list
   int numEvents;
   status = camera.GetEventCount(&numEvents);

   int eventIndex;
   for (eventIndex = 0; eventIndex < numEvents; eventIndex++)
   {
      char eventName[64];
      status = camera.GetEventNameByIndex(eventIndex, eventName, sizeof(eventName));
   }

   // Register event by name
   status = camera.RegisterCallback("Feature Value Changed", CameraCallback, NULL);

   // Modified a feature (Will trigger callback function)
   status = camera.SetFeatureValue("Gain", 80);

   // Unregister event by name
   status = camera.UnregisterCallback("Feature Value Changed");
}
```

Writing Feature Values by Group

When a series of features are tightly coupled, they are difficult to modify without following a specific order. For example, a region-of-interest (ROI) has four values (OffsetX, OffsetY, Width and Height) that are inter-dependent and must be defined as a group. To solve this problem, the *SapAcqDevice* class allows you to temporarily set the feature values in an “internal cache” and then download the values to the camera at the same time. The following code illustrates this process using an ROI example.

Sample Code for Sapera++

```
   …
   // Set manual mode to update features
   success = pAcq->SetUpdateFeatureMode(SapAcqDevice::UpdateFeatureManual);

   // Set buffer left position (in the internal cache only)
   success = pAcq->SetFeatureValue("OffsetX", 50);

   // Set buffer top position (in the internal cache only)
   success = pAcq->SetFeatureValue("OffsetY", 50);

   // Set buffer width (in the internal cache only)
   success = pAcq->SetFeatureValue("Width", 300);

   // Set buffer height (in the internal cache only)
   success = pAcq->SetFeatureValue("Height", 300);

   // Write features to device (by reading values from the internal cache)
   success = pAcq->UpdateFeaturesToDevice();

   // Set back the automatic mode
   success = pAcq->SetUpdateFeatureMode(SapAcqDevice::UpdateFeatureAuto);

   …
```

Sapera .NET – Modifying Camera Features

The following section describes how to modify camera features individually or by group. These features are defined in a camera XML file, which is either host-based or stored directly in the on-camera firmware. They are then accessible through the SapAcqDevice and SapFeature classes.

Accessing Feature Information and Values

The following example shows how Teledyne DALSA camera features are accessed. Information such as type, range and access mode can be retrieved for each supported feature. The SapAcqDevice class also allows modifying the feature values by directly writing to the camera. In some circumstances a set of feature values are tightly coupled together and must be written to the camera at the same time. The next section shows how to proceed in such a case.

Sample Code for C#

```
// Event handler
public static void
AcqDevice_AcqDeviceNotify(Object sender, SapAcqDeviceNotifyEventArgs args)
{
   SapAcqDevice acqDevice = sender as SapAcqDevice;

   // Retrieve count, index and name of the received event
   int eventCount = args.EventCount;
   int eventIndex = args.EventIndex;
   string eventName = acqDevice.EventNames[eventIndex];

   // Check for "Feature Value Changed" event
   if (eventName == "Feature Value Changed")
   {
      // Retrieve index and name of the feature that has changed
      int featureIndex = args.FeatureIndex;
      string featureName = acqDevice.FeatureNames[featureIndex];
   }
}

static void Main(string[] args)
{
   // Allocate acquisition object using default camera settings,
   // and create resources
   SapAcqDevice acqDevice =
      new SapAcqDevice(new SapLocation("CameraLink_1", 0));
   bool success = acqDevice.Create();

   // Get the number of features provided by the camera
   int featureCount = acqDevice.FeatureCount;

   // Create an empty feature object (to receive information),
   // and create resources
   SapFeature feature = new SapFeature(new SapLocation("CameraLink_1", 0));
   success = feature.Create();

   //
   // Example 1 : Browse through the feature list
   //
   for (int featureIndex = 0; featureIndex < featureCount; featureIndex++)
   {
      // Get information from current feature
      // Get feature object
      success = acqDevice.GetFeatureInfo(featureIndex, feature);

      // Extract name and type from object
      string featureName = feature.Name;
      SapFeature.Type featureDataType = feature.DataType;
      SapFeature.AccessMode featureAccessMode = feature.DataAccessMode;

      // Get/set value from/to current feature
      switch (featureDataType)
      {
         // Feature is a 64-bit integer
         case SapFeature.Type.Int64:
         {
```

```
            long localFeatureValue;
            if (featureAccessMode == SapFeature.AccessMode.RW)
            {
               success = acqDevice.GetFeatureValue(
                  featureIndex, out localFeatureValue);
               localFeatureValue += 10;
               success = acqDevice.SetFeatureValue(
                  featureIndex, localFeatureValue);
            }
         }
         break;

         // Feature is a boolean
         case SapFeature.Type.Bool:
         {
            bool localFeatureValue;
            if (featureAccessMode == SapFeature.AccessMode.RW)
            {
               success = acqDevice.GetFeatureValue(
                  featureIndex, out localFeatureValue);
               localFeatureValue = !localFeatureValue;
               success = acqDevice.SetFeatureValue(
                  featureIndex, localFeatureValue);
            }
         }
         break;

         // Other feature types
         // ...
      }
   }

   //
   // Example 2 : Access specific feature (integer example)
   //
   // Get feature object
   success = acqDevice.GetFeatureInfo("Gain", feature);

   // Extract minimum, maximum and increment values
   int featureValueMin;
   int featureValueMax;
   int featureValueIncrement;

   success = feature.GetValueMin(out featureValueMin);
   success = feature.GetValueMax(out featureValueMax);
   success = feature.GetValueIncrement(out featureValueIncrement);

   // Read, modify and write value
   int featureValue;
   success = acqDevice.GetFeatureValue("Gain", out featureValue);
   featureValue += 10;
   success = acqDevice.SetFeatureValue("Gain", featureValue);

   //
   // Example 3 : Access specific feature (enumeration example)
   //
   // Get feature object
   success = acqDevice.GetFeatureInfo("DeviceScanType", feature);

   // Get number of items in enumeration
   int featureEnumCount = feature.EnumCount;

   // Get all enumeration strings and values
   string[] featureEnumText = feature.EnumText;
   int[] featureEnumValues = feature.EnumValues;

   // Get individual enumeration strings and values
   string enumText;
   int enumValue;

   for (int featureEnumIndex = 0; featureEnumIndex < featureEnumCount;
        featureEnumIndex++)
   {
      enumText = featureEnumText[featureEnumIndex];
      enumValue = featureEnumValues[featureEnumIndex];
   }

   // Read a value and get its associated string
```

```
   success = acqDevice.GetFeatureValue("DeviceScanType", out enumValue);
   success = feature.GetEnumTextFromValue(enumValue, out enumText);

   // Write a value corresponding to known string
   success = feature.GetEnumValueFromText("Linescan", out enumValue);
   success = acqDevice.SetFeatureValue("DeviceScanType", enumValue);

   //
   // Example 4 : Callback management
   //
   // Get all event names
   string[] eventNames = acqDevice.EventNames;

   // Browse event list
   int numEvents = acqDevice.EventCount;

   for (int eventIndex = 0; eventIndex < numEvents; eventIndex++)
   {
      string eventName = eventNames[eventIndex];
   }

   // Enable event by name
   success = acqDevice.EnableEvent("Feature Value Changed");
   acqDevice.AcqDeviceNotify +=
      new SapAcqDeviceNotifyHandler(AcqDevice_AcqDeviceNotify);

   // Modify a feature (will trigger an event)
   success = acqDevice.SetFeatureValue("Gain", 80);

   // Disable event by name
   acqDevice.AcqDeviceNotify -=
      new SapAcqDeviceNotifyHandler(AcqDevice_AcqDeviceNotify);

   // Release resources for all objects
   success = lut.Destroy();
   success = feature.Destroy();
   success = acqDevice.Destroy();

   // Free all objects
   lut.Dispose();
   feature.Dispose();
   acqDevice.Dispose();
}
```

Equivalent Code for Visual Basic .NET

```
' Event handler
Sub AcqDevice_AcqDeviceNotify(ByVal sender As Object, _
      ByVal args As SapAcqDeviceNotifyEventArgs)
   Dim acqDevice As SapAcqDevice = sender

   ' Retrieve count, index and name of the received event
   Dim eventCount As Integer = args.EventCount
   Dim eventIndex As Integer = args.EventIndex
   Dim eventName As String = acqDevice.EventNames(eventIndex)

   ' Check for "Feature Value Changed" event
   If eventName = "Feature Value Changed" Then
      ' Retrieve index and name of the feature that has changed
      Dim featureIndex As Integer = args.FeatureIndex
      Dim featureName As String = acqDevice.FeatureNames(featureIndex)
   End If
End Sub

Sub Main()
   ' Allocate acquisition object using default camera settings,
   ' and create resources
   Dim acqDevice As SapAcqDevice = _
      New SapAcqDevice(New SapLocation("CameraLink_1", 0))
   Dim success As Boolean = acqDevice.Create()

   ' Get the number of features provided by the camera
   Dim featureCount As Integer = acqDevice.FeatureCount

   ' Create an empty feature object (to receive information),
```

```
' and create resources
Dim feature As SapFeature = _
   New SapFeature(New SapLocation("CameraLink_1", 0))
success = feature.Create()

'
' Example 1 : Browse through the feature list
'
For featureIndex As Integer = 0 To featureCount - 1
   ' Get information from current feature
   ' Get feature object
   success = acqDevice.GetFeatureInfo(featureIndex, feature)

   ' Extract name and type from object
   Dim featureName As String = feature.Name
   Dim featureDataType As SapFeature.Type = feature.DataType
   Dim featureAccessMode As SapFeature.AccessMode = feature.DataAccessMode

   ' Get/set value from/to current feature
   Select Case featureDataType
      ' Feature is a 64-bit integer
      Case SapFeature.Type.Int64
         Dim localFeatureValue As Long
         If featureAccessMode = SapFeature.AccessMode.RW Then
            success = acqDevice.GetFeatureValue( _
               featureIndex, localFeatureValue)
            localFeatureValue += 10
            success = acqDevice.SetFeatureValue( _
               featureIndex, localFeatureValue)
         End If

      ' Feature is a boolean
      Case SapFeature.Type.Bool
         Dim localFeatureValue As Boolean
         If featureAccessMode = SapFeature.AccessMode.RW Then
            success = acqDevice.GetFeatureValue( _
               featureIndex, localFeatureValue)
            localFeatureValue = Not localFeatureValue
            success = acqDevice.SetFeatureValue( _
               featureIndex, localFeatureValue)
         End If

      Case Else
         ' Other feature types
         ' ...
   End Select
Next

'
' Example 2 : Access specific feature (integer example)
'
' Get feature object
success = acqDevice.GetFeatureInfo("Gain", feature)

' Extract minimum, maximum and increment values
Dim featureValueMin As Integer
Dim featureValueMax As Integer
Dim featureValueIncrement As Integer

success = feature.GetValueMin(featureValueMin)
success = feature.GetValueMax(featureValueMax)
success = feature.GetValueIncrement(featureValueIncrement)

' Read, modify and write value
Dim featureValue As Integer
success = acqDevice.GetFeatureValue("Gain", featureValue)
featureValue = featureValue + 10
success = acqDevice.SetFeatureValue("Gain", featureValue)

'
' Example 3 : Access specific feature (enumeration example)
'
' Get feature object
success = acqDevice.GetFeatureInfo("DeviceScanType", feature)

' Get number of items in enumeration
Dim featureEnumCount As Integer = feature.EnumCount
```

```
   ' Get all enumeration strings and values
   Dim featureEnumText() As String = feature.EnumText
   Dim featureEnumValues() As Integer = feature.EnumValues

   ' Get individual enumeration strings and values
   Dim enumText As String = Nothing
   Dim enumValue As Integer

   For featureEnumIndex As Integer = 0 To featureEnumCount - 1
      enumText = featureEnumText(featureEnumIndex)
      enumValue = featureEnumValues(featureEnumIndex)
   Next

   ' Read a value and get its associated string
   success = acqDevice.GetFeatureValue("DeviceScanType", enumValue)
   success = feature.GetEnumTextFromValue(enumValue, enumText)

   ' Write a value corresponding to known string
   success = feature.GetEnumValueFromText("Linescan", enumValue)
   success = acqDevice.SetFeatureValue("DeviceScanType", enumValue)

   '
   ' Example 4 : Callback management
   '
   ' Get all event names
   Dim eventNames() As String = acqDevice.EventNames

   ' Browse event list
   Dim numEvents As Integer = acqDevice.EventCount

   For eventIndex As Integer = 0 To numEvents - 1
      Dim eventName As String = eventNames(eventIndex)
   Next

   ' Enable event by name
   success = acqDevice.EnableEvent("Feature Value Changed")
   AddHandler acqDevice.AcqDeviceNotify, AddressOf AcqDevice_AcqDeviceNotify

   ' Modified a feature (will trigger an event)
   success = acqDevice.SetFeatureValue("Gain", 80)

   ' Disable event by name
   RemoveHandler acqDevice.AcqDeviceNotify, _
      AddressOf AcqDevice_AcqDeviceNotify

   ' Release resources for all objects
   success = lut.Destroy()
   success = feature.Destroy()
   success = acqDevice.Destroy()

   ' Free all objects
   lut.Dispose()
   feature.Dispose()
   acqDevice.Dispose()
End Sub
```

Writing Feature Values by Group

When a series of features are tightly coupled, they are difficult to modify without following a specific order. For example, a region-of-interest (ROI) has four values (OffsetX, OffsetY, Width and Height) that are inter-dependent and must be defined as a group. To solve this problem, the SapAcqDevice class allows you to temporarily set the feature values in an “internal cache” and then download the values to the camera at the same time. The following code samples illustrate this process using an ROI example.

Sample Code for C#

```
   // Set manual mode to update features
   acqDevice.UpdateMode = SapAcqDevice.UpdateFeatureMode.Manual;

   // Set buffer left position (in the internal cache only)
   success = acqDevice.SetFeatureValue("OffsetX", 50);

   // Set buffer top position (in the internal cache only)
```

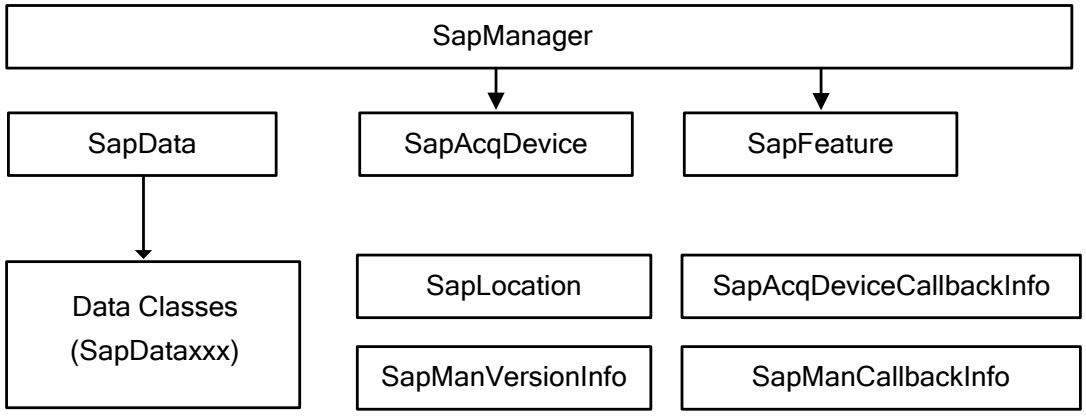
```
success = acqDevice.SetFeatureValue("OffsetY", 50);

// Set buffer width (in the internal cache only)
success = acqDevice.SetFeatureValue("Width", 300);

// Set buffer height (in the internal cache only)
success = acqDevice.SetFeatureValue("Height", 300);

// Write features to device (by reading values from the internal cache)
success = acqDevice.UpdateFeaturesToDevice();

// Set back the automatic mode
acqDevice.UpdateMode = SapAcqDevice.UpdateFeatureMode.Auto;
```

Equivalent Code for Visual Basic .NET

```
' Set manual mode to update features
acqDevice.UpdateMode = SapAcqDevice.UpdateFeatureMode.Manual

' Set buffer left position (in the internal cache only)
success = acqDevice.SetFeatureValue("OffsetX", 50)

' Set buffer top position (in the internal cache only)
success = acqDevice.SetFeatureValue("OffsetY", 50)

' Set buffer width (in the internal cache only)
success = acqDevice.SetFeatureValue("Width", 300)

' Set buffer height (in the internal cache only)
success = acqDevice.SetFeatureValue("Height", 300)

' Write features to device (by reading values from the internal cache)
success = acqDevice.UpdateFeaturesToDevice()

' Set back the automatic mode
acqDevice.UpdateMode = SapAcqDevice.UpdateFeatureMode.Auto
```

Using Sapera

Sapera++ Basic Class Hierarchy Chart



Header Files, Libraries, and DLLs

The following files are provided with Sapera Camera SDK. Also, 'XX' refers to the current Sapera Camera SDK version number, for example, **SapClassBasic73.dll** for the version 7.30 Basic Classes DLL.

File Name	Description	Location
SapClassBasic.h	Basic class header file	**Sapera\Classes\Basic**
SapClassBasic.lib	Basic class libraries for all Visual C++ versions	**Sapera\Lib\Win32** **Sapera\Lib\Win64**
SapClassBasicXX.dll	Basic class DLL	**<windir>\System32**

Sapera++ – Creating an Application

The following sections describe how to create a Sapera++ application in Visual C++ 2005/2008/2010/2012/2013/2015.

Visual Studio 2005/2008/2010/2012/2013/2015

Follow the steps below to compile and link a 32-bit or 64-bit application that uses the *Basic Classes*:

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

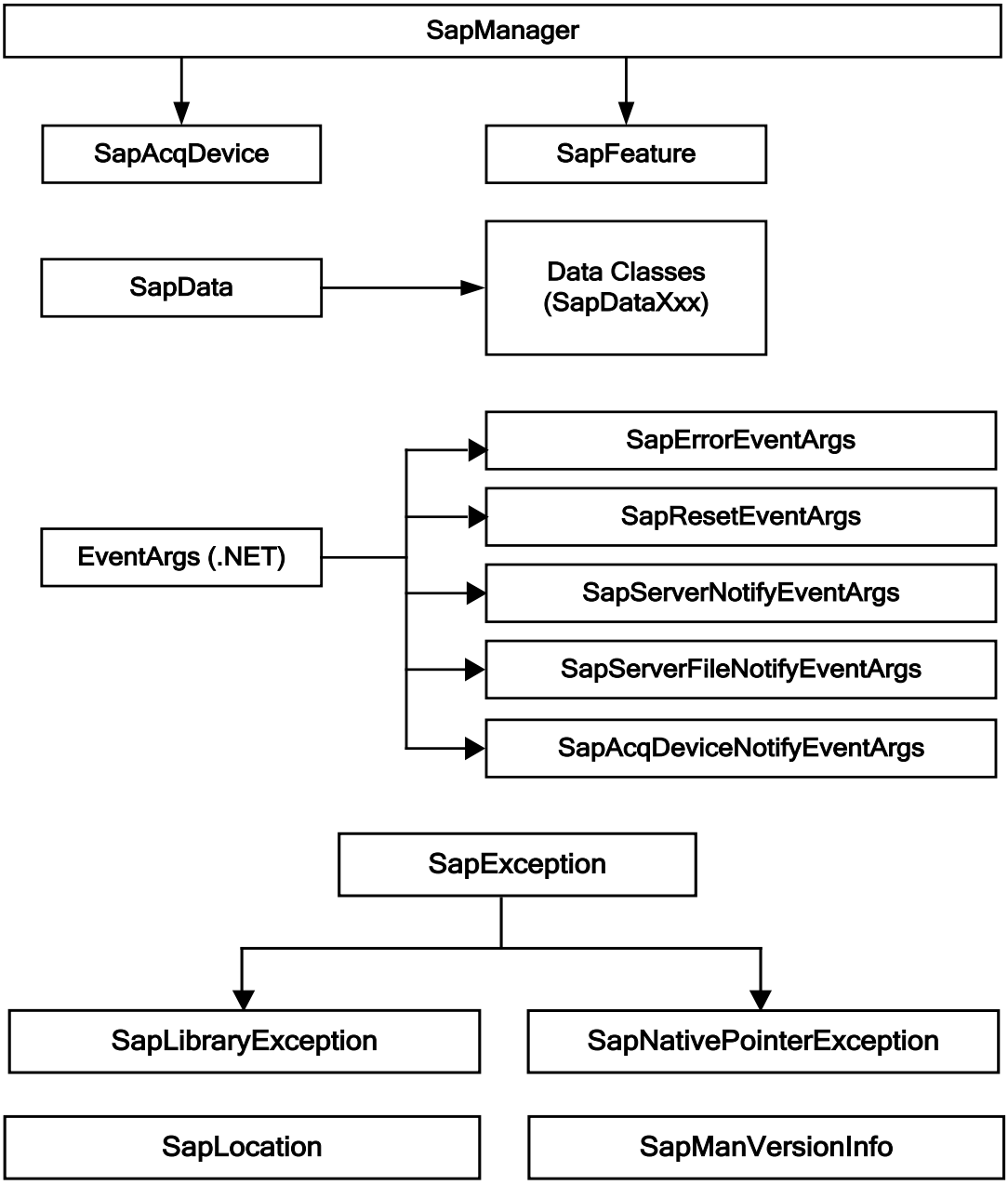
Updating Existing Visual Studio Projects

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

- Open the newer version of Visual Studio
- Open the existing solution (or workspace) file with the project(s) to convert from the older version
- Follow the instructions for converting the old projects to the new project format
- Review the warnings (if any) listed in the conversion report, you may need to make changes to project properties as a result
- If the projects only target 32-bit Windows (Win32), you will need to add 64-bit targets (x64) if necessary
- Compile the converted projects, you may need to fix compiler/linker errors and warnings

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

Sapera .NET Basic Class Hierarchy Chart



Sapera .NET – Creating an Application

The instructions below describe how to create a Sapera .NET application using C#, Visual Basic .NET, or C++ in Visual Studio 2005/2008/2010/2012/2013/2015.

Sapera .NET DLLs

The following files are provided with Sapera LT.

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

Using C#

Follow the steps below to create a Sapera .NET application using C#:

- From the main menu select **File • New • Project.**
- (**Visual Studio 2005/2008**) In the New Project dialog select **Other Languages • Visual C# • Windows** in the Project types pane.
- (**Visual Studio 2010/2012/2013/2015**) In the New Project dialog select **Visual C# • Windows** in the **Installed Templates** pane.
- (**Visual Studio 2005/2008**) In the Templates pane select **Windows Application** or **Console Application**.
- (**Visual Studio 2010/2012/2013/2015**) In the Templates pane select **Windows Forms Application** or **Console Application**.
- After the project has been successfully created go to Solution Explorer. Right-click on the References category and select **Add Reference**.
- In the Add References dialog select **DALSA.SaperaLT.SapClassBasic** in the .NET tab. If this entry is not present, use the Browse tab instead and select **DALSA.SaperaLT.SapClassBasic.dll** in the **Sapera\Components\NET\Bin** directory.
- (**Visual Studio 2010/2012/2013/2015**) Add to the project a file called **app.config** with the following contents:

  ```
  <?xml version="1.0"?>
  <configuration>
    <startup useLegacyV2RuntimeActivationPolicy="true" />
  </configuration>
  ```

  If this file already exists, just add the "useLegacyV2RuntimeActivationPolicy" contents to the "startup" section.
- In each source file which needs to access the Sapera .NET classes add the following statement:

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

Using Visual Basic .NET

Follow the steps below to create a Sapera .NET application using Visual Basic:

- From the main menu select **File • New • Project.**
- (**Visual Studio 2005/2008**) In the New Project dialog select **Other Languages • Visual Basic • Windows** in the Project types pane.
- (**Visual Studio 2010/2012/2013/2015**) In the New Project dialog select **Visual Basic • Windows** in the **Installed Templates** pane.
- (**Visual Studio 2005/2008**) In the Templates pane select **Windows Application** or **Console Application**.
- (**Visual Studio 2010/2012/2013/2015**) In the Templates pane select **Windows Forms Application** or **Console Application**.
- After the project has been successfully created go to Solution Explorer. Right-click on the project name and select **Add Reference**.
- In the Add References dialog select **DALSA.SaperaLT.SapClassBasic** in the .NET tab. If this entry is not present, use the Browse tab instead and select **DALSA.SaperaLT.SapClassBasic.dll** in the **Sapera\Components\NET\Bin** directory.
- Invoke the Project Properties dialog and select the References category. In the list below the Imported Namespaces heading check the **DALSA.SaperaLT.SapClassBasic** entry.
- (**Visual Studio 2010/2012/2013/2015**) Add to the project a file called **app.config** with the following contents:

  ```
  <?xml version="1.0"?>
  <configuration>
     <startup useLegacyV2RuntimeActivationPolicy="true" />
  </configuration>
  ```

  If this file already exists, just add the "useLegacyV2RuntimeActivationPolicy" contents to the "startup" section.

Notes on the Sapera LT GenICam Implementation

The following functions have GenICam specific notes about their implementation:

SapAcqDevice.UpdateMode Property (.NET) SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode (C++)	Only the *UpdateFeatureAuto* mode is implemented. Therefore, the SapAcqDevice::UpdateFeaturesFromDevice SapAcqDevice.UpdateFeaturesFromDevice Method SapAcqDevice.UpdateFeaturesToDevice Method SapAcqDevice::UpdateFeaturesToDevice functions are not implemented.
SapFeature::GetPollingTime SapFeature.PollingTime Property	GenICam does not provide polling information to the user, therefore this function property always returns 0.
SapFeature.SavedToConfigFile Property (.NET) SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile (C++)	The SapFeature class provides functions properties to control which features are saved to the device configuration file. In GenICam, this is hardcoded by the device manufacturer in the device description file. Therefore, the SapFeature.SavedToConfigFile PropertySapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile has no effect, and returns False when the value is read.
SapFeature class	The retrieval of feature enumeration properties is currently not implemented; only the name and value can be retrieved.

Events

The SapAcqDevice object always provides two events; "*Feature Info Changed*" and "*Feature Value Changed*". These events are related to feature state changes and not the device. Since GenICam does not give information on what changed in the feature, only "Feature Info Changed" events are generated; the "Feature Value Changed" is never generated.

Type

GenAPI interface mapping to SapFeature types.

GenICam Interface	Sapera Type
IInteger	SapFeature::TypeInt64
IFloat	SapFeature::TypeDouble
IString	SapFeature::TypeString
IEnummeration	SapFeature::TypeEnum
ICommand	SapFeature::TypeBool (write only)
IBoolean	SapFeature::TypeBool
IRegister	SapFeature::TypeArray
ICategory	Not exported; the category is a property of the feature.
ISelector	The selector is a property of the feature regardless of its type.
IPort	This is the interface to the underlying transport technologies; it is not exported to the user.

You can retrieve the type of a feature using the SapFeature::GetType function (C++) or SapFeature.DataType Property (.NET). If the type returned is TypeArray, reading /writing to this feature must use a SapBuffer.

Currently the ICommand is mapped to a SapFeature::TypeBool. Setting any value will execute that action and return when the action is complete. One limitation of this mapping is that if the action takes more than the Sapera timeout, setting the value might return false even if the action succeeded.

Sapera LT Utilities

Sapera LT Camera SDK includes the following utilities that can be used to monitor Sapera LT hardware and software events, error messages, as well as frame grabber configuration and diagnostics:

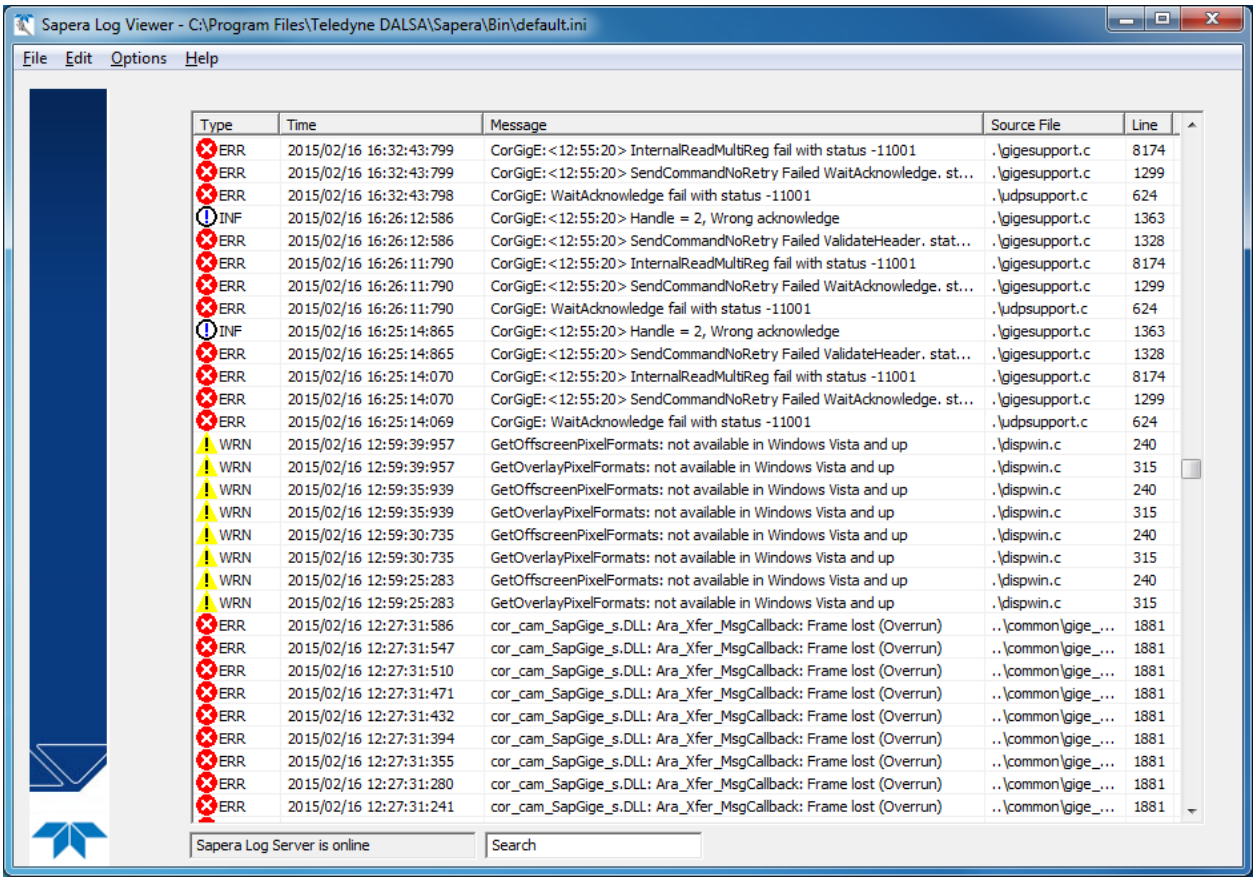
Sapera Log Viewer	Displays error and other messages generated by Sapera LT applications and Teledyne DALSA hardware.
Sapera Configuration	Configuration utility for Teledyne DALSA frame grabbers.
Sapera PCI Diagnostics	Low-level diagnostic utility for Teledyne DALSA frame grabbers.
Sapera Color Calibration Tool	Generates color correction coefficients (if supported) that are used to adjust the camera sensor's color response.

Sapera Log Viewer

The Sapera Log Viewer utility program included with Sapera LT provides an easy way to view error and other types of messages generated by Sapera LT applications and Teledyne DALSA hardware, such as cameras and frame grabbers. Typically, the Sapera Log Viewer application is used by technical support to troubleshoot software and hardware problems.

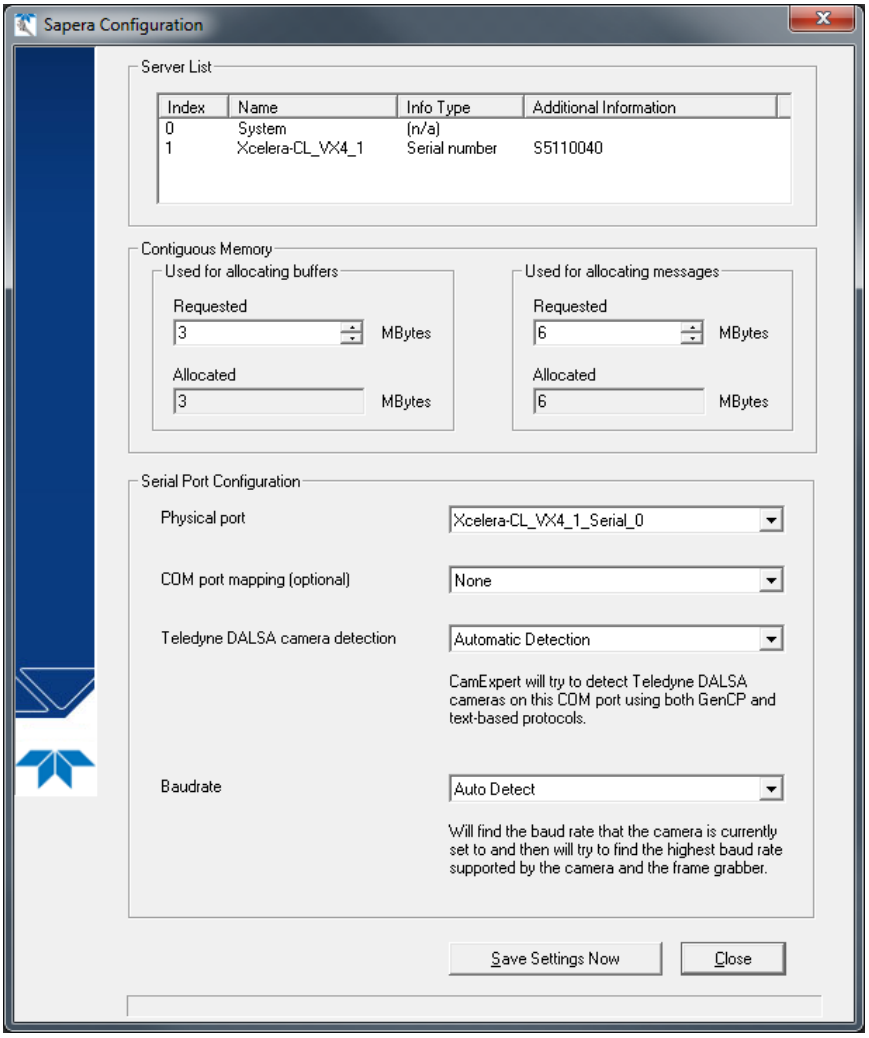
During development it is recommended to start the Sapera Log Viewer before your application and then let it run so it can be referred to any time a detailed error description is required. However, errors are also stored by a low-level service (running in the background), even if the utility is not running. Therefore it is possible to run it only when a problem occurs with your application.

Refer to the utility's online help for more information on using the Sapera Log Viewer.



Sapera Configuration Utility

The **Sapera Configuration** program (**SapConf.exe)** allows you to see all the Sapera LT-compatible devices present within your system, together with their respective serial numbers. It can also adjust the amount of contiguous memory to be allocated at boot-time or map serial ports. After activating this program, it displays all the servers related to the installed devices as shown in the figure below (64-bit version shown).



- The **System** entry represents the system server. It corresponds to the host machine (your computer), and is the only server that should always be present. The other servers correspond to the device's present within the system.
- The **Serial Port Configuration** section allows you to select the frame grabber's serial port, map COM ports, specify the type of Teledyne DALSA camera detection, as well as configure the baud rate.

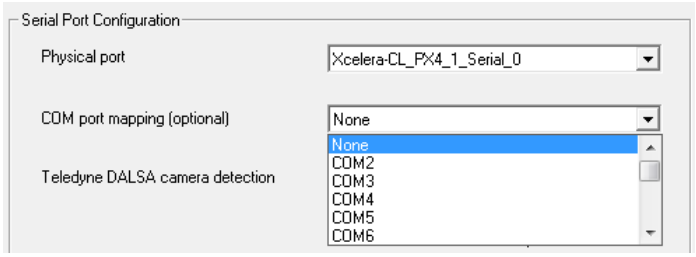
Configuring Frame Grabber Board Serial Ports

Certain frame grabber boards provide an onboard serial port for direct camera control by the frame grabber. Refer to the specific board user manual for information on how to configure and use it.

Serial Port Mapping

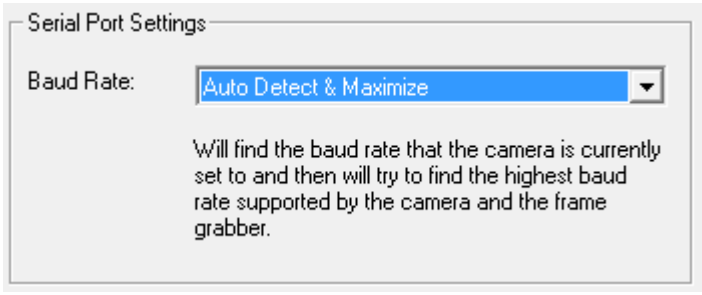
For certain 3rd party applications, it may be necessary to map frame grabber serial ports to Windows COM ports. To do so, run the Sapera Configuration utility, select the frame grabber serial port

connected to the camera, and set the **Serial Port Configuration > COM port mapping** parameter. For new mappings to take effect, a system reboot is generally required.

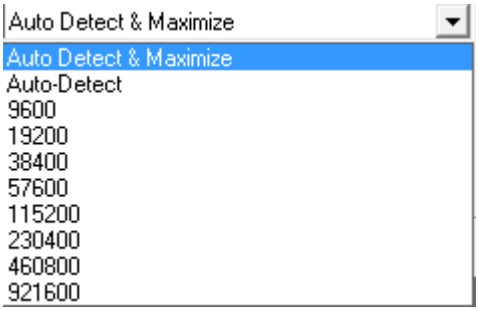


Baud Rate

By default, the baud rate is set to **Auto Detect & Maximize**, which will find the baud rate that the camera is currently set to and then find the highest baud rate supported by both the camera and the frame grabber.



Otherwise, the baud rate can be specifically chosen from among the options available in the drop-down list.



Configuring Contiguous Memory

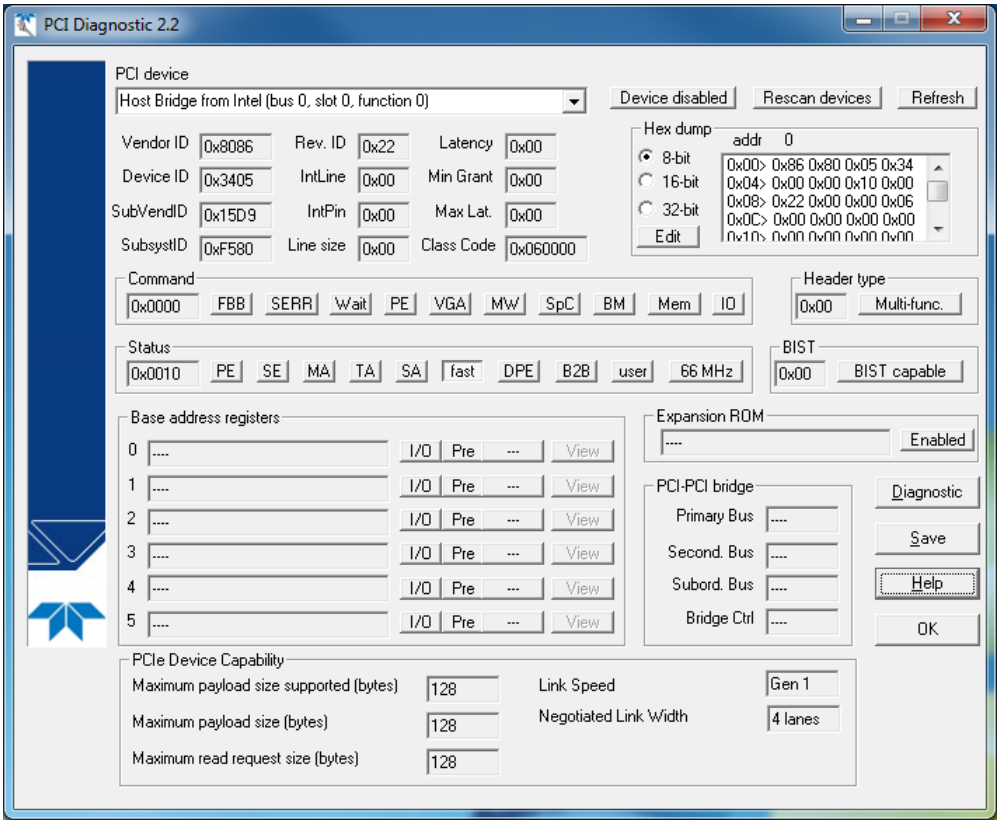
The **Contiguous Memory** section lets you specify the total amount of contiguous memory to be reserved for allocating **buffers** and **messages**.

	Note: All Sapera LT demos and examples **do not use contiguous memory**. Therefore, you should not modify these settings unless your application requires contiguous memory.

The **Requested** value displays what was requested. The **Allocated** value displays the amount of contiguous memory that was allocated successfully. The current value for **buffers** determines the total amount of contiguous memory reserved at boot-time for the allocation of dynamic resources (for example, buffers, lookup tables, kernels). Adjust this value according to the need of your application for contiguous memory. The current value for **messages** determines the total amount of contiguous memory reserved at boot-time for the allocation of messages. This memory space is used to store arguments when a Sapera LT function is called. Increase this value if you are using functions with large arguments, such as arrays.

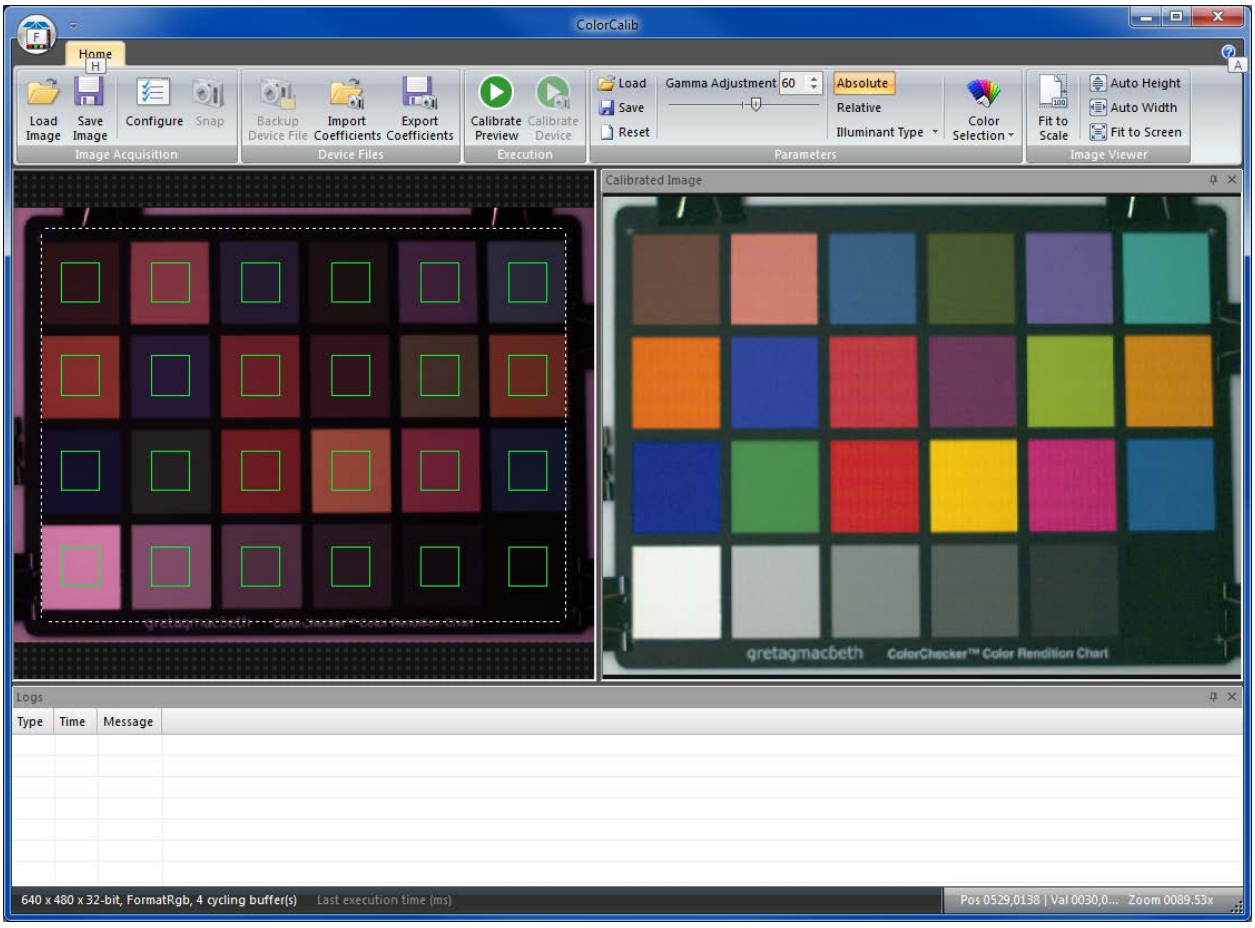
PCI Diagnostic

The PCI Diagnostic tool is used for debugging frame grabber hardware issues. PCI Diagnostic reads the content of the PCI configuration space and detects memory and I/O conflicts between PCI devices. Use it to verify the integrity of your system before and after installing a new PCI device. Refer to the utility's online help for more information on using the PCI Diagnostic utility.



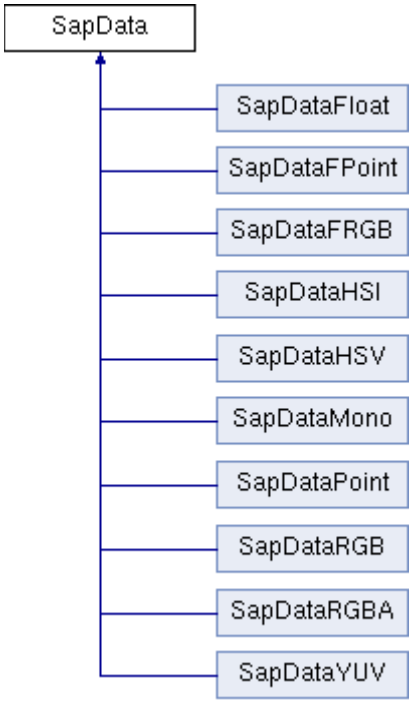
Sapera Color Calibration Tool

The Sapera LT Color Tool allows you to generate color correction coefficients used to adjust the camera sensor's color response for known colors at a specific illumination. This ensures that the camera outputs the correct color for a given scene.
The color correction coefficients are saved as a **.ccor* file that can be uploaded to the camera to perform real-time correction using the camera's hardware (if supported, instead of performing correction on the host computer after the image transfer).



Sapera++ Basic Class Reference

Data Classes



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

SapData Class

Purpose

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

void **Clear**();

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

SapFormatType **GetType**();

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

Demo/Example Usage

Not available

SapDataFRGB Class

Purpose
Encapsulates one element supporting Sapera floating-point RGB data types

SapDataFRGB();
SapDataFRGB(float *red*, float *green*, float *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specifies an initial value other than black

float **Red**();
Returns the red component of the current value of the data element

float **Green**();
Returns the green component of the current value of the data element

float **Blue**();
Returns the blue component of the current value of the data element

void **Set**(float *red*, float *green*, float *blue*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataHSI Class

Purpose
Encapsulates one element supporting Sapera HSI data types

SapDataHSI();
SapDataHSI(int *h*, int *s*, int *i*);
Class constructor, where the *h*, *s*, and *i* arguments specify an initial value other than black

int **H**();
Returns the H component of the current value of the data element

int **S**();
Returns the S component of the current value of the data element

int **I**();
Returns the I component of the current value of the data element

void **Set**(int *h*, int *s*, int *i*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataHSV Class

Purpose
Encapsulates one element supporting Sapera HSV data types

SapDataHSV();
SapDataHSV(int *h*, int *s*, int *v*);
Class constructor, where the *h*, *s*, and *v* arguments specify an initial value other than black

int **H**();
Returns the H component of the current value of the data element

int **S**();
Returns the S component of the current value of the data element

int **V**();
Returns the V component of the current value of the data element

void **Set**(int *h*, int *s*, int *v*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataFloat Class

Purpose
Encapsulates one element supporting Sapera floating-point data types

SapDataFloat();
SapDataFloat(float *flt*);
Class constructor, where the *flt* argument specifies an initial value other than black

int **Float**();
Returns the current value of the data element

void **Set**(float *flt*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataFPoint Class

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

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

float **X(**);
Returns the X component of the current value of the data element

float **Y**();
Returns the Y component of the current value of the data element

void **Set**(float *x*, float *y*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataMono Class

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

SapDataMono();
SapDataMono(int *mono*);
Class constructor, where the *mono* argument specifies an initial value other than black

int **Mono**();
Returns the current value of the data element

void **Set**(int *mono*);
Specifies a new value for the data element

Demo/Example Usage

Example Common Utiltities

SapDataPoint Class

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

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

int **X()**;
Returns the X component of the current value of the data element

int **Y**();
Returns the Y component of the current value of the data element

void **Set**(int *x*, int *y*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataRGB Class

Purpose
Encapsulates one element supporting Sapera RGB data types

SapDataRGB();
SapDataRGB(int *red*, int *green*, int *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specify an initial value other than black

int **Red**();
Returns the red component of the current value of the data element

int **Green**();
Returns the green component of the current value of the data element

int **Blue**();
Returns the blue component of the current value of the data element

void **Set**(int *red*, int *green*, int *blue*);
Specifies a new value for the data element

Demo/Example Usage

Example Common Utiltities

SapDataRGBA Class

Purpose
Encapsulates one element supporting Sapera RGB with alpha channel data types

SapDataRGBA();SapDataRGBA(int *red*, int *green*, int *blue*, int *alpha*);
Class constructor, where the *red*, *green*, *blue* and *alpha* arguments specify an initial value other than black

int Red();
Returns the red component of the current value of the data element

int Green();
Returns the green component of the current value of the data element

int Blue();
Returns the blue component of the current value of the data element

int Alpha();
Returns the alpha component of the current value of the data element

void Set(int *red*, int *green*, int *blue*, int *alpha*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapDataYUV Class

Purpose
Encapsulates one element supporting Sapera YUV data types

SapDataYUV();
SapDataYUV(int *y*, int *u*, int *v*);
Class constructor, where the *y*, *u*, and *v* arguments specify an initial value other than black

int **Y**();
Returns the Y component of the current value of the data element

int **U**();
Returns the U component of the current value of the data element

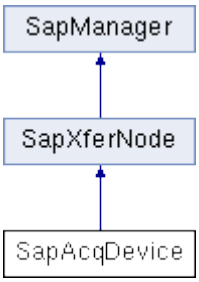
int **V**();
Returns the V component of the current value of the data element

void **Set**(int *y*, int *u*, int *v*);
Specifies a new value for the data element

Demo/Example Usage

Not available

SapAcqDevice



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

#include <SapClassBasic.h>

Construction

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

General Parameters

GetConfigFile,	Gets/sets the name of the acquisition configuration file (CCF)
SetConfigFile	
GetReadOnly,	Gets/sets whether or not the class has read-only access to the device
SetReadOnly	
GetUpdateFeatureMode,	Gets/sets the mode by which features are written to the device
SetUpdateFeatureMode	
GetLabel	Gets a text description of the acquisition device
GetConfigName,	Gets/sets the configuration name to be used when saving the device features using the *SaveFeatures* method
SetConfigName	
GetModeName,	Gets/sets the mode name to be used when saving the device features using the *SaveFeatures* method
SetModeName	
UpdateLabel	Updates the device label.

Parameter Access

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,	Gets/sets the value of a low-level Sapera C library parameter
SetParameter	

Feature Access

GetFeatureCount	Returns the number of features supported by the acquisition device
GetFeatureNameByIndex	Returns the name of a feature associated with a specified index
GetFeatureIndexByName	Returns the index of a feature associated with a specified name
IsFeatureAvailable	Returns whether or not a feature is supported by the acquisition device

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

SapAcqDevice Class Members SapAcqDevice Member Functions

The following are members of the SapAcqDevice Class.

SapAcqDevice::SapAcqDevice

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

Parameters

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

Remarks

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

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

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

The SapAcqDevice object is used only for storing the acquisition device parameters.

Demo/Example Usage

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

SapAcqDevice::Create

BOOL **Create**();

Return Value

Returns TRUE if successful, FALSE otherwise.

Remarks

Creates all the low-level Sapera resources needed by the acquisition object.

Demo/Example Usage

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

SapAcqDevice::DeleteDeviceFile

BOOL **DeleteDeviceFile**(const char ***deviceFileName*);
BOOL **DeleteDeviceFile**(int *deviceFileIndex*);

Parameters

deviceFileName	Name of the device file. See the acquisition device User's Manual for the list of supported files.
deviceFileIndex	Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.

Return Value

Returns TRUE if successful, FALSE otherwise.

Remarks

Deletes the specified file on the device.

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

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::Destroy

BOOL **Destroy**();

Return Value

Returns TRUE if successful, FALSE otherwise.

Remarks

Destroys all the low-level Sapera resources needed by the acquisition object.

Demo/Example Usage

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

SapAcqDevice::GetCapability

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

Parameters

cap Low-level Sapera C library capability to read

pValue Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::GetCategoryCount

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

Parameters

categoryCount Number of feature categories available on the acquisition device

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Demo/Example Usage

Not available

SapAcqDevice::GetCategoryPath

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

Parameters

categoryIndex	Index of the category. All indices from 0 to the value returned by the GetCategoryCount function, minus 1, are valid.
path	Returns the full path name of the category associated with the specified index
pathSize	Size (in bytes) of the buffer pointed to by *path*

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

All path names begin with “\Root” or “\SaperaRoot”

Top level categories are returned as “\Root\CategoryName”

Second level categories are returned as “\Root\CategoryName\SubCategoryName”

and so on…

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

Demo/Example Usage

Not available

SapAcqDevice::GetConfigFile, SapAcqDevice::SetConfigFile

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

Parameters

configFile	Name of the configuration file

Remarks

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

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

You can only call SetConfigFile before the Create method.

Demo/Example Usage

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

SapAcqDevice::GetConfigName, SapAcqDevice::SetConfigName

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

Parameters

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

Remarks

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

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

Demo/Example Usage

Not available

SapAcqDevice::GetEventCount

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

Parameters

eventCount Number of events supported by the acquisition device

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Events Example

SapAcqDevice::GetEventIndexByName

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

Parameters

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

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice::GetEventNameByIndex

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

Parameters

eventIndex	Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.
eventName	Returns the name of the event associated with the specified index
eventNameSize	Size (in bytes) of the buffer pointed to by *eventName*

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Events Example

SapAcqDevice::GetFeatureCount

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

Parameters

featureCount	Number of features supported by the acquisition device

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice::GetFeatureIndexByName

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

Parameters

featureName	Name of the feature. See the acquisition device User's Manual for the list of supported features.
featureIndex	Returns the index of the feature associated with the specified name

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice::GetFeatureInfo

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

BOOL **GetFeatureInfo**(int *featureIndex*, SapFeature* *feature*);

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

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

---

SapAcqDevice::GetFeatureNameByIndex

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Features Example

SapAcqDevice::GetFeatureValue

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

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

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

Demo/Example Usage

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

SapAcqDevice::GetFileCount

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

Parameters

fileCount Number of files supported by the acquisition device

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::GetFileIndexByName

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

Parameters

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

fileIndex Returned index of the device file associated with the specified name

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice::GetFileNameByIndex

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

Parameters

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

fileName Returned name of the device file associated with the specified index

fileSize Size (in bytes) of the buffer pointed to by *fileName*

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::GetFileProperty

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

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::GetLabel

const char* **GetLabel**();

Remarks

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

Example: "Genie_HM1400_1 [UserName]"

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

Demo/Example Usage

Not available

SapAcqDevice::GetModeName, SapAcqDevice::SetModeName

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

Parameters

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

Remarks

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

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

Demo/Example Usage

Not available

SapAcqDevice::GetParameter, SapAcqDevice::SetParameter

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::GetReadOnly, SapAcqDevice::SetReadOnly

BOOL **GetReadOnly**();
BOOL **SetReadOnly**(BOOL *readOnly*);

Parameters

readOnly TRUE to force read-only access to the device

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode

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

Parameters

mode The mode can be one of the following values:

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

Remarks

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

Note, for devices not using the Network Imaging Package (GigE Vision Framework), only the SapAcqDevice::UpdateFeatureAuto mode is implemented; setting the update mode to SapAcqDevice::UpdateFeatureManual has no effect. Consequently, the SapAcqDevice::UpdateFeaturesFromDevice and SapAcqDevice::UpdateFeaturesToDevice functions are not implemented.

Demo/Example Usage

Not available

SapAcqDevice::IsCallbackRegistered

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

Parameters

eventName	Name of the event. See the acquisition device User's Manual for the list of supported events.
eventIndex	Index of the event. All indices in the range from 0 to the value returned by the GetEventCount method, minus 1, are valid.
isRegistered	Returns TRUE if a callback function was registered on this event. FALSE otherwise

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice::IsCapabilityValid

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

Parameters

cap	Low-level Sapera C library capability to be checked

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::IsEventAvailable

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

Parameters

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

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

GigE FlatField Demo

SapAcqDevice::IsFeatureAvailable

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

Parameters

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

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

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

SapAcqDevice::IsFileAccessAvailable

BOOL **IsFileAccessAvailable**();

Remarks

Gets availability of file access by the acquisition device. If this function returns FALSE, then you should not use the GetFileCount, GetFileNameByIndex, GetFileIndexByName, GetFileProperty, WriteFile, ReadFile, and DeleteDeviceFile functions.

Demo/Example Usage

Not available

SapAcqDevice::IsFlatFieldAvailable

BOOL **IsFlatFieldAvailable**();

Remarks

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

Demo/Example Usage

GigE FlatField Demo

SapAcqDevice::IsParameterValid

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

Parameters

param Low-level Sapera C library parameter to be checked

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::IsRawBayerOutput

BOOL **IsRawBayerOutput**();

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice::LoadFeatures

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::ReadFile

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

Parameters

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

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

localFilePath Full directory path and filename on the host computer to save the file.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::RegisterCallback

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

Parameters

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

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

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

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

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Example

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

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice::SaveFeatures

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

Demo/Example Usage

Not available

SapAcqDevice::SetFeatureValue

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

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

Parameters

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

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

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

featureString String feature to write

featureBuffer SapBuffer object to write

featureLut SapLut object to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

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

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

Demo/Example Usage

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

---

SapAcqDevice::UnregisterCallback

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Example

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice::UpdateFeaturesFromDevice

BOOL **UpdateFeaturesFromDevice**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets all the features from the acquisition device at once.

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::UpdateFeaturesToDevice

BOOL **UpdateFeaturesToDevice**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes all the features to the acquisition device at once.

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

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

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

Demo/Example Usage

Not available

SapAcqDevice::UpdateLabel

BOOL **UpdateLabel**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

Demo/Example Usage

Camera Files Example

SapAcqDevice::WriteFile

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Camera Files Example

SapAcqDeviceCallbackInfo

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

#include <SapClassBasic.h>

SapAcqDeviceCallbackInfo Class Members

Construction

SapAcqDeviceCallbackInfo	Class constructor

Attributes

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

SapAcqDeviceCallbackInfo Member Functions

The following are members of the SapAcqDeviceCallbackInfo Class.

SapAcqDeviceCallbackInfo::SapAcqDeviceCallbackInfo

SapAcqDeviceCallbackInfo(SapAcqDevice* *pAcqDevice*, void* c*ontext*, COREVENTINFO *eventInfo*);

Parameters

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

Remarks

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

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

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDeviceCallbackInfo::GetAcqDevice

SapAcqDevice* **GetAcqDevice**();

Remarks

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDeviceCallbackInfo::GetAuxiliaryTimeStamp

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

Parameters

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

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetContext

void ***GetContext**();

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetCustomData

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

Parameters

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

Remarks

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

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

Example

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

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetCustomSize

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

Parameters

customSize Address of an integer to return the value

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetEventCount

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

Parameters

eventCount Address of an integer where the count is written

Remarks

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

Demo/Example Usage

Camera Events Example

SapAcqDeviceCallbackInfo::GetEventIndex

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

Parameters

eventIndex Address of an integer where the event index is written

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetEventInfo

COREVENTINFO **GetEventInfo**();

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetFeatureIndex

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

Parameters

featureIndex Address of an integer where the feature index is written

Remarks

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDeviceCallbackInfo::GetGenericParam0
SapAcqDeviceCallbackInfo::GetGenericParam1
SapAcqDeviceCallbackInfo::GetGenericParam2
SapAcqDeviceCallbackInfo::GetGenericParam3

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

Parameters

paramValue Address of an integer where the parameter value is written

Remarks

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

Demo/Example Usage

Not available

SapAcqDeviceCallbackInfo::GetHostTimeStamp

BOOL **GetHostTimeStamp**(UINT64* *hostTimeStamp*);

Parameters

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

Remarks

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

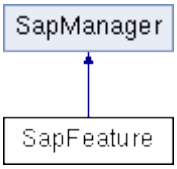
Under Windows, the value corresponding to the high-resolution performance counter is directly returned. Refer to the QueryPerformanceCounter and QueryPerformanceFrequency functions in the Windows API documentation for more details on how to convert this value to time units.

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

Demo/Example Usage

Not available

SapFeature



The purpose of the SapFeature class is to retrieve individual feature information from the SapAcqDevice class. Each feature supported by SapAcqDevice provides a set of capabilities such as name, type, access mode, and so forth, which can be obtained through SapFeature. The GetFeatureInfo method of SapAcqDevice gives access to this information.

#include <SapClassBasic.h>

SapFeature Class Members

Construction

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

Attributes

GetLocation	Gets/sets the location where the feature resource is located
SetLocation	

General Parameters

GetName	Gets the short name of the feature
GetType	Gets the data type of the feature
IsStandard	Checks if a feature is standard or custom
GetAccessMode	Gets the current data access mode for a feature
GetPollingTime	Gets the interval of time between two consecutive feature updates
GetToolTip	Gets the text which represents the explanation of the feature
GetDescription	Gets the text which represents the full description of the feature
GetDisplayName	Gets the descriptive name of the feature
GetFloatNotation	Gets the notation type to use to display a float type feature
GetFloatPrecision	Gets the number of decimal places to display for a float type feature
GetRepresentation	Gets the mathematical representation of a integer or float feature
GetSign	Checks if an integer/float feature is signed or not
GetSiUnit	Gets the physical units representing the feature in the international system (SI)
GetCategory	Gets the category to which the current feature belongs
GetWriteMode	Checks if a feature can be modified when the transfer object is connected and/or acquiring
IsSavedToConfigFile SetSavedToConfigFile	Checks if a feature is saved to a CCF configuration file
GetSiToNativeExp10	Gets the feature conversion factor from international system (SI) units to native units
GetVisibility	Gets the level of visibility assigned to a feature

GetArrayLength	Gets the number of bytes required for an array type feature
GetIncrementType	Gets the type of increment for an integer or floating-point feature
GetValidValueCount	Get the number of valid values for an integer or floating-point feature which defines them as a list
Integer/float-Parameters	
GetMin	Gets the minimum acceptable value for a feature
GetMax	Gets the maximum acceptable value for a feature
GetInc	Gets the minimum acceptable increment for an integer or a float feature
GetValidValue	Gets one of a predefined set of valid values for a feature
Enumeration-Parameters	
GetEnumCount	Get the number of possible values for a feature which belongs to an enumerated type
GetEnumString	Gets the string value at a specified index for the enumerated type corresponding to the current feature
GetEnumValue	Gets the integer value at a specified index for the enumerated type corresponding to the current feature
IsEnumEnabled	Checks if the enumeration value corresponding to a specified index is enabled
GetEnumStringFromValue	Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature
GetEnumValueFromString	Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature
Selector-Parameters	
IsSelector	Determines if the value of a feature directly affects other features
GetSelectedFeatureCount	Gets the number of features associated with a selector
GetSelectedFeatureIndex	Gets the index of a feature associated with a selector
GetSelectedFeatureName	Gets the name of a feature associated with a selector
GetSelectingFeatureCount	Gets the number of selectors associated with a feature
GetSelectingFeatureIndex	Gets the index of a selector associated with a feature
GetSelectingFeatureName	Gets the name of a selector associated with a feature

SapFeature Member Functions

The following are members of the SapFeature Class.

SapFeature::SapFeature

SapFeature(SapLocation *location* = SapLocation::ServerSystem);

Parameters

location SapLocation object specifying where the feature is located. This location must be the same as that of the SapAcqDevice object from which the feature is retrieved.

Remarks

The SapFeature constructor does not actually create the low-level Sapera resources. To do this, you must call the SapFeature::Create method. Upon creation the feature object contents are meaningless. To fill-in a feature object, call the SapAcqDevice::GetFeatureInfo method.

Demo/Example Usage

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

SapFeature::Create

BOOL **Create**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the feature object. Call this method before using the object as a parameter to the SapAcqDevice::GetFeatureInfo method.

Demo/Example Usage

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

SapFeature::Destroy

BOOL **Destroy**();

Return Value

Returns TRUE if successful, FALSE otherwise.

Remarks

Destroys all the low-level Sapera resources needed by the feature object.

Demo/Example Usage

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

SapFeature::GetAccessMode

BOOL **GetAccessMode**(SapFeature::AccessMode* *accessMode*);

Parameters

accessMode Returned data access mode can be one of the following values:

SapFeature::AccessUndefined	Undefined access mode
SapFeature::AccessRW	The feature may be read and written. Most features are of this type.
SapFeature::AccessRO	The feature can only be read.
SapFeature::AccessWO	The feature can only be written. This is the case for some features which represent commands (or actions) such as 'TimestampReset'.
SapFeature::AccessNP	The feature is not present. The feature is visible in the interface but is not implemented for this device.
SapFeature::AccessNE	The feature is present but currently not enabled. Often used when a feature depends on another feature's value.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the current data access mode for a feature.

Demo/Example Usage

Camera Features Example, Camera Files Example

SapFeature::GetArrayLength

BOOL **GetArrayLength**(int* *arrayLength*);

Parameters

arrayLength Returned array length (in bytes).

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the number of bytes required to store the value of a feature of array type, that is, when the value returned by the GetType method is SapFeature::TypeArray.

Demo/Example Usage

Not available

SapFeature::GetCategory

BOOL **GetCategory**(char* *category*, int *categorySize*);

Parameters

category Buffer for the returned text string. Must be large enough for 64 characters.

categorySize Size of the buffer pointed to by *category* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the category to which the current feature belongs. To simplify the classification of a large set of features from the same SapAcqDevice object, the features are divided into categories. These categories are useful for presenting a list of features in a graphical user interface.

Demo/Example Usage

Not available

SapFeature::GetDescription

BOOL **GetDescription**(char* *description*, int *descriptionSize*);

Parameters

description Buffer for the returned text string. Must be large enough for 512 characters.

descriptionSize Size of the buffer pointed to by *description* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the text which represents the full description of the feature. This information can be used to display detailed textual information in a graphical user interface.

Demo/Example Usage

Not available

SapFeature::GetDisplayName

BOOL **GetDisplayName**(char* *displayName*, int *displayNameSize*);

Parameters

displayName Buffer for the returned text string. Must be large enough for 64 characters.

displayNameSize Size of the buffer pointed to by *displayName* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the descriptive name of the feature. This name can be used for listing features in a graphical user interface.

Demo/Example Usage

Camera Features Example

SapFeature::GetEnumCount

BOOL **GetEnumCount**(int* *enumCount*);

Parameters

enumCount Returned number of enumeration items

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the number of possible values for a feature which belongs to an enumerated type. Use this method along with the GetEnumString and GetEnumValue methods to enumerate all the items contained within an enumeration feature.

Demo/Example Usage

Camera Features Example

SapFeature::GetEnumString

BOOL **GetEnumString**(int *enumIndex*, char* *enumString*, int *enumStringSize*);

Parameters

enumIndex Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)

enumString Buffer for the returned text string.

enumStringSize Size of the buffer pointed to by *enumString* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the string value at a specified index for the enumerated type corresponding to the current feature. Use this method along with the GetEnumCount and GetEnumValue methods to enumerate all the items contained within an enumeration feature.

Demo/Example Usage

Camera Features Example

SapFeature::GetEnumStringFromValue

BOOL **GetEnumStringFromValue**(int *enumValue*, char* *enumString*, int *enumStringSize*);

Parameters

enumValue Value to look for in the enumeration items

enumString Buffer for the returned text string

enumStringSize Size of the buffer pointed to by *enumString* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature. For example you may use this method to retrieve the string corresponding to an enumeration value returned by the SapAcqDevice::GetFeatureValue method.

Demo/Example Usage

Camera Features Example

SapFeature::GetEnumValue

BOOL **GetEnumValue**(int *enumIndex*, int* *enumValue*);

Parameters

enumIndex Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)

enumValue Returns enumeration value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the integer value at a specified index for the enumerated type corresponding to the current feature. Use this method along with the GetEnumCount and GetEnumString methods to enumerate all the items contained within an enumeration feature.

Demo/Example Usage

Not available

SapFeature::GetEnumValueFromString

BOOL **GetEnumValueFromString**(const char* *enumString*, int* *enumValue*);

Parameters

enumString Text string to look for in the enumeration

enumValue Returned integer value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature. For example you may use this method to retrieve the value corresponding to a known enumeration string before calling the SapAcqDevice::SetFeatureValue method.

Demo/Example Usage

Not available

SapFeature::GetFloatNotation

BOOL **GetFloatNotation**(FloatNotation**notation*);

Parameters

notation Specifies how the float type feature is displayed. Possible values are:

SapFeature::FloatNotationFixed	Display variable using fixed notation. For exampe, 123.4
SapFeature::FloatNotationScientific	Display variable using scientific notation. For example, 1.234e-2.
SapFeature::FloatNotationUndefined	Undefined.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the type of notation to use to display a float type feature.

Demo/Example Usage

Not available

SapFeature::GetFloatPrecision

BOOL **GetFloatPrecision**(int64 **precision*);

Parameters

precision Number of decimal places of a float type feature to display.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the number of decimal places to display for a float type feature.

Demo/Example Usage

Not available

SapFeature::GetInc

BOOL **GetInc**(INT32* *incValue*);
BOOL **GetInc**(UINT32* *incValue*);
BOOL **GetInc**(INT64* *incValue*);
BOOL **GetInc**(UINT64* *incValue*);
BOOL **GetInc**(float* *incValue*);
BOOL **GetInc**(double* *incValue*);

Parameters

incValue Returned increment value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the minimum acceptable increment for an integer or a float feature. Some features cannot vary by increments of 1. Their value must be a multiple of a certain increment. For example the buffer cropping dimensions might require to be a multiple of 4 in order to optimize the data transfer.

Demo/Example Usage

Not available

SapFeature::GetIncrementType

BOOL **GetIncrementType**(SapFeature::IncrementType* *incrementType*);

Parameters

incrementType Returned increment type can be one of the following values:

SapFeature::IncrementUndefined	Undefined increment type. This normally means that the acquisition device to which the feature is associated does not support reading the value of the increment type.
SapFeature::IncrementNone	The feature has no increment. Use the GetMin and GetMax functions to find out the feature value limits.
SapFeature::IncrementLinear	The feature has a fixed increment. Use the GetMin and GetMax functions to find the feature value limits, and GetInc to find the increment.
SapFeature::IncrementList	The feature has a fixed set of valid values. Use the GetValidValueCount function to find the number of values, and the GetValidValue function to enumerate them.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the type of increment for an integer or floating-point feature. This is useful for finding out which values are valid for this feature.

Demo/Example Usage

Camera Features Example

SapFeature::GetLocation, SapFeature::SetLocation

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

Remarks

Gets/sets the location where the feature resource is located. This location must be the same as that of the corresponding SapAcqDevice object. A specific location can also be specified through the SapFeature constructor.

You can only call SetLocation before the Create method.

Demo/Example Usage

Not available

SapFeature::GetMax

BOOL **GetMax**(INT32* *maxValue*);
BOOL **GetMax**(UINT32* *maxValue*);
BOOL **GetMax**(INT64* *maxValue*);
BOOL **GetMax**(UINT64* *maxValue*);
BOOL **GetMax**(float* *maxValue*);
BOOL **GetMax**(double* *maxValue*);

Parameters

maxValue Returned maximum value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the maximum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the UINT32 version to get the maximum length of the string (excluding the trailing null character).

Demo/Example Usage

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

SapFeature::GetMin

BOOL **GetMin**(INT32* *minValue*);
BOOL **GetMin**(UINT32* *minValue*);
BOOL **GetMin**(INT64* *minValue*);
BOOL **GetMin**(UINT64* *minValue*);
BOOL **GetMin**(float* *minValue*);
BOOL **GetMin**(double* *minValue*);

Parameters

minValue Returned minimum value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the minimum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the UINT32 version to get the minimum length of the string (excluding the trailing null character).

Demo/Example Usage

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

SapFeature::GetName

BOOL **GetName**(char* *name*, int *nameSize*);

Parameters

name Buffer for the returned text string. Must be large enough for 64 characters.

nameSize Size of the buffer pointed to by *name* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the short name of the feature. This name can be used with SapAcqDevice method which expect a feature name. This string should not be used for display in a graphical user interface. Use the GetDisplayName method instead to provide a more descriptive name.

Demo/Example Usage

Not available

SapFeature::GetPollingTime

BOOL **GetPollingTime**(int* *pollingTime*);

Parameters

pollingTime Returned polling time (in milliseconds).

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the interval of time between two consecutive feature updates. Some read-only features (such as ‘InternalTemperature’) are read internally from the acquisition device at a certain frequency in order to always stay up to date.

Note that this method is only relevant for acquisition devices which are supported through the Network Imaging Package (GigE Vision Framework). Other devices do not return a polling time, but instead use internal polling that generates “Feature Info Changed” events whenever required.

Demo/Example Usage

Not available

SapFeature::GetRepresentation

BOOL **GetRepresentation**(SapFeature::Representation* *representation*);

Parameters

representation Returned representation can be one of the following values:

SapFeature::RepresentationUndefined	Undefined representation
SapFeature::RepresentationLinear	The feature follows a linear scale
SapFeature::RepresentationLogarithmic	The feature follows a logarithmic scale
SapFeature::RepresentationBoolean	The feature can have two values: zero or non-zero

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the mathematical representation of a integer or float feature.

Demo/Example Usage

Not available

SapFeature::GetSelectedFeatureCount

BOOL **GetSelectedFeatureCount**(int* *selectedCount*);

Parameters

selectedCount Returned number of features associated with the selector, 0 if the current feature is not a selector.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the number of features associated with a selector (value returned by IsSelector method is TRUE). These selected features can be considered as children of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature::GetSelectedFeatureIndex

BOOL **GetSelectedFeatureIndex**(int *selectedIndex*, int* *featureIndex*);

Parameters

selectedIndex	Index of the selected feature, relative to the selector, from 0 to the value returned by the GetSelectedFeatureCount method, minus 1.
featureIndex	Returned index of the selected feature, relative to the acquisition device, from 0 to the value returned by the SapAcqDevice::GetFeatureCount method, minus 1.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the acquisition device index of a feature associated with a selector (value returned by the IsSelector method is TRUE). This feature can be considered as a child of the current SapFeature object.

The number of features associated with the selector is returned by the GetSelectedFeatureCount method.

The returned index can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the.SapAcqDevice::GetFeatureCount method.

Demo/Example Usage

Not available

SapFeature::GetSelectedFeatureName

BOOL **GetSelectedFeatureName**(int *selectedIndex*, char* *featureName*, int *featureNameSize*);

Parameters

selectedIndex	Index of the selected feature, relative to the selector, from 0 to the value returned by the GetSelectedFeatureCount method, minus 1.
featureName	Acquisition device feature name.
featureNameSize	Size (in bytes) of the buffer pointed to by *featureName*

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the acquisition device name of a feature associated with a selector (value returned by the IsSelector method is TRUE). This feature can be considered as a child of the current SapFeature object.

The number of features associated with the selector is returned by the GetSelectedFeatureCount method.

The returned name can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the.SapAcqDevice::GetFeatureCount method.

Demo/Example Usage

Not available

SapFeature::GetSelectingFeatureCount

BOOL **GetSelectingFeatureCount**(int* *selectingCount*);

Parameters

selectingCount Returned number of selectors associated with the current feature, 0 if there are no associated selectors.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the number of selectors (value returned by IsSelector method is TRUE) associated with a feature. These selectors can be considered as parents of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature::GetSelectingFeatureIndex

BOOL **GetSelectingFeatureIndex**(int *selectingIndex*, int* *featureIndex*);

Parameters

selectingIndex Index of the selector, relative to the current feature, from 0 to the value returned by the GetSelectingFeatureCount method, minus 1.

featureIndex Returned index of the selector, relative to the acquisition device, from 0 to the value returned by the SapAcqDevice::GetFeatureCount method, minus 1.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the acquisition device index of a selector (value returned by the IsSelector method is TRUE) associated with a feature. This selector can be considered as a parent of the current SapFeature object.

The number of selectors associated with a feature is returned by the GetSelectingFeatureCount method.

The returned index can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the.SapAcqDevice::GetFeatureCount method.

Demo/Example Usage

Not available

SapFeature::GetSelectingFeatureName

BOOL **GetSelectingFeatureName**(int *selectingIndex*, char* *featureName*, int *featureNameSize*);

Parameters

selectingIndex	Index of the selector, relative to the current feature, from 0 to the value returned by the GetSelectingFeatureCount method, minus 1.
featureName	Acquisition device feature name.
featureNameSize	Size (in bytes) of the buffer pointed to by *featureName*

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Returns the acquisition device name of a selector (value returned by the IsSelector method is TRUE) associated with a feature. This selector can be considered as a parent of the current SapFeature object.

The number of selectors associated with a feature is returned by the GetSelectingFeatureCount method.

The returned name can be used by the SapAcqDevice::GetFeatureInfo method to access the corresponding SapFeature object. The number of features supported by the acquisition device is returned by the.SapAcqDevice::GetFeatureCount method.

Demo/Example Usage

Not available

SapFeature::GetSign

BOOL **GetSign**(SapFeature::Sign* *sign*);

Parameters

sign	Returned sign can be one of the following values:	
	SapFeature::SignUndefined	Sign is undefined
	SapFeature::Signed	The feature is a signed integer of float
	SapFeature::Unsigned	The feature is an unsigned integer of float

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the sign of an integer or float feature. This information is useful when reading and writing feature values. By knowing the sign of the feature value you can cast it to the corresponding C/C++ type.

Demo/Example Usage

Not available

SapFeature::GetSiToNativeExp10

BOOL **GetSiToNativeExp10**(int* *exponent*);

Parameters

exponent Returned exponent value (base 10). It can be negative or positive.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the exponent for converting the value of a feature from international system (SI) units to native units (the units used to read/write the feature through the API).

The following equation describes the relation between the two unit systems:

$$V_{NATIVE} = V_{SI} * 10^{E}$$

Where *V* is the value of a feature and *E* is the current parameter.

Example 1

You want to set the camera exposure time to a known value in seconds. The 'ExposureTime' feature is represented in microseconds. Therefore the current exponent value is 6. If the desired integration time is 0.5 second, then you can compute the actual value for the SapAcqDevice::SetFeatureValue method as follows:
$V_{NATIVE} = 0.5*10^{6} = 500000$

Example 2

You want to monitor the temperature of the camera sensor. The 'InternalTemperature' feature is reported in degrees Celcius. Therefore the current exponent value is 0. If the feature value returned by the SapAcqDevice::GetFeatureValue method is 50 then the temperature in Celcius is also equal to 50.

Use the GetSiUnit method to retrieve the international system (SI) units corresponding to the feature to monitor.

Demo/Example Usage

Camera Events Example, GigE Auto-White Balance Example

SapFeature::GetSiUnit

BOOL **GetSiUnit**(char* *unit*, int *unitSize*);

Parameters

unit Buffer for the returned text string. Must be large enough for 32 characters.

unitSize Size of the buffer pointed to by *unit* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the physical units representing the feature in the international system (SI). Examples of units are Volts, Pixels, Celsius, Degrees, etc. This information is useful to present in a graphical user interface.

Most of the time the units used by the feature (the native units) are NOT the same as SI units, but rather a multiple of them. For example, the exposure time may be represented in microseconds instead of seconds. To convert the feature value to the SI units you must use the exponent value provided by the GetSiToNativeExp10 method.

Demo/Example Usage

Not available

SapFeature::GetToolTip

BOOL **GetToolTip**(char* *tooltip*, int *tooltipSize*);

Parameters

tooltip	Buffer for the returned text string. Must be large enough for 256 characters.
tooltipSize	Size of the buffer pointed to by *tooltip* (in bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the text which represents the explanation of the feature. This information can be used to implement tool tips in a graphical user interface.

Demo/Example Usage

Not available

SapFeature::GetType

BOOL **GetType**(SapFeature::Type **type*);

Parameters

type Returned type can be one of the following values:

SapFeature::TypeUndefined	Undefined type
SapFeature::TypeInt32	32-bit integer
SapFeature::TypeInt64	64-bit integer
SapFeature::TypeFloat	32-bit floating-point
SapFeature::TypeDouble	64-bit floating-point
SapFeature::TypeBool	Boolean
SapFeature::TypeEnum	Enumeration
SapFeature::TypeString	ASCII character string
SapFeature::TypeBuffer	Sapera LT buffer object (SapBuffer)
SapFeature::TypeLut	Sapera LT look-up table object (SapLut)
SapFeature::TypeArray	Sapera LT buffer object (SapBuffer)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the data type of the feature.

If the feature is of array type, then the SapBuffer object should have a height of one line, and a width corresponding to the number of bytes given by the value returned by the GetArrayLength method.

Demo/Example Usage

Camera Features Example

SapFeature::GetValidValue

BOOL **GetValidValue**(int *validValueIndex*, INT32* *validValue*);
BOOL **GetValidValue**(int *validValueIndex*, UINT32* *validValue*);
BOOL **GetValidValue**(int *validValueIndex*, INT64* *validValue*);
BOOL **GetValidValue**(int *validValueIndex*, UINT64* *validValue*);
BOOL **GetValidValue**(int *validValueIndex*, float* *validValue*);
BOOL **GetValidValue**(int *validValueIndex*, double* *validValue*);

Parameters

validValueIndex	Index of the valid value, can be any value from 0 to the value returned by the GetValidValueCount function, minus 1
validValue	Returned valid value, must point to a variable of the same type as the feature

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets one of a predefined set of valid values for an integer or floating-point feature which defines them as a list, that is, the GetIncrementType function returns SapFeature::IncrementList.

Demo/Example Usage

Camera Features Example

SapFeature::GetValidValueCount

BOOL **GetValidValueCount**(int* *validValueCount*);

Parameters

validValueCount	Returned count of valid values.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Get the number of valid values for an integer or floating-point feature which defines them as a list, that is, the GetIncrementType function returns SapFeature::IncrementList. In this case, use the GetValidValue function to enumerate these values.

Demo/Example Usage

Camera Features Example

SapFeature::GetVisibility

BOOL **GetVisibility**(SapFeature::Visibility* *visibility*);

Parameters

visibility	Returned visibility can be one of the following values:	
	SapFeature::VisibilityUndefined	Undefined visibility level
	SapFeature::VisibilityBeginner	The feature should be made visible to any user
	SapFeature::VisibilityExpert	The feature should be made visible to users with a certain level of expertise
	SapFeature::VisibilityGuru	Specifies that the feature should be made visible to users with a high level of expertise
	SapFeature::VisibilityInvisible	The feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the level of visibility assigned to a feature. This information is useful to classify the features in a graphical user interface in terms of user expertise.

Demo/Example Usage

Not available

SapFeature::GetWriteMode

BOOL **GetWriteMode**(SapFeature::WriteMode* *writeMode*);

Parameters

writeMode Returned write mode can be one of the following values:

SapFeature::WriteUndefined	Undefined write mode
SapFeature::WriteAlways	The feature can always be written
SapFeature::WriteNotAcquiring	The feature can only be written when the transfer object is not acquiring. If the transfer is currently acquiring you must stop the acquisition using the SapTransfer.Freeze or SapTransfer.Wait methods before modifying the feature value.
SapFeature::WriteNotConnected	The feature can only be written when the transfer object is not connected. If the transfer is currently connected you must disconnect it using the SapTransfer.Disconnect or SapTransfer.Destroy method before modifying the feature value. After modifying the value reconnect the transfer object using the SapTransfer.Connect or SapTransfer.Create method.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Checks if a feature can be modified when the corresponding transfer object (SapTransfer) is connected and/or acquiring. This transfer object is the one which uses the SapAcqDevice object from which the feature object was read.

Some features like buffer dimensions cannot be changed while data is being transfered to the buffer. Use this information to prevent an application from changing certain features when the transfer object is connected and/or acquiring.

Note that this function is only relevant for features which are writable, that is, when the GetAccessMode function identifies the feature as read/write or read-only.

Demo/Example Usage

Not available

SapFeature::IsEnumEnabled

BOOL **IsEnumEnabled**(int *enumIndex*, BOOL* *enabled*);

Parameters

enumIndex Index of the enumeration item (from 0 to the value returned by the GetEnumCount method, minus 1)

enabled Returned item enabled value (TRUE or FALSE)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Checks if the enumeration value corresponding to a specified index is enabled.

Each item in an enumeration is present for all the application duration. However an enumeration item may be dynamically enabled/disabled according to the value of another feature. Use this function to find out the enable state of an item at a given time.

Demo/Example Usage

Not available

SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile

BOOL **IsSavedToConfigFile**(BOOL* *savedToConfigFile*);
BOOL **SetSavedToConfigFile**(BOOL *savedToConfigFile*);

Parameters

savedToConfigFile TRUE for allowing the feature to be saved, FALSE otherwise.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Checks if a feature is saved to a CCF configuration file when calling the SapAcqDevice::SaveFeatures method.

All features are assigned a default behavior. For example, the read-only features are not saved while the read/write features are. You can, however, change the default behavior. For example a read-only feature such as 'InternalTemperature' is not saved by default. You can set *savedToConfigFile* to TRUE to force the feature to be written to the configuration file.

If you force read-only features to be saved those features will not be restored when loading back the CCF file. The reason is that the features are not writable to the device.

For acquisition devices which are not supported through the Network Imaging Package (GigE Vision Framework), the features saved to the configuration file are hardcoded and cannot be changed. Therefore these functions have no effect and always return FALSE.

Demo/Example Usage

Not available

SapFeature::IsSelector

BOOL **IsSelector**(BOOL* *isSelector*);

Parameters

isSelector Returns TRUE if the current feature is a selector, FALSE otherwise

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Determines if the value of the current feature directly affects the values of other features, using a one to many parent-child relationship. For example, if the current feature represents a look-up table index, then the affected features could represent values associated with one specific look-up table.

In this case, the current feature is called the selector.

Use the following methods to find out which features are associated: GetSelectedFeatureCount, GetSelectedFeatureIndex, and GetSelectedFeatureName.

You can only call IsSelector after the Create method

Demo/Example Usage

Not available

SapFeature::IsStandard

BOOL **IsStandard**(BOOL* *isStandard*);

Parameters

isStandard Returns whether the feature is standard (TRUE) or not (FALSE).

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Checks if a feature is standard or custom. Most of the features are standard. However, sometimes custom features might be provided as part of a special version of an acquisition device driver.

Demo/Example Usage

Not available

SapLocation

The SapLocation Class identifies a Sapera server/resource pair.

A Sapera server is an abstract representation of a physical device like a frame grabber, a processing board, a GigE camera, or the host computer. In general, a Teledyne DALSA board or GigE camera camera is a server. Resources are attached to these physical devices. For example, a frame grabber can have one or more acquisition resources.

Sapera Class methods do not always need the server information from SapLocation. In these cases, the resource index is simply ignored.

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

SapLocation Class Members

Construction

SapLocation	Class constructor

Attributes

GetServerIndex	Gets the server index
GetServerName	Gets the server name
GetResourceIndex	Gets the resource index
IsUnknown	Checks if neither the server index nor the server name is valid

SapLocation Member Functions

The following are members of the SapLocation Class.

SapLocation::SapLocation

SapLocation();
SapLocation(int *serverIndex*, int *resourceIndex* = 0);
SapLocation(const char **serverName*, int *resourceIndex* = 0);
SapLocation(const SapLocation &*loc*);
SapLocation(const SapLocation &*loc*, int *resourceIndex*);

Parameters

serverIndex	Sapera server index. There is always one server associated with the host computer at SapLocation::ServerSystem (index 0).
serverName	Sapera server name. The 'System' server is associated with the host computer.
resourceIndex	Sapera resource index
loc	Existing SapLocation object from which server and resource information are to be extracted.

Remarks

Use the Sapera Configuration utility to find the names and indices of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

The constructor with only the *loc* argument allows you to reuse the same server and resource information. The constructor with both *loc* and *resourceIndex* allows use to reuse the same server information, while specifying a different resource index.

Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, IO Demo, GigE Auto-White Balance Example,. GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapLocation::GetResourceIndex

int **GetResourceIndex**();

Remarks

Returns the resource index.

Demo/Example Usage

Not available

SapLocation::GetServerIndex

int **GetServerIndex**();

Remarks

Returns the server index. If the returned value is equal to SapLocation::ServerUnknown, it does not necessarily mean that the object is invalid. In this case, use the GetServerName method instead.

Demo/Example Usage

Not available

SapLocation::GetServerName

char* **GetServerName**();

Remarks

Returns the server name. If the returned value is an empty string, it does not necessarily mean that the object is invalid. In this case, use the GetServerIndex method instead.

Demo/Example Usage

Not available

SapLocation::IsUnknown

BOOL **IsUnknown**();

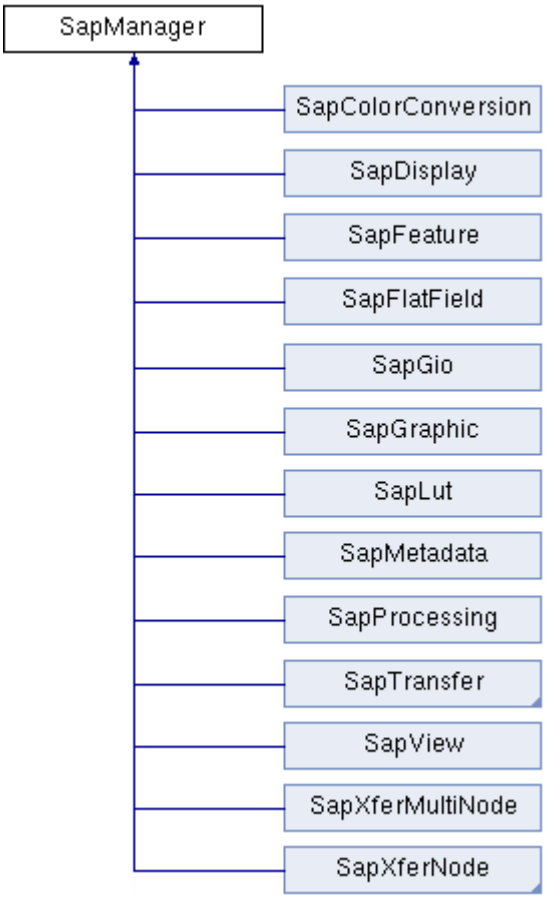
Remarks

Returns TRUE if neither the server index nor the server name is valid, FALSE otherwise. Although an unknown SapLocation is usually invalid, it may be useful in some particular cases.

Demo/Example Usage

Not available

SapManager



The SapManager Class includes methods for describing the Sapera resources present on the system. It also includes error management capabilities.

You will never need to explicitly create a SapManager object. First, almost all methods are declared as static, which means you may use them at any time. Second, most Sapera LT ++ classes are derived from SapManager, so they inherit its methods and protected data members.

#include <SapClassBasic.h>

SapManager Class Members

Attributes

operator BOOL	Checks whether the Create method has succeeded for a derived object
GetDisplayStatusMode, SetDisplayStatusMode	Gets/sets the global reporting mode for messages and errors
GetCommandTimeout, SetCommandTimeout	Gets/sets the timeout value used when waiting for completion of Sapera LT commands
GetResetTimeout, SetResetTimeout	Gets/sets the timeout value used when resetting a hardware device
GetServerEventType	Gets the currently registered event type for server related events

Operations

Open	Initializes access to the Sapera LT libraries
Close	Terminates access to the Sapera LT libraries
DetectAllServers	Detects GenCP cameras after a Sapera application has been started
GetVersionInfo	Gets Sapera LT version and licensing information
GetServerCount	Gets the number of available Sapera servers
GetServerIndex	Gets the index of a Sapera server
GetServerName	Gets the name of a Sapera server
GetServerType	Gets the type of a Sapera server
IsServerAccessible	Checks if the resources for a server are accessible
GetServerHandle	Returns the low-level Sapera handle of a server resource
GetServerSerialNumber	Gets the serial number corresponding to a Sapera server
GetResourceCount	Gets the number of Sapera resources of a specific type on a server
GetResourceIndex	Gets the index of a Sapera resource
GetResourceName	Gets the name of a Sapera resource
IsResourceAvailable	Checks whether a resource is available for use
IsSystemLocation	Check whether a SapLocation object is located on the system server
IsSameServer	Checks whether two SapLocation objects are located on the same server
IsSameLocation	Checks whether two SapLocation objects are the same
GetFormatType	Gets the data type corresponding to a Sapera data format
GetStringFromFormat	Gets a text description of a Sapera data format
GetLastStatus	Gets a description of the latest Sapera low-level C library error
DisplayMessage	Reports a custom message using the current reporting mode
ResetServer	Resets the hardware device associated with a specific server
GetInstallDirectory	Gets the directory where a Sapera product is installed
RegisterServerCallback	Registers a callback function for server related events
UnregisterServerCallback	Unregister the callback function for server related events
WriteFile	Writes a file to non-volatile memory on the device

SapManager Member Functions

The following are members of the SapManager Class.

SapManager::operator BOOL

operator BOOL();

Remarks

Checks whether the Create method has succeeded for an object derived from SapManager. This allows the variable representing the object to be used in a Boolean expression.

Calling the Destroy method resets this attribute to FALSE.

Demo/Example Usage

All demos and examples

SapManager::Close

static BOOL **Close**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Terminates access to the Sapera LT libraries. See the SapManager::Open method for more details.

Demo/Example Usage

Find Camera Example

SapManager::DetectAllServers

static BOOL **DetectAllServers**(DetectServerType *type* = DetectServerAll);

Parameters

type	Specifies the type of server to detect. Possible values are:	
	DetectServerType::DetectServerGenCP	Detect GenCP servers only.
	DetectServerType::DetectServerAll	Currently equivalent to GenCP only.

Remarks

Use this function to detect GenCP cameras after a Sapera application has been started. In a typical application device detection (discovery) is initiated during application startup. If a GenCP camera is connected after an application has been launched, it will not be detected automatically. Use this function to trigger the camera discovery process.

Note that you must register the EventServerNew event before calling this function. See SapManager::RegisterServerCallback for details.

Demo/Example Usage

Find Camera example

SapManager::DisplayMessage

static void **DisplayMessage**(const char* *message*, const char* *fileName* = NULL, int *lineNumber* = 0, ...);

Parameters

message	Custom message to report
fileName	Name of source file from which DisplayMessage is called
lineNumber	Line number from which DisplayMessage is called
...	Variable arguments if *message* includes printf-style format specifications

Remarks

Reports a custom message using the current reporting mode. File and line information is automatically appended to *message*, unless you set *fileName* to NULL.

See the SetDisplayStatusMode method for a description of the available reporting modes.

Note that, when the reporting mode is set to the Log Viewer, messages are logged as errors. It is possible to log these as informational instead by using the case insensitive '(Sapera app)' prefix at the beginning of each message. This prefix will be automatically stripped from the message before logging.

Demo/Example Usage

Not available

SapManager::GetCommandTimeout, SapManager::SetCommandTimeout

static int **GetCommandTimeout**();
static void **SetCommandTimeout** (int *commandTimeout*);

Remarks

Gets/sets the timeout value (in milliseconds) used when waiting for completion of Sapera LT commands. The initial value for this attribute is 20000 (20 seconds).

If you need to control the timeout value used by the ResetServer method, use the GetResetTimeout and SetResetTimeout methods instead.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo, Camera Files Example

SapManager::GetDisplayStatusMode, SapManager::SetDisplayStatusMode

static SapManager::StatusMode **GetDisplayStatusMode**();
static BOOL **SetDisplayStatusMode**(SapManager::StatusMode *mode,* SapManCallback *pCallback* = NULL,
void* *pContext* = NULL);

Parameters

mode	New reporting mode. The following values are available:	
	SapManager::StatusNotify	Sends messages to a popup window
	SapManager::StatusLog	Sends messages to the Sapera Log Server (can be displayed using the Sapera Log Viewer)
	SapManager::StatusDebug	Sends messages to the active debugger, if any
	SapManager::StatusCustom	Error information is not reported, it is just stored internally
	SapManager::StatusCallback	Notifies application code through a callback function
pCallback	Application callback function to be called when reporting a message. This function must be declared as: void MyCallback(SapManCallbackInfo* *pInfo*);	
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.	

Remarks

Gets/sets the global reporting mode for messages and errors. This mode is used by the IsStatusOk and DisplayMessage methods, and also internally by the Sapera LT ++ library.

The initial value for this attribute is StatusNotify.

For StatusCallback reporting mode, you can call one of the following SapManCallbackInfo methods from the application callback function to retrieve the relevant information: GetErrorValue, GetErrorMessage, and GetContext.

For StatusCustom reporting mode, the only way to retrieve messages is by calling the GetLastStatus method.

Note that, for all reporting modes, any Sapera LT ++ method returns FALSE to indicate an error.

Demo/Example Usage

Camera Features Example, Find Camera Example

SapManager::GetFormatType

static SapFormatType **GetFormatType**(SapFormat *format*);

Parameters

format Sapera data format

Remarks

Gets the data type corresponding to the specified Sapera data format as one of the following values:

SapFormatTypeUnknown	Unable to determine data type
SapFormatTypeMono	Monochrome
SapFormatTypeRGB	RGB color
SapFormatTypeYUV	YUV color
SapFormatTypeHSI	HSI color
SapFormatTypeHSV	HSV color
SapFormatTypeColor	Lookup table color data
SapFormatTypeRGBA	RGB color with an additional component (alpha channel, infrared component, etc.)

Demo/Example Usage

Not available

SapManager::GetInstallDirectory

static BOOL **GetInstallDirectory**(int *serverIndex*, char* *installDir*);
static BOOL **GetInstallDirectory** (const char* *serverName*, char* *installDir*);
static BOOL **GetInstallDirectory** (SapLocation *loc*, char* *installDir*);

Parameters

serverIndex	Sapera server index
installDir	Memory area large enough to receive the installation directory (at least 257 bytes)
serverName	Sapera server name
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the directory where a Sapera product is installed.

For the system server, this corresponds to the Sapera installation directory, for example, **c:\ Program Files\Teledyne DALSA\Sapera**.

For a server corresponding to a hardware device, this corresponds to the directory where the driver for the device is installed, .for example, **c: \Program Files\Teledyne DALSA\X64 Xcelera-CL PX4**.

Demo/Example Usage

Not available

SapManager::GetLastStatus

static const char* **GetLastStatus**();
static void **GetLastStatus**(SAPSTATUS* *pLastStatus*);

Parameters

pLastStatus Pointer to the most recent Sapera low-level status code to retrieve

Remarks

Gets a description of the latest Sapera LT ++ and/or low-level C library error.

The first form of GetLastStatus returns the latest text description, similar to what is generated by the IsStatusOk method. If the actual error occurred inside a call to the low-level C library, then you may also use the second form to retrieve the actual error code.

Note that each thread in a Sapera LT application has its own latest error code and description. This means that you cannot call GetLastStatus to retrieve error information for a Sapera LT ++ function which has been called in another thread.

See the SetDisplayStatusMode method for a description of the available reporting modes.

Demo/Example Usage

Not available

SapManager::GetResetTimeout, SapManager::SetResetTimeout

static int **GetResetTimeout**();
static void **SetResetTimeout** (int *timeOut*);

Remarks

Gets/sets the timeout value (in milliseconds) used when resetting a hardware device. This value is used by the ResetServer method.

If you need to get/set the timeout value used when waiting for completion of Sapera LT commands, use the GetCommandTimeout and SetCommandTimeout methods instead.

The initial value for this attribute is 20000 (20 seconds).

Demo/Example Usage

Not available

SapManager::GetResourceCount

static int **GetResourceCount**(int *serverIndex*, SapManager::ResType *resourceType*);
static int **GetResourceCount**(const char* *serverName*, SapManager::ResType *resourceType*);
static int **GetResourceCount**(SapLocation *loc*, SapManager::ResType *resourceType*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.
serverName	Sapera server name
loc	Valid SapLocation object

Remarks

Gets the number of resources of a specified type on a Sapera server. This only applies to static resources, that is, those attached to physical devices. Dynamic resources, like buffers, do not have a fixed count.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

IO Demo, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

SapManager::GetResourceIndex

static int **GetResourceIndex**(int *serverIndex*, SapManager::ResType *resourceType*, const char* *resourceName*);
static int **GetResourceIndex**(const char **serverName*, SapManager::ResType *resourceType*, const char* *resourceName*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceName	Sapera resource name
serverName	Sapera server name

Remarks

Gets the index of a Sapera resource. Returns SapLocation::ResourceUnknown if the specified resource cannot be found.

The first form of GetResourceIndex looks for the resource of the specified name and type on the server specified by *index*. The second form uses the server name instead of the index.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server and resource names for that product.

Demo/Example Usage

Not available

SapManager::GetResourceName

static BOOL **GetResourceName**(int *serverIndex*, SapManager::ResType *resourceType*, int *resourceIndex*,
char* *resourceName*);
static BOOL **GetResourceName**(const char **serverName*, SapManager::ResType *resourceType*,
int *resourceIndex*, char* *resourceName*, int *nameSize* = MaxLabelSize);
static BOOL **GetResourceName**(SapLocation *loc*, SapManager::ResType *resourceType*, char*
resourceName);

Parameters

serverIndex	Index of Sapera server containing the resource.
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type.
resourceName	Memory area large enough to receive the resource name (at least 128 bytes).
serverName	Name of Sapera server containing the resource.
loc	Valid SapLocation object.
nameSize	Size of memory allocated by *resourceName*, in bytes.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the name of a Sapera resource of a specified type.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the ‘Servers and Resources’ section in the user’s manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

Demo/Example Usage

Not available

SapManager::GetServerCount

static int **GetServerCount**();
static int **GetServerCount**(SapManager::ResType *resourceType*);

Parameters

resourceType	Resource type to inquire, can be one of the following:	
	SapManager::ResourceAcq	Frame grabber acquisition hardware
	SapManager::ResourceAcqDevice	Camera acquisition hardware (for example, Genie)
	SapManager::ResourceCounter	Event counters
	SapManager::ResourceDisplay	Physical displays
	SapManager::ResourceDsp	Digital Signal Processors
	SapManager::ResourceGio	General inputs and outputs
	SapManager::ResourceGraphic	Graphics engine
	SapManager::ResourceRtPro	Realtime Processor hardware

Remarks

Gets the number of available Sapera servers.

The first form of this method considers all servers, regardless of their resource type. In this case, the return value is at least 1, since the system server is always present. The second form returns the number of servers for the specified resource type only, so the return value may be equal to 0.

Note that the following resource types apply only to older products: ResourceCab, ResourceCounterResourceDsp, and ResourcePixPro. See the *Sapera LT ++ Legacy Classes Reference Manual* for related classes.

Demo/Example Usage

IO Demo, Find Camera Example

SapManager::GetServerEventType

static SapManager::EventType **GetServerEventType**();

Remarks

Gets the currently registered event type for server related events. See the RegisterServerCallback method for the list of possible values.

If this method returns the special value SapManager::EventNone, this means that no server related events are currently registered. This is the default value, which is also reset when calling the UnregisterServerCallback method.

Demo/Example Usage

Not available

SapManager::GetServerHandle

static BOOL **GetServerHandle**(int *serverIndex*, PCORSERVER *pServer*);
static BOOL **GetServerHandle**(const char* *serverName*, PCORSERVER *pServer*);
static BOOL **GetServerHandle**(SapLocation *loc*, PCORSERVER *pServer*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
pServer	Pointer to returned low-level Sapera server handle

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the low-level handle of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

Not available

SapManager::GetServerIndex

static int **GetServerIndex**(const char* *serverName*);
static int **GetServerIndex**(SapLocation *loc*);

Parameters

serverName	Sapera server name
loc	Valid SapLocation object

Remarks

Gets the index of a Sapera server. Returns SapLocation::ServerUnknown if the specified server cannot be found.

The first form of this method uses the server name. The second form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

IO Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

SapManager::GetServerName

static BOOL **GetServerName**(int *serverIndex*, char* *serverName*);
static BOOL **GetServerName**(SapLocation *loc*, char* *serverName*);
static BOOL **GetServerName**(int *serverIndex*, SapManager::ResType *resourceType*, char* *serverName*);

Parameters

serverIndex	Sapera server index
serverName	Memory area large enough to receive the server name (at least 32 bytes)
loc	Valid SapLocation object
resourceType	Resource type to inquire. See the GetServerCountmethod for the list of possible values.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the name of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses an existing SapLocation object with valid server information. The third form only considers servers with at least one resource of the specified type. For example, index 1 corresponds to the second server with at least one acquisition device.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

Dual Acquisition Demo, IO Demo, Find Camera Example

SapManager::GetServerSerialNumber

static BOOL **GetServerSerialNumber**(int *serverIndex*, char* *serialNumber*);
static BOOL **GetServerSerialNumber**(const char* *serverName*, char* *serialNumber*);
static BOOL **GetServerSerialNumber**(SapLocation *loc*, char* *serialNumber*);

Parameters

serverIndex	Sapera server index
serialNumber	Memory area large enough to receive the text for the serial number (at least 16 bytes)
serverName	Sapera server name
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets a text representation of the serial number corresponding to the hardware device for the specified Sapera server. It consists of either the letter 'S' or 'H' followed by seven digits, for example, "S1234567".
Note that there is no serial number associated with the System server. Also, this function is only supported for frame grabbers and older Genie cameras (not Genie-TS). When using other camera servers (GigE-Vision or GenCP), you need a valid SapAcqDevice object from which the serial number can be retrieved through a named feature.

Demo/Example Usage

Not available

SapManager::GetServerType

static SapManager::ServerType **GetServerType**(int *serverIndex*);

static SapManager::ServerType **GetServerType**(const char* *serverName*);
static SapManager::ServerType **GetServerType**(SapLocation *loc*);

Parameters

serverIndex	Sapera server index	
serverName	Sapera server name	
loc	Valid SapLocation object	
Return Value	Can be one of the following:	
	SapManager::ServerNone	Server type cannot be determined
	SapManager::ServerSystem	System server
	SapManager::ServerBandit3CV	Bandit-3 CV Express VGA frame grabber
	SapManager::ServerX64CL	X64-CL acquisition board
	SapManager::ServerX64CLiPRO	X64-CL iPro acquisition board
	SapManager::ServerX64CLExpress	X64-CL-Express acquisition board
	SapManager::ServerX64CLLX4	X64 Xcelera-CL LX4 acquisition board
	SapManager::ServerX64CLPX4	X64 Xcelera-CL PX4 acquisition board
	SapManager::ServerX64LVDS	X64-LVDS acquisition board
	SapManager::ServerX64LVDSPX4	X64-LVDSPX4 acquisition board
	SapManager::ServerX64LVDSVX4	X64-LVDSVX4 acquisition board
	SapManager::ServerX64ANQuad	X64-AN Quad acquisition board
	SapManager::ServerX64ANLX1	X64-ANLX1 acquisition board
	SapManager::ServerPC2Vision	PC2-Vision acquisition board
	SapManager::ServerPC2Comp	PC2-Comp Express acquisition board
	SapManager::ServerPC2CamLink	PC2-CamLink acquisition board
	SapManager::ServerGenie	Genie camera
	SapManager::ServerAnacondaCL	Anaconda-CL vision processor
	SapManager::ServerAnacondaLVDS	Anaconda-LVDS vision processor

Remarks

Gets the type of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

Not available

SapManager::GetStringFromFormat

static BOOL **GetStringFromFormat**(SapFormat *format*, char* *txtFormat*);

Parameters

format Sapera data format

txtFormat Memory area large enough to receive the description (at least 16 bytes)

Return Value

Returns TRUE if the low-level function call succeeded, FALSE otherwise

Remarks

Gets a text description of the specified Sapera data format, for example, 'MONO8' for SapFormatMono8.

Demo/Example Usage

Grab Console Multiformat Example

SapManager::GetVersionInfo

BOOL **GetVersionInfo**(SapManVersionInfo* *pVersionInfo*);

Parameters

pVersionInfo Pointer to a SapManVersionInfo object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets Sapera LT version and licensing information. When the method call returns, the SapManVersionInfo object pointed to by *pVersionInfo* contains the information. It has the following attributes:

int GetMajor()	Gets the major version number. For example, if the version number is 6.30.01.0806, this method returns 6.
int GetMinor()	Gets the minor version number. For example, if the version number is 6.30.01.0806, this method returns 30.
int GetRevision()	Gets the revision number. For example, if the version number is 6.30.01.0806, this method returns 1.
int GetBuild()	Gets the build number. For example, if the version number is 6.30.01.0806, this method returns 806.
SapManVersionInfo::LicenseType GetLicenseType()	Gets the Sapera LT license type for the current installation, which can be one of SapManVersionInfo::Runtime, SapManVersionInfo::Evaluation, or SapManVersionInfo::FullSDK
int GetEvalDaysRemaining()	Gets the number of days remaining in the evaluation period when the license type is Runtime, where a value equal to 0 means that the evaluation period has expired.

Demo/Example Usage

Not available

SapManager::IsResourceAvailable

static BOOL **IsResourceAvailable**(int *serverIndex*, SapManager::ResType *resourceType*, int *resourceIndex*);
static BOOL **IsResourceAvailable**(const char* *serverName*, SapManager::ResType *resourceType*, int *resourceIndex*);
static BOOL **IsResourceAvailable** (SapLocation *loc*, SapManager::ResType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

Return Value

Returns TRUE if the specified resource is not already used, FALSE otherwise

Remarks

Determines if a specific Sapera resource on a server is available. You may use this method, for example, before calling SapAcquisition::Create to avoid getting an error when the acquisition resource is already in use.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

Demo/Example Usage

Not available

SapManager::IsSameLocation

static BOOL **IsSameLocation**(SapLocation *loc1*, SapLocation *loc2*);

Parameters

loc1	First valid SapLocation object
loc2	Second valid SapLocation object

Remarks

Checks if the two specified SapLocation objects have the same server and resource information.

Demo/Example Usage

Not available

SapManager::IsSameServer

static BOOL **IsSameServer**(SapLocation *loc1*, SapLocation *loc2*);

Parameters

loc1 First valid SapLocation object

loc2 Second valid SapLocation object

Remarks

Checks if the two specified SapLocation objects have the same server information.

Demo/Example Usage

IsSameServer

SapManager::IsServerAccessible

static BOOL **IsServerAccessible**(int *serverIndex*);
static BOOL **IsServerAccessible**(const char* *serverName*);
static BOOL **IsServerAccessible**(SapLocation *loc*);

Parameters

serverIndex Index of Sapera server containing the resource

serverName Name of Sapera server containing the resource

Loc Valid SapLocation object

Return Value

Returns TRUE if the resources for the server are accessible, FALSE otherwise.

Remarks

Checks if the resources belonging to a server are currently accessible. Although existing objects for these resources are still valid when their server becomes unaccessible, they must be left alone or destroyed (for example, SapAcqDevice::Destroy).

When a Sapera application starts, all detected servers are automatically accessible. However, Sapera camera devices (GigE-Vision and GenCP) can be connected and disconnected while a Sapera application is running. When such a device is connected for the first time, its server is automatically accessible. When the device is later disconnected, the server becomes unaccessible. If it is reconnected again, the server is once again accessible.

The first form of this method uses a server index. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object.

Accessibility of servers can also be determined by registering callbacks for server related events using the RegisterServerCallback method.

Note that you should not use this method for devices which are always connected (for example, frame grabbers), since the return value may not correspond to the actual resource accessibility for the corresponding server.

Demo/Example Usage

Not available

SapManager::IsSystemLocation

static BOOL **IsSystemLocation**();
static BOOL **IsSystemLocation**(SapLocation *loc*);

Parameters

loc Valid SapLocation object

Remarks

Check if the current application is running on the system server, or if the SapLocation object refers to this server.

Demo/Example Usage

Not available

SapManager::Open

static BOOL **Open**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Initiates access to the Sapera LT libraries.

For most applications you do not have to call this method, as the libraries are automatically loaded during the first Sapera LT ++ constructor call in the application code, and automatically unloaded during the last Sapera LT ++ destructor call. For example:

```
// No Sapera LT ++ calls before this line
// This loads the libraries
SapBuffer *buf1 = new SapBuffer();
SapBuffer *buf2 = new SapBuffer();
delete buf1;
// This unloads the libraries
delete buf2;
// No Sapera LT ++ calls after this line
```

There is, however, at least one case for which the default behavior may not be acceptable. If the application code frequently deletes all Sapera LT ++ objects and then reallocates them again, the libraries will be unloaded and reloaded each time, causing a noticeable delay. In this case, you can call the Open method as the first Sapera LT ++ call in the application, with a call to SapManager::Close as the last Sapera LT ++ call. This results in the libraries being loaded and unloaded exactly once, as follows:

```
// No Sapera LT ++ calls before this line
// This loads the libraries
SapManager::Open();
//
// Arbitrary Sapera LT ++ calls, none of which unloads or reloads the libraries
//
// This unloads the libraries
SapManager::Close();
// No Sapera LT ++ calls after this line
```

Demo/Example Usage

Find Camera Example

SapManager::RegisterServerCallback

static BOOL **RegisterServerCallback**(SapManager::EventType *eventType*, SapManCallback c*allback,* void* *context* = NULL);

Parameters

eventType Manager events for which the application callback function will be called. One or more of the following values may be combined together using a bitwise OR operation:

SapManager::EventServerNew	A new device is connected while a Sapera application is already running
SapManager::EventServerDisconnected	The device corresponding to an existing server is disconnected. (Replaces SapManager::EventServerNotAccessible which is now deprecated.)
SapManager::EventServerConnected	The device corresponding to an existing, unaccessible server is reconnected. (Replaces SapManager::EventServerAccessible, which is now deprecated.)
SapManager::EventServerDatabaseFull	There is no room in the Sapera server database for a new device that has just been connected
SapManager::EventResourceInfoChanged	The information describing a resource (typically its label) has changed
SapManager::EventServerFile	The information about the progress of the file being transferred

c*allback* Application callback function to be called each time one of the events specified above is received.

The callback function must be declared as:
void MyCallback(SapManCallbackInfo **pInfo*);

c*ontext* Optional pointer to an application context to be passed to the callback function.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Registers a callback function for server related events. The callback function provides information on the corresponding event (through the SapManCallbackInfo object). The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

In the callback function, you can obtain the event type that triggered the callback by calling the SapManCallbackInfo::GetEventType method. For all events except EventServerDatabaseFull, you can obtain the index of the server by calling SapManCallbackInfo::GetServerIndex. For the EventResourceInfoChanged event, you can obtain the index of the affected resource by calling SapManCallbackInfo::GetResourceIndex.

The EventResourceInfoChanged event can only occur as a result of modifying the value of the 'DeviceUserID' feature through the SapAcqDevice class.

The EventServerFile event will occur only when a firmware file is being uploaded to a device like a frame-grabber. You can obtain the progress of the upload by calling SapManCallbackInfo::GetFilePercentProgress.

Note that server related events are only available when dealing with Sapera camera devices (GigE-Vision and GenCP), that can be connected and disconnected while a Sapera application is running.

Demo/Example Usage

Not available

SapManager::ResetServer

static BOOL **ResetServer**(int *serverIndex*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void* *pContext* = NULL);

static BOOL **ResetServer**(const char* *serverName*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void* *pContext* = NULL);

static BOOL **ResetServer**(SapLocation *loc*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void* *pContext* = NULL);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
wait	Specifies whether this method should return immediately after resetting the specified server, or if it should wait for the server to be operational again
pCallback	Application callback function to be called when the server is operational again after a reset. The callback function must be declared as: void MyCallback(SapManCallbackInfo* *pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Resets the hardware frame grabber associated with a specific server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

There are three ways to use this method:

wait = TRUE *pCallback* = don't care	Returns only when the reset is complete, and the server is operational again
wait = FALSE *pCallback* = NULL	Returns immediately after resetting the server. The application program is then responsible for figuring out when the server is operational again.
wait = FALSE *pCallback* != NULL	Returns immediately after resetting the server, and notifies the application using the callback function when the server is operational again

You can call the GetServerIndex and GetContext methods of the SapManCallbackInfo class to retrieve the required information from the application callback function.

Note that all other Sapera LT ++ objects must be destroyed before calling this method, otherwise application behavior is undefined. To reset the server, use the following sequence:

- Call Destroy on all Sapera LT ++ objects (SapTransfer, SapBuffer, SapAcquisition, ...)
- Call ResetServer
- Call Create for all needed Sapera LT ++ objects

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Note: this method is only for use with frame grabbers; for cameras use the *DeviceReset* feature to reset

the device.

Demo/Example Usage

Not available

SapManager::UnregisterServerCallback

static BOOL **UnregisterServerCallback**(void);

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Unregisters the callback function for server related events.

Demo/Example Usage

Not available

SapManager::WriteFile

static BOOL **WriteFile**(int *serverIndex*, const char *l*o*calFilePath*, int *deviceFileIndex)*;
static BOOL **WriteFile(**const char* *serverName*, const char **localFilePath*, int *deviceFileIndex*);
static BOOL **WriteFile(**SapLocation *loc*, const char *l*o*calFilePath*, int *deviceFileIndex*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
deviceFileName	Name of the device file. See the acquisition device User's Manual for the list of supported files.
deviceFileIndex	Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
localFilePath	Full directory path and filename on the host computer to save the file.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes a file to non-volatile memory on the device. Refer to the acquisition device's User's Manual for the list of supported files.

Demo/Example Usage

Camera Files Example

SapManCallbackInfo

The SapManCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapManager class.

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

SapManCallbackInfo Class Members

Construction

SapManCallbackInfo	Class constructor

Attributes

GetEventType	Gets the manager events that triggered the call to the application callback
GetServerIndex	Gets the Sapera server index associated with the call to the application callback
GetResourceIndex	Gets the Sapera resource index associated with the call to the application callback
GetContext	Gets the application context associated with the callback
GetErrorValue	Gets the low-level Sapera C library error code associated with the callback
GetErrorMessage	Gets the error message associated with the callback
GetFilePercentProgress	Gets the file transfer progress when writing a file to a device.

SapManCallbackInfo Member Functions

The following are members of the SapManCallbackInfo Class.

SapManCallbackInfo::SapManCallbackInfo

```
SapManCallbackInfo(
  int serverIndex,
  void* context);

SapManCallbackInfo(
  SapManager::EventType eventType,
  int serverIndex,
  void* context);

SapManCallbackInfo(
  SapManager::EventType eventType,
  int serverIndex,
  int resourceIndex,
  void* context);

SapManCallbackInfo(
  SAPSTATUS errorValue,
  const char* errorMessage,
  void* context);

SapManCallbackInfo(
SapManager::EventType eventType,
int serverIndex,
void *context,
 int filePercentProgress)
```

Parameters

eventType	Combination of manager events. See the SapManager::RegisterServerCallback method for the list of possible values.
serverIndex	Sapera server index
resourceIndex	Sapera resource index
context	Pointer to the application context
errorValue	Low-level Sapera C library error code
errorMessage	Error message as a text string
filePercentProgress	File transfer progress

Remarks

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

SapManager uses this class in various situations. The first corresponds to the case when a server is operational again after being reset. The second case corresponds to the callback reporting mode which is set by calling the SapManager::SetDisplayStatusMode method.

The third case corresponds to other server related events, for example, when an acquisition device is physically disconnected. This involves explicitly registering a callback function using the SapManager::RegisterServerCallback method.

Demo/Example Usage

Not available

SapManCallbackInfo::GetContext

void* GetContext();

Remarks

Gets the application context associated with the call to the application callback. See the SapManager::SetDisplayStatusMode method for more details.

Demo/Example Usage

Not available

SapManCallbackInfo::GetErrorMessage

const char* GetErrorMessage();

Remarks

Gets the error message associated with the call to the application callback as a text string. See the SapManager::SetDisplayStatusMode method for more details.

Demo/Example Usage

Not available

SapManCallbackInfo::GetErrorValue

SAPSTATUS GetErrorValue();

Remarks

Gets the low-level Sapera C library error code associated with the call to the application callback. See the SapManager::SetDisplayStatusMode method for more details.

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

Demo/Example Usage

Not available

SapManCallbackInfo::GetEventType

SapManager::EventType GetEventType();

Remarks

Gets the combination of manager events that triggered the call to the application callback. See the SapManager::RegisterServerCallback method for more details.

Demo/Example Usage

Not available

SapManCallbackInfo::GetFilePercentProgress

int GetFilePercentProgress() const

Remarks

Gets the file transfer progress, as a percentage of the file size, when writing a file to a device.

Demo/Example Usage

Not available

SapManCallbackInfo::GetResourceIndex

int GetResourceIndex();

Remarks

Gets the Sapera resource index associated with the call to the application callback. See the SapManager::RegisterServerCallback method for more details.

Demo/Example Usage

Not available

SapManCallbackInfo::GetServerIndex

int GetServerIndex();

Remarks

Gets the Sapera server index associated with the call to the application callback. See the SapManager::ResetServer and SapManager::RegisterServerCallback methods for more details.

Demo/Example Usage

Not available

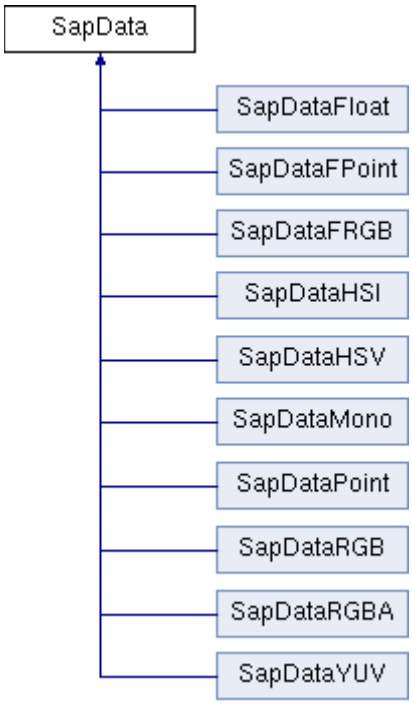
Sapera .NET Basic Class Reference

The reference material for Sapera .NET uses the C# language. Wherever there are significant differences with other .NET languages (for example, VB.NET), these differences will be clearly stated. For all other cases, you can use the following table, which lists data type equivalents between the various .NET languages, together with the corresponding type from the .NET Framework.

Native .NET	C#	VB.NET
System.Void	void	(none)
System.Int32	int	Integer
System.UInt32	uint	UInteger
System.Boolean	bool	Boolean
System.Single	float	Single
System.Double	double	Double
System.String	string	String
System.Int64	long	Long
System.UInt64	ulong	ULong
System.IntPtr	System.IntPtr	System.IntPtr

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

Data Classes



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

SapData Class

Purpose

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

void **Clear**();

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

SapFormatType **FormatType** (read-only property)

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

Demo/Example Usage

Not available

SapDataFRGB Class

Purpose
Encapsulates one element supporting Sapera floating-point RGB data types

SapDataFRGB();
SapDataFRGB(float *red*, float *green*, float *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specifies an initial value other than black

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

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

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

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

Demo/Example Usage

Not available

SapDataHSI Class

Purpose
Encapsulates one element supporting Sapera HSI data types

SapDataHSI();
SapDataHSI(int *h*, int *s*, int *i*);
Class constructor, where the *h*, *s*, and *i* arguments specify an initial value other than black

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

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

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

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

SapDataHSV Class

Purpose
Encapsulates one element supporting Sapera HSV data types

SapDataHSV();
SapDataHSV(int *h*, int *s*, int *v*);
Class constructor, where the *h*, *s*, and *v* arguments specify an initial value other than black

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

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

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

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

SapDataMono Class

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

SapDataMono();
SapDataMono(int *mono*);
Class constructor, where the *mono* argument specifies an initial value other than black

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

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

Demo/Example Usage

Example Common Utiltities

SapDataRGB Class

Purpose
Encapsulates one element supporting Sapera RGB data types

SapDataRGB();
SapDataRGB(int *red*, int *green*, int *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specify an initial value other than black

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

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

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

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

Demo/Example Usage

Example Common Utiltities

SapDataRGBA Class

Purpose
Encapsulates one element supporting Sapera RGB with alpha channel data types

SapDataRGBA();
SapDataRGBA(int *red*, int *green*, int *blue*, int *alpha*);
Class constructor, where the *red*, *green*, *blue* and *alpha* arguments specify an initial value other than black

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

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

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

int Blue (read/write property)
Returns the blue component of the current value of the data element

int Alpha (read/write property)
Returns the alpha component of the current value of the data element

Demo/Example Usage

Not available

SapDataYUV Class

Purpose
Encapsulates one element supporting Sapera YUV data types

SapDataYUV();
SapDataYUV(int *y*, int *u*, int *v*);
Class constructor, where the *y*, *u*, and *v* arguments specify an initial value other than black

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

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

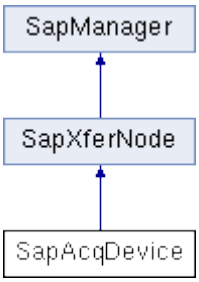
int **U** (read/write property)
Returns the U component of the current value of the data element

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

Demo/Example Usage

Not available

SapAcqDevice



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

Namespace: DALSA.SaperaLT.SapClassBasic

SapAcqDevice Class Members

Construction	
SapAcqDevice	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources
Properties	
AcqDeviceNotifyContext	Application specific data for acquisition events
ConfigFileName	Name of the acquisition configuration file (CCF)
CategoryCount	Number of unique feature category names supported by the acquisition device
CategoryPathNames	Full path names of all unique feature categories supported by the acquisition device
ConfigurationName	Configuration name to be used when saving the device features using the SaveFeatures method
EventCount	Number of events supported by the acquisition device
EventNames	Names of all events supported by the acquisition device
FeatureCount	Number of features supported by the acquisition device
FeatureNames	Names of all features supported by the acquisition device
FileAccessAvailable	Availability of file access by the acquisition device
FileCount	Number of files supported by the acquisition device
FileNames	File names supported by the acquisition device
FlatFieldAvailable	Availability of hardware-based flat-field correction
Label	Text description of the acquisition device
ModeName	Mode name to be used when saving the device features using the SaveFeatures method
RawBayerOutput	Returns whether or not the acquisition device output is raw Bayer
ReadOnly	Checks if the class has read-only access to the acquisition device

UpdateMode	Mode by which features are written to the acquisition device
Methods	
DeleteDeviceFile	Deletes a file from the acquisition device
DisableEvent	Disables the event associated with a specified name or index
EnableEvent	Enables the event associated with a specified name or index
GetCapability	Gets the value of a low-level Sapera capability
GetCapabilityType	Gets the data type of a low-level Sapera capability
GetEventIndexByName	Returns the index of an event associated with a specified name
GetFeatureIndexByName	Returns the index of a feature associated with a specified name
GetFeatureInfo	Returns information on a feature associated with a specified name or index
GetFeatureValue	Returns the value of a feature associated with a specified name or index
GetFileIndexByName	Returns the index of a device file associated with a specified name
GetFileProperty	Gets a property of a specific file on the acquisition device
GetParameter, SetParameter	Gets/sets the value of a low-level Sapera parameter
GetParameterType	Gets the data type of a low-level Sapera parameter
IsCapabilityAvailable	Checks for the availability of a low-level Sapera capability
IsEventAvailable	Returns whether or not an event is supported by the acquisition device
IsEventEnabled	Checks if the event associated with a specified name or index is enabled
IsFeatureAvailable	Returns whether or not a feature is supported by the acquisition device
IsParameterAvailable	Checks for the availability of a low-level Sapera parameter
LoadFeatures	Loads all the features from a configuration file
ReadFile	Reads a file from an acquisition device
SaveFeatures	Saves all (or a subset of) features to a configuration file.
SetFeatureValue	Sets the value of a feature associated with a specified name or index
UpdateFeaturesFromDevice	Gets all the features from the acquisition device at once
UpdateFeaturesToDevice	Sets all the features to the acquisition device at once
UpdateLabel	Updates the device label.
WriteFile	Writes a file to an acquisition device
Events	
AcqDeviceNotify	Notification of acquisition device related events

SapAcqDevice Member Functions

The following are members of the SapAcqDevice Class.

SapAcqDevice.SapAcqDevice (constructor)

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

Parameters

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

Remarks

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

The first three constructors are used when no configuration file is required. In such a case the default parameters of the acquisition device are used. Using the third constructor, you can obtain read-only access to the device. This option is useful only when another application has already obtained a read-write access to the same device.
The fourth constructor allows you to load a configuration file (CCF) previously created by the CamExpert tool or by your own application.

The SapAcqDevice object is used only for storing the acquisition device parameters.

Demo/Example Usage

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

SapAcqDevice.AcqDeviceNotify Event

SapAcqDeviceNotifyHandler **AcqDeviceNotify**

Description

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

The application event handler method is defined as follows:

(for C#)

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

(for VB.NET)

```
Sub AcqDevice_AcqDeviceNotify(ByVal sender As Object, ByVal args As
SapAcqDeviceNotifyEventArgs)
```

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

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.AcqDeviceNotifyContext Property

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

Description

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

Demo/Example Usage

GigE FlatField Demo

SapAcqDevice.CategoryCount

int **CategoryCount** (read-only)

Description

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

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

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

Demo/Example Usage

Not available

SapAcqDevice.CategoryPathNames

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

Description

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

The path names are formatted according to the following rules:

All path names begin with “\Root” or “\SaperaRoot”

Top level categories are returned as “\Root\CategoryName”

Second level categories are returned as “\Root\CategoryName\SubCategoryName”

and so on…

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

Here are examples of how to retrieve this property:

(for C#)

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

(for VB.NET)

```
Dim categoryPathNames() As String = acqDevice.CategoryPathNames
```

Demo/Example Usage

Not available

SapAcqDevice.ConfigFileName Property

string **ConfigFileName** (read/write)

Description

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

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

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

Demo/Example Usage

GigE Metatdata Demo, Camera Compression Demo

SapAcqDevice.ConfigurationName Property

string **ConfigurationName** (read/write)

Description

Configuration name to use when saving the device features with the SaveFeatures method. It is then possible to uniquely identify different configuration files when the company name, camera model name, and mode name are the same. For example, ‘High Contrast’ might be used as configuration name.

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

Demo/Example Usage

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

SapAcqDevice.Create Method

bool **Create();**

Return Value

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

Remarks

Creates all the low-level Sapera resources needed by the acquisition object.

Demo/Example Usage

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

SapAcqDevice.DeleteDeviceFile Method

bool DeleteDeviceFile(string *deviceFileName*);
bool DeleteDeviceFile(int *deviceFileIndex*);

Parameters

deviceFileName	Name of the device file. See the acquisition device User's Manual for the list of supported files.
deviceFileIndex	Index of the file. All indices in the range from 0 to the value returned by the FileCount property, minus 1, are valid.

Remarks

Deletes the specified file on the device.

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

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

Demo/Example Usage

Camera Files Example

SapAcqDevice.Destroy Method

bool **Destroy();**

Return Value

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

Remarks

Destroys all the low-level Sapera resources needed by the acquisition object.

Demo/Example Usage

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

SapAcqDevice.DisableEvent Method

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

Parameters

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

Remarks

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

See the AcqDeviceNotify event for more details.

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.Dispose Method

void **Dispose**();

Remarks

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

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

Demo/Example Usage

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

SapAcqDevice.EnableEvent Method

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

Parameters

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

Remarks

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

See the AcqDeviceNotify event for more details.

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.EventCount Property

int **EventCount** (read-only)

Description

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

Demo/Example Usage

Sequential Grab Demo, Camera Events Example

SapAcqDevice.EventNames Property

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

Description

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

Here are examples of how to retrieve this property:

(for C#)

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

(for VB.NET)

```
Dim eventNames() As String = acqDevice.EventNames
```

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.FeatureCount Property

int **FeatureCount (**read-only)

Description

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

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

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice.FeatureNames Property

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

Description

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

Here are examples of how to retrieve this property:

(for C#)
```
string[] featureNames = acqDevice.FeatureNames;
```

(for VB.NET)
```
Dim featureNames() As String = acqDevice.FeatureNames
```

Demo/Example Usage

GigE FlatField Demo, Camera Events Example, Camera Features Example

SapAcqDevice.FileAccessAvailable Property

bool **FileAccessAvailable** (read-only)

Description

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

Demo/Example Usage

Not available

SapAcqDevice.FileCount Property

int **FileCount** (read-only)

Description

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

Demo/Example Usage

Camera Files Example

SapAcqDevice.FileNames Property

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

Description

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

Here are examples of how to retrieve this property:

(for C#)
```
string[] fileNames = acqDevice.FileNames;
```

(for VB.NET)
```
Dim fileNames() As String = acqDevice.FileNames
```

Demo/Example Usage

Camera Files Example

SapAcqDevice.FlatFieldAvailable Property

bool **FlatFieldAvailable (**read-only)

Description

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

Demo/Example Usage

FlatField Demo

SapAcqDevice.GetCapability Method

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

Parameters

capId Low-level Sapera capability to read

capValue Capability value to read back

Return Value

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

Remarks

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

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

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *capValue* argument. This is not necessary when using the VB.NET language.

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

CORACQDEVICE_CAP_PERSISTENCE becomes SapAcqDevice.Cap.PERSISTENCE

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

Demo/Example Usage

Not available

SapAcqDevice.GetCapabilityType Method

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

Parameters

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

Return Value

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

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice.GetEventIndexByName Method

bool **GetEventIndexByName**(string *eventName*, out int *eventIndex*)

Parameters

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

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

Return Value

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

Remarks

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

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *eventIndex* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Not available

SapAcqDevice.GetFeatureIndexByName Method

bool **GetFeatureIndexByName**(string *featureName*, out int *featureIndex*);

Parameters

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

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

Return Value

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

Remarks

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

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *featureIndex* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Not available

SapAcqDevice.GetFeatureInfo Method

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

Parameters

featureName	Name of the feature. See the acquisition device User's Manual for the list of supported features.
featureIndex	Index of the feature. All indices from 0 to the value returned by the FeatureCount property, minus 1, are valid.
feature	A SapFeature object to store the feature information

Return Value

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

Remarks

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

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

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

Demo/Example Usage

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

SapAcqDevice.GetFeatureValue Method

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

bool **GetFeatureValue**(int *featureIndex*, out int *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out uint *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out long *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out ulong *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out float *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out double *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out bool *featureValue*);
bool **GetFeatureValue**(int *featureIndex*, out string *featureString*);

Parameters

featureName	Name of the feature. See the acquisition device User's Manual for the list of supported features.
featureIndex	Index of the feature. All indices from 0 to the value returned by the FeatureCount property, minus 1, are valid.
featureValue	Returns the value of the specified feature. You must choose the which function overload to use according to the feature type.
featureString	Returns the content of a string feature

Return Value

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

Remarks

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

To find out which overloaded function to use, you must obtain the type of the feature by calling the GetFeatureInfo method, followed by reading the SapFeature.DataType property. To find out if the feature is readable, use the SapFeature.DataAccessMode property.

Note that, except for unitless features, each feature has its specific native unit, for example, milliseconds, KHz, tenth of degree, etc. This information is obtained through the SapFeature.SiUnit and SapFeature.SiToNativeExp10 properties.

Also, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *featureValue* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

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

SapAcqDevice.GetFileIndexByName Method

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

Parameters

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

Return Value

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

Remarks

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

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *fileIndex* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Not available

SapAcqDevice.GetFileProperty Method

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

Parameters

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

Return Value

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

Remarks

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

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

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

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

When calling this method from C#, you need to explicitly specify the ‘out’ keyword as a prefix to the *propertyValue* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Camera Files Example

SapAcqDevice.GetParameter, SapAcqDevice.SetParameter Methods

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

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

Parameters

paramId Low-level Sapera parameter to read or write

paramValue Parameter value to read or write

Return Value

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

Remarks

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

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

When calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *paramValue* argument. This is not necessary when using the VB.NET language.

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

> CORACQDEVICE_PRM_UPDATE_FEATURE_MODE becomes
> SapAcqDevice.Prm.UPDATE_FEATURE_MODE

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

> CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_AUTO becomes
> SapAcqDevice.Val.UPDATE_FEATURE_MODE_AUTO

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

Demo/Example Usage

Not available

SapAcqDevice.GetParameterType Method

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

Parameters

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

Return Value

The returned type may be one of the following:

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

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice.IsCapabilityAvailable Method

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

Parameters

capId Low-level Sapera capability to be checked

Return Value

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

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice.IsEventAvailable Method

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

Parameters

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

Return Value

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

Remarks

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

Demo/Example Usage

GigE FlatField Demo

SapAcqDevice.IsEventEnabled Method

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

Parameters

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

eventIndex Index of the event. Valid values for this index are from 0 to the value returned by the EventCount property, minus 1.

Remarks

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

See the AcqDeviceNotify event for more details.

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDevice.IsFeatureAvailable

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

Parameters

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

Return Value

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

Remarks

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

Demo/Example Usage

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

SapAcqDevice.IsParameterAvailable Method

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

Parameters

paramId Low-level Sapera parameter to be checked

Return Value

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

Remarks

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

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

See the Sapera LT Basic Modules Reference Manual for a description of all parameters and their possible values. Also see the GetParameter method for more information about the syntax to use for paramId.

Demo/Example Usage

Not available

SapAcqDevice.Label Property

string **Label** (read-only)

Description

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

Example: "Genie_HM1400_1 [UserName]"

The part of the label inside the square brackets corresponds to the value of the 'DeviceUserID' feature, which can be modified by the application. When this happens, the label is automatically updated, and the application gets a SapManager.EventType.ResourceInfoChanged event (if registered using the SapManager.EventType property).

Demo/Example Usage

Not available

SapAcqDevice.LoadFeatures Method

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

Parameters

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

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

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

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

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

Demo/Example Usage

Not available

SapAcqDevice.ModeName Property

string **ModeName** (read/write)

Description

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

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

Demo/Example Usage

Not available

SapAcqDevice.RawBayerOutput Property

bool **RawBayerOutput** (read only)

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice.ReadFile Method

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

Parameters

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

Return Value

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

Remarks

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

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

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

Demo/Example Usage

Camera Files Example

SapAcqDevice.ReadOnly Property

bool **ReadOnly** (read/write)

Description

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

Demo/Example Usage

Not available

SapAcqDevice.SaveFeatures Method

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

Parameters

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

Return Value

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

Remarks

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

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

Demo/Example Usage

Not available

SapAcqDevice.SetFeatureValue Method

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

bool **SetFeatureValue**(int *featureIndex*, int *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, uint *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, long *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, ulong *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, float *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, double *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, bool *featureValue*);
bool **SetFeatureValue**(int *featureIndex*, string *featureString*);

Parameters

featureName	Name of the feature. See the acquisition device User's Manual for the list of supported features.
featureIndex	Index of the feature. All indices from 0 to the value returned by the FeatureCount property, minus 1, are valid.
featureValue	Feature value to write. You must choose which function overload to use according to the feature type.
featureString	String feature to write

Return Value

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

Remarks

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

To find out which overloaded function to use, you must obtain the type of the feature by calling the GetFeatureInfo method, followed by reading the SapFeature.DataType property. To find out if the feature is writable, use the SapFeature.DataAccessMode property.

Note that, except for unitless features, each feature has its specific native unit, for example, milliseconds, KHz, tenth of degree, etc. This information is obtained through the SapFeature.SiUnit and SapFeature.SiToNativeExp10 properties.

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

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

Demo/Example Usage

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

SapAcqDevice.UpdateFeaturesFromDevice Method

bool **UpdateFeaturesFromDevice();**

Return Value

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

Remarks

Gets all the features from the acquisition device at once.

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

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

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

Demo/Example Usage

Not available

SapAcqDevice.UpdateFeaturesToDevice Method

bool **UpdateFeaturesToDevice**();

Return Value

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

Remarks

Writes all the features to the acquisition device at once.

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

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

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

Demo/Example Usage

Not available

SapAcqDevice.UpdateMode Property

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

Description

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

UpdateFeatureMode.Auto	New feature values are immediately sent to the acquisition device
UpdateFeatureMode.Manual	New feature values are temporarily cached before being sent to the acquisition device

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

Note, for devices not using the Network Imaging Package (GigE Vision Framework), only UpdateFeature.Auto is implemented; setting the property value to UpdateFeature.Manual has no effect. Consequently, the SapAcqDevice.UpdateFeaturesFromDevice Method and SapAcqDevice.UpdateFeaturesToDevice Method functions are not implemented.

Demo/Example Usage

Not available

SapAcqDevice.UpdateLabel Method

bool **UpdateLabel**();

Return Value

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

Remarks

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

Demo/Example Usage

Not available

SapAcqDevice.WriteFile Method

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

Parameters

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

Return Value

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

Remarks

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

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

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

Demo/Example Usage

Camera Files Example

SapAcqDeviceNotifyEventArgs Class

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

Namespace: DALSA.SaperaLT.SapClassBasic

SapAcqDeviceNotifyEventArgs Class Members

Properties

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

SapAcqDeviceNotifyEventArgs Member Properties

The following are the members of the SapAcqDeviceNotifyEventArgs Class.

SapAcqDeviceNotifyEventArgs.AuxTimeStamp Property

long **AuxTimeStamp** (read-only)

Description

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

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.Context Property

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

Description

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

Demo/Example Usage

Sequential Grab Demo

SapAcqDeviceNotifyEventArgs.CustomData Property

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

Description

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

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

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.CustomSize Property

int **CustomSize (**read-only)

Description

Size of the custom data returned by the CustomData property.

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.EventCount Property

int **EventCount (**read-only)

Description

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

Demo/Example Usage

Sequential Grab Demo

SapAcqDeviceNotifyEventArgs.EventIndex Property

int **EventIndex (**read-only)

Description

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

Demo/Example Usage

Camera Events Example, Camera Features Example

SapAcqDeviceNotifyEventArgs.FeatureIndex Property

int **FeatureIndex** (read-only)

Description

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

Demo/Example Usage

Camera Events Example, Camera Features Example

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

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

Description

Any of the four generic properties supported by some events. You should use aliases instead when they are available. For example, the 'Feature Info Changed' event of the SapAcqDevice class use the FeatureIndex property as an alias to GenericParam0. See the acquisition device User's Manual for a list of events using generic properties.

Demo/Example Usage

Not available

SapAcqDeviceNotifyEventArgs.HostTimeStamp Property

long **HostTimeStamp** (read-only)

Description

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

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

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

Demo/Example Usage

Not available

SapErrorEventArgs

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

Namespace: DALSA.SaperaLT.SapClassBasic

SapErrorEventArgs Class Members

Properties

Code	Low-level Sapera error code associated with the error event
Context	Application context associated with the error event
Message	Error message associated with the error event

SapErrorEventArgs Member Properties

The following are members of the SapErrorEventArgs Class.

SapErrorEventArgs.Code Property

SapStatus **Code** (read-only)

Description

Low-level Sapera error code associated with the call to the application event handler method.

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

CORSTATUS_TIMEOUT becomes SapStatus.TIMEOUT

Demo/Example Usage

Not available

SapErrorEventArgs.Context Property

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

Description

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

Demo/Example Usage

Not available

SapErrorEventArgs.Message Property

string **Message** (read-only)

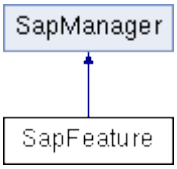
Description

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

Demo/Example Usage

Not available

SapFeature



The purpose of the SapFeature class is to retrieve individual feature information from the SapAcqDevice class. Each feature supported by SapAcqDevice provides a set of capabilities such as name, type, access mode, and so forth, which can be obtained through SapFeature. The GetFeatureInfo method of SapAcqDevice gives access to this information.

Namespace: DALSA.SaperaLT.SapClassBasic

SapFeature Class Members

Construction

SapFeature	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Dispose	Frees unmanaged memory resources

Properties

ArrayLength	Number of bytes required for an array t ype feature
Category	Category to which the current feature belongs
DataAccessMode	Current data access mode for a feature
DataRepresentation	Mathematical representation of a integer or float feature
DataSign	Checks if an integer/float feature is signed or not
DataType	Data type of the feature
DataWriteMode	Checks if a feature can be modified when the transfer object is connected and/or acquiring
Description	Text which represents the full description of the feature
DisplayName	Descriptive name of the feature
EnumCount	Number of possible values for a feature which belongs to an enumerated type
EnumEnabled	Checks if the enumeration value corresponding to a specified index is enabled
EnumText	String values for the enumerated type corresponding to the current feature
EnumValues	Integer values for the enumerated type corresponding to the current feature
FloatDisplayNotation	Gets the notation type to use to display a float type feature
FloatDisplayPrecision	Gets the number of decimal places to display for a float type feature
IsSelector	Determines if the value of a feature directly affects other features
Location	Location where the feature resource is located
Name	Short name of the feature
PollingTime	Interval of time between two consecutive feature updates

SavedToConfigFile	Checks if a feature is saved to a CCF configuration file
SelectedFeatureCount	Number of features associated with a selector
SelectedFeatureIndexes	Indexes of all features associated with a selector
SelectedFeatureNames	Names of all features associated with a selector
SelectingFeatureCount	Number of selectors associated with a feature
SelectingFeatureIndexes	Indexes of all selectors associated with a feature
SelectingFeatureNames	Names of all selectors associated with a feature
SiToNativeExp10	Feature conversion factor from international system (SI) units to native units
SiUnit	Physical units representing the feature in the international system (SI)
Standard	Checks if a feature is standard or custom
ToolTip	Text which represents the explanation of the feature
UserVisibility	Level of visibility assigned to a feature
ValueIncrementType	Type of increment for an integer or floating-point feature
ValidValueCount	Number of valid values for an integer or floating-point feature which defines them as a list

Methods

GetEnumTextFromValue	Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature
GetEnumValueFromText	Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature
GetValidValue	Gets one of a predefined set of valid values for a feature
GetValueIncrement	Gets the minimum acceptable increment for an integer or a float feature
GetValueMin	Gets the minimum acceptable value for a feature
GetValueMax	Gets the maximum acceptable value for a feature

SapFeature Member Functions

The following are members of the SapFeature Class.

SapFeature.SapFeature (constructor)

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

Parameters

location	SapLocation object specifying where the feature is located. The location must be the same as that of the SapAcqDevice object from which the feature is retrieved.

Remarks

The SapFeature constructor does not actually create the low-level Sapera resources. To do this, you must call the SapFeature.Create Method. Upon creation the feature object contents are meaningless. To fill-in a feature object, call the SapAcqDevice.GetFeatureInfo Method function.

Demo/Example Usage

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

SapFeature.ArrayLength Property

int **ArrayLength (**read-only)

Description

Number of bytes required to store the value of an array type featuree, that is, when the value returned by the DataType property is SapFeature.Type.Array.

Demo/Example Usage

Not available

SapFeature.Category Property

string **Category** (read-only)

Description

Category to which the current feature belongs. To simplify the classification of a large set of features from the same SapAcqDevice object, the features are divided into categories. These categories are useful for presenting a list of features in a graphical user interface.

Demo/Example Usage

Not available

SapFeature.Create Method

bool **Create**();

Return Value

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

Remarks

Creates all the low-level Sapera resources needed by the feature object. Call this method before using the object as a parameter to the SapAcqDevice.GetFeatureInfo method.

Demo/Example Usage

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

SapFeature.DataAccessMode Property

SapFeature.AccessMode **DataAccessMode** (read-only)

Description

Current data access mode for a feature, which can be one of the following values:

AccessMode.Undefined	Undefined access mode
AccessMode.RW	The feature may be read and written. Most features are of this type.
AccessMode.RO	The feature can only be read.
AccessMode.WO	The feature can only be written. This is the case for some features which represent commands (or actions) such as 'TimestampReset'.
AccessMode.NP	The feature is not present. The feature is visible in the interface but is not implemented for this device.
AccessMode.NE	The feature is present but currently not enabled. Often used when a feature depends on another feature's value.

Demo/Example Usage

Camera Features Example, Camera Files Example

SapFeature.DataRepresentation Property

SapFeature.Representation **DataRepresentation** (read-only)

Description

Mathematical representation of a integer or float feature, which can be one of the following values:

Representation.Undefined	Undefined representation
Representation.Linear	The feature follows a linear scale
Representation.Logarithmic	The feature follows a logarithmic scale
Representation.Boolean	The feature can have two values: zero or non-zero

Demo/Example Usage

Not available

SapFeature.DataSign Property

SapFeature.Sign **DataSign** (read-only)

Description

Sign of an integer or float feature. This information is useful when reading and writing feature values. By knowing the sign of the feature value you can cast it to the corresponding C/C++ type. It can be one of the following values:

Sign.Undefined	Sign is undefined
Sign.Signed	The feature is a signed integer of float
Sign.Unsigned	The feature is an unsigned integer of float

Demo/Example Usage

Not available

SapFeature.DataType Property

SapFeature.Type **DataType** (read-only)

Description

Data type of a feature, which can be one of the following values:

Sapera Type	Description
Type.Undefined	Undefined type
Type.Int32	32-bit integer
Type.Int64	64-bit integer
Type.Float	32-bit floating-point
Type.Double	64-bit floating-point
Type.Bool	Boolean
Type.Enum	Enumeration
Type.String	ASCII character string
Type.Buffer	Sapera LT buffer object (SapBuffer)
Type.Lut	Sapera LT look-up table object (SapLut)
Type.Array	Sapera LT buffer object (SapBuffer)

If the feature is of array type, then the SapBuffer object should have a height of one line, and a width corresponding to the number of bytes given by the value returned by the ArrayLength property.

Demo/Example Usage

Camera Features Example

SapFeature.DataWriteMode Property

SapFeature.WriteMode **DataWriteMode (**read-only)

Description

Checks if a feature can be modified when the corresponding transfer object (SapTransfer) is connected and/or acquiring. This transfer object is the one which uses the SapAcqDevice object from which the feature object was read.

Some features like buffer dimensions cannot be changed while data is being transfered to the buffer. Use this information to prevent an application from changing certain features when the transfer object is connected and/or acquiring.

Note that this property is only relevant for features which are writable, that is, when the DataAccessMode property identifies the feature as read/write or read-only.

The data write mode can be one of the following values:

WriteMode.Undefined	Undefined write mode
WriteMode.Always	The feature can always be written
WriteMode.NotAcquiring	The feature can only be written when the transfer object is not acquiring. If the transfer is currently acquiring you must stop the acquisition using the SapTransfer.Freeze or SapTransfer.Wait methods before modifying the feature value.
WriteMode.NotConnected	The feature can only be written when the transfer object is not connected. If the transfer is currently connected you must disconnect it using the SapTransfer.Disconnect or SapTransfer.Destroy method before modifying the feature value. After modifying the value reconnect the transfer object using the SapTransfer.Connect or SapTransfer.Create method.

Demo/Example Usage

Not available

SapFeature.Description Property

string **Description** (read-only)

Description

Text which represents the full description of the feature. This information can be used to display detailed textual information in a graphical user interface.

Demo/Example Usage

Not available

SapFeature.Destroy Method

bool **Destroy**();

Return Value

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

Remarks

Destroys all the low-level Sapera resources needed by the feature object.

Demo/Example Usage

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

SapFeature.DisplayName Property

string **DisplayName** (read-only)

Description

Descriptive name of the feature. This name can be used for listing features in a graphical user interface.

Demo/Example Usage

Camera Features Example

SapFeature.Dispose Method

void **Dispose**();

Remarks

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

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

Demo/Example Usage

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

SapFeature.EnumCount Property

int **EnumCount** (read-only)

Description

Number of possible values for a feature which belongs to an enumerated type. Use this property along with the EnumText and EnumValues properties to enumerate all the items contained within an enumeration feature.

Demo/Example Usage

Camera Features Example

SapFeature.EnumEnabled Property

bool **get_EnumEnabled**(int *index*) (read-only)

Parameters

enumIndex Index of the enumeration item (from 0 to the value returned by the EnumCount property, minus 1)

Description

Checks if the enumeration value corresponding to a specified index is enabled

Each item in an enumeration is present for all the application duration. However an enumeration item may be dynamically enabled/disabled according to the value of another feature. Use this property to find out the enable state of an item at any given time.

There is only an indexed version of this property. Here are examples of how to use it for .NET languages:

(for C#)
```
bool enabled = feature.get_EnumEnabled(index);
```

(for VB.NET)
```
Dim enabled as Boolean = feature.EnumEnabled(index)
```

Demo/Example Usage

Not available

SapFeature.EnumText Property

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

Description

String values for the enumerated type corresponding to the current feature. Use this property with the EnumCount and EnumValues properties to enumerate all the items contained within an enumeration feature.

Here are examples of how to retrieve this property:

(for C#)
```
string[] enumText = feature.EnumText;
```

(for VB.NET)
```
Dim enumText() As String = feature.EnumText
```

Demo/Example Usage

Camera Features Example

SapFeature.EnumValues Property

int[] **EnumValues** (read-only)

Description

Integer values for the enumerated type corresponding to the current feature. Use this property along with EnumCount and EnumText properties to enumerate all the items contained within an enumeration feature.

Here are examples of how to retrieve this property:

(for C#)
```
int[] enumValues = feature.EnumValues;
```

(for VB.NET)
```
Dim enumValues() As Integer = feature.EnumValues
```

Demo/Example Usage

Not available

SapFeature.FloatDisplayNotation Property

SapFeature.FloatNotation **FloatDisplayNotation** (read-only)

Description

Gets the notation type to use to display a float type feature. Possible values are:

FloatNotation.Fixed	Display variable using fixed notation. For exampe, 123.4
FloatNotation.Scientific	Display variable using scientific notation. For example, 1.234e-2.
FloatNotation.Undefined	Undefined.

Demo/Example Usage

Not available

SapFeature.FloatDisplayPrecision Property

long **FloatDisplayPrecision** (read-only)

Description

Gets the number of decimal places to display for a float type feature.

Demo/Example Usage

Not available

SapFeature.GetEnumTextFromValue Method

bool **GetEnumTextFromValue**(int *enumValue*, out string *enumText*);

Parameters

enumValue	Value to look for in the enumeration items
enumText	Returned text string

Return Value

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

Remarks

Gets the string value corresponding to a specified integer value for the enumerated type corresponding to the current feature For example you may use this method to retrieve the string corresponding to an enumeration value returned by the SapAcqDevice.GetFeatureValue method.

Note that, when calling this method from C#, you need to explicitly specify the ‘out’ keyword as a prefix to the *enumText* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Camera Features Example

SapFeature.GetEnumValueFromText Method

bool **GetEnumValueFromText**(string *enumText*, out int *enumValue*);

Parameters

enumText	Text string to look for in the enumeration
enumValue	Returned integer value

Return Value

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

Remarks

Gets the integer value corresponding to a specified string value for the enumerated type corresponding to the current feature. For example you may use this method to retrieve the value corresponding to a known enumeration string before calling the SapAcqDevice.SetFeatureValue method.

Note that, when calling this method from C#, you need to explicitly specify the 'out' keyword as a prefix to the *enumValue* argument. This is not necessary when using the VB.NET language.

Demo/Example Usage

Not available

SapFeature.GetValidValue Method

bool **GetValidValue**(int *validValueIndex*, out int *validValue*);
bool **GetValidValue**(int *validValueIndex*, out uint *validValue*);
bool **GetValidValue**(int *validValueIndex*, out long *validValue*);
bool **GetValidValue**(int *validValueIndex*, out ulong *validValue*);
bool **GetValidValue**(int *validValueIndex*, out float *validValue*);
bool **GetValidValue**(int *validValueIndex*, out double *validValue*);

Parameters

validValueIndex	Index of the valid value, can be any value from 0 to the value returned by the ValidValueCount property, minus 1
validValue	Returned valid value, must point to a variable of the same type as the feature

Return Value

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

Remarks

Gets one of a predefined set of valid values for an integer or floating-point feature which defines them as a list, that is, the ValueIncrementType property returns IncrementType.List.

Demo/Example Usage

Camera Features Example

SapFeature.GetValueIncrement Method

bool **GetValueIncrement**(out int *valueIncrement*);
bool **GetValueIncrement**(out uint *valueIncrement*);
bool **GetValueIncrement**(out long *valueIncrement*);
bool **GetValueIncrement**(out ulong *valueIncrement*);
bool **GetValueIncrement**(out float *valueIncrement*);
bool **GetValueIncrement**(out double *valueIncrement*);

Parameters

valueIncrement Returned increment value

Return Value

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

Remarks

Gets the minimum acceptable increment for an integer or a float feature. Some features cannot vary by increments of 1. Their value must be a multiple of a certain increment. For example the buffer cropping dimensions might require to be a multiple of 4 in order to optimize the data transfer.

Demo/Example Usage

Not available

SapFeature.GetValueMax Method

bool **GetValueMax**(out int *valueMax*);
bool **GetValueMax**(out uint *valueMax*);
bool **GetValueMax**(out long *valueMax*);
bool **GetValueMax**(out ulong *valueMax*);
bool **GetValueMax**(out float *valueMax*);
bool **GetValueMax**(out double *valueMax*);

Parameters

maxValue Returned maximum value

Return Value

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

Remarks

Gets the maximum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the integer version to get the maximum length of the string (excluding the trailing null character).

Demo/Example Usage

GigE Camera Compression Demo, Camera Events Example, Camera Features Example, GigE Auto White Balance Example, Grab Camera Link Example

SapFeature.GetValueMin Method

bool **GetValueMin**(out int *valueMin*);
bool **GetValueMin**(out uint *valueMin*);
bool **GetValueMin**(out long *valueMin*);
bool **GetValueMin**(out ulong *valueMin*);
bool **GetValueMin**(out float *valueMin*);
bool **GetValueMin**(out double *valueMin*);

Parameters

minValue Returned minimum value

Return Value

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

Remarks

Gets the minimum acceptable value for a feature. For integer and floating-point types use the version of the method corresponding to the type of the feature. For a string type, use the integer version to get the minimum length of the string (excluding the trailing null character).

Demo/Example Usage

GigE Camera Compression Demo, Camera Events Example, Camera Features Example, GigE Auto White Balance Example, Grab Camera Link Example

SapFeature.IsSelector Property

bool **IsSelector** (read-only)

Description

Determines if the value of the current feature directly affects the values of other features, using a one to many parent-child relationship. For example, if the current feature represents a look-up table index, then the affected features could represent values associated with one specific look-up table.

In this case, the current feature is called the selector.

Use the following properties to find out which features are associated: SelectedFeatureCount, SelectedFeatureIndexes, and SelectedFeatureNames.

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

Demo/Example Usage

Not available

SapFeature.Location Property

SapLocation **Location (**read/write)

Description

Location where the feature resource is located. This location must be the same as that of the corresponding SapAcqDevice object. A specific location can also be specified through the SapFeature constructor.

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

Demo/Example Usage

Not available

SapFeature.Name Property

string **Name** (read-only)

Description

Short name of the feature. This name can be used with SapAcqDevice methods which expect a feature name. This string should not be used for display in a graphical user interface. Use the DisplayName property instead to provide a more descriptive name.

Demo/Example Usage

Not available

SapFeature.PollingTime Property

int **PollingTime (**read-only)

Description

Interval of time between two consecutive feature updates. Some read-only features (such as 'InternalTemperature') are read internally from the acquisition device at a certain frequency in order to always stay up to date.

Note that this property is only relevant for acquisition devices which are supported through the Network Imaging Package (GigE Vision Framework). Other devices do not return a polling time, but instead use internal polling that generates "Feature Info Changed" events whenever required.

Demo/Example Usage

Not available

SapFeature.SavedToConfigFile Property

bool **SavedToConfigFile** (read/write)

Description

Checks if a feature is saved to a CCF configuration file when calling the SapAcqDevice.SaveFeatures method.

All features are assigned a default behavior. For example, the read-only features are not saved while the read/write features are. You can, however, change the default behavior. For example a read-only feature such as 'InternalTemperature' is not saved by default. You can set this property to **true** to force the feature to be written to the configuration file.

If you force read-only features to be saved those features will not be restored when loading back the CCF file. The reason is that the features are not writable to the device.

For acquisition devices which are not supported through the Network Imaging Package (GigE Vision Framework), the features saved to the configuration file are hardcoded and cannot be changed. Therefore writing this property has no effect.

Demo/Example Usage

Not available

SapFeature.SelectedFeatureCount Property

int **SelectedFeatureCount (**read-only)

Description

Number of features associated with a selector (value of IsSelector property is **true**), or 0 if the current feature is not a selector. These selected features can be considered as children of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature.SelectedFeatureIndexes Property

int[] **SelectedFeatureIndexes** (read-only)

Description

Indexes of all features associated with a selector (value of IsSelector property is **true**). The indexes are relative to the acquisition device, with possible values from 0 to the value returned by the SapAcqDevice.FeatureCount proeprty, minus 1. These features can be considered as the children of the current SapFeature object.

The SelectedFeatureCount property returns the number of features associated with the selector.

The indexes returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
(for C#)
int[] selectedFeatureIndexes = feature.SelectedFeatureIndexes;

(for VB.NET)
Dim selectedFeatureIndexes() As Integer = feature. SelectedFeatureIndexes
```

Demo/Example Usage

Not available

SapFeature.SelectedFeatureNames Property

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

Description

Names of all features associated with a selector (value of IsSelector property is **true**). These features can be considered as the children of the current SapFeature object.

The SelectedFeatureCount property returns the number of features associated with the selector.

The names returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
(for C#)
string[] selectedFeatureNames = feature.SelectedFeatureNames;

(for VB.NET)
Dim selectedFeatureNames() As String = feature. SelectedFeatureNames
```

Demo/Example Usage

Not available

SapFeature.SelectingFeatureCount Property

int **SelectingFeatureCount (**read-only)

Description

Number of selectors (value of IsSelector property is **true**) associated with a feature, or 0 if there are no associated selectors. These selectors can be considered as parents of the current SapFeature object.

Demo/Example Usage

Not available

SapFeature.SelectingFeatureIndexes Property

int[] **SelectingFeatureIndexes** (read-only)

Description

Indexes of all selectors (value of IsSelector property is **true**) associated with a feature. The indexes are relative to the acquisition device, with possible values from 0 to the value returned by the SapAcqDevice.FeatureCount proeprty, minus 1. These selectors can be considered as the parents of the current SapFeature object.

The SelectingFeatureCount property returns the number of selectors associated with the feature.

The indexes returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
(for C#)
int[] selectingFeatureIndexes = feature.SelectingFeatureIndexes;

(for VB.NET)
Dim selectingFeatureIndexes() As Integer = feature.SelectingFeatureIndexes
```

Demo/Example Usage

Not available

SapFeature.SelectingFeatureNames Property

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

Description

Names of all selectors (value of IsSelector property is **true**) associated with a feature. These selectors can be considered as the parents of the current SapFeature object.

The SelectingFeatureCount property returns the number of selectors associated with the feature.

The names returned by this property can be used by the SapAcqDevice.GetFeatureInfo method to access the corresponding SapFeature object. The SapAcqDevice.FeatureCount property returns the number of features supported by the acquisition device.

Here are examples of how to retrieve this property:

```
(for C#)
string[] selectingFeatureNames = feature.SelectingFeatureNames;

(for VB.NET)
Dim selectingFeatureNames() As String = feature.SelectingFeatureNames
```

Demo/Example Usage

Not available

SapFeature.SiToNativeExp10 Property

int **SiToNativeExp10** (read-only)

Description

Gets the base 10 exponent (positive or negative) for converting the value of a feature from international system (SI) units to native units (the units used to read/write the feature through the API). The following equation describes the relation between the two unit systems:

VNATIVE = VSI * 10E

Where *V* is the value of a feature and *E* is the current parameter.

Example 1

You want to set the camera exposure time to a known value in seconds. The 'ExposureTime' feature is represented in microseconds. Therefore the current exponent value is 6. If the desired integration time is 0.5 second, then you can compute the actual value for the SapAcqDevice.SetFeatureValue method as follows:
VNATIVE = 0.5*106 = 500000

Example 2

You want to monitor the temperature of the camera sensor. The 'InternalTemperature' feature is reported in degrees Celcius. Therefore the current exponent value is 0. If the feature value returned by the SapAcqDevice.GetFeatureValue method is 50 then the temperature in Celcius is also equal to 50.

Use the SiUnit property to retrieve the international system (SI) units corresponding to the feature to monitor.

Demo/Example Usage

Camera Events Example, GigE Auto-White Balance Example

SapFeature.SiUnit Property

string **SiUnit (**read-only)

Description

Physical units representing the feature in the international system (SI). Examples of units are Volts, Pixels, Celsius, Degrees, etc. This information is useful to present in a graphical user interface.

Most of the time the units used by the feature (the native units) are NOT the same as SI units, but rather a multiple of them. For example, the exposure time may be represented in microseconds instead of seconds. To convert the feature value to the SI units you must use the exponent value provided by the SiToNativeExp10 property.

Demo/Example Usage

Not available

SapFeature.Standard Property

bool **Standard (**read-only)

Description

Checks if a feature is standard or custom. Most of the features are standard. However, sometimes custom features might be provided as part of a special version of an acquisition device driver.

Demo/Example Usage

Not available

SapFeature.ToolTip Property

string **ToolTip** (read-only)

Description

Text which represents the explanation of the feature. This information can be used to implement tool tips in a graphical user interface.

Demo/Example Usage

Not available

SapFeature.UserVisibility Property

SapFeature.Visibility **UserVisibility** (read-only)

Description

Level of visibility assigned to a feature. This information is useful to classify the features in a graphical user interface in terms of user expertise. It can be one of the following values:

Visibility.Undefined	Undefined visibility level
Visibility.Beginner	The feature should be made visible to any user
Visibility.Expert	The feature should be made visible to users with a certain level of expertise
Visibility.Guru	Specifies that the feature should be made visible to users with a high level of expertise
Visibility.Invisible	The feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features

Demo/Example Usage

Not available

SapFeature.ValueIncrementType Property

SapFeature.IncrementType **ValueIncrementType** (read-only)

Description

Type of increment for an integer or floating-point feature. This is useful for finding out which values are valid for this feature. It can be one of the following values:

IncrementType.Undefined	Undefined increment type. This normally means that the acquisition device to which the feature is associated does not support reading the value of the increment type.
IncrementType.None	The feature has no increment. Use the GetValueMin and GetValueMax functions to find out the feature value limits.
IncrementType.Linear	The feature has a fixed increment. Use the GetValueMin and GetValueMax functions to find the feature value limits, and GetValueIncrement to find the increment.
IncrementType.List	The feature has a fixed set of valid values. Use the ValidValueCount property to find the number of values, and the GetValidValue function to enumerate them.

Demo/Example Usage

Camera Features Example

SapFeature.ValidValueCount Property

int **ValidValueCount** (read-only)

Description

Number of valid values for an integer or floating-point feature which defines them as a list, that is, the ValueIncrementType property returns IncrementType.List. In this case, use the GetValidValue function to enumerate these values.

Demo/Example Usage

Camera Features Example

SapLocation

The SapLocation Class identifies a Sapera server/resource pair.

A Sapera server is an abstract representation of a physical device like a frame grabber, a processing board, a GigE camera, or the host computer. In general, a Teledyne DALSA board or GigE camera camera is a server. Resources are attached to these physical devices. For example, a frame grabber can have one or more acquisition resources.

Sapera Class methods do not always need the server information from SapLocation. In these cases, the resource index is simply ignored.

Namespace: DALSA.SaperaLT.SapClassBasic

SapLocation Class Members

Construction	
SapLocation	Class constructor
Dispose	Frees unmanaged memory resources
Properties	
ResourceIndex	Resource index
ServerIndex	Server index
ServerName	Server name

SapLocation Member Functions

The following are members of the SapLocation Class.

SapLocation.SapLocation (constructor)

SapLocation();
SapLocation(int *serverIndex*, int *resourceIndex*);
SapLocation(string *serverName*, int *resourceIndex*);

Parameters

serverIndex	Sapera server index. There is always one server associated with the host computer at SapLocation.ServerSystem (index 0).
serverName	Sapera server name. The 'System' server is associated with the host computer.
resourceIndex	Sapera resource index

Remarks

Use the Sapera Configuration utility to find the names and indices of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, IO Demo, GigE Auto-White Balance Example,. GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapLocation.Dispose Method

void **Dispose**();

Remarks

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

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

Demo/Example Usage

GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, IO Demo, GigE Auto-White Balance Example,. GigE Camera LUT Example, Grab CameraLink Example, Grab Console Example, Grab LUT Example

SapLocation.ResourceIndex Property

int **ResourceIndex** (read-only)

Remarks

Resource index.

Demo/Example Usage

Not available

SapLocation.ServerIndex Property

int **ServerIndex** (read-only)

Remarks

Server index. If the returned value is equal to SapLocation.ServerIndexUnknown, it does not necessarily mean that the object is invalid. In this case, use the ServerName property instead.

Demo/Example Usage

Not available

SapLocation.ServerName Property

string **ServerName (**read-only)

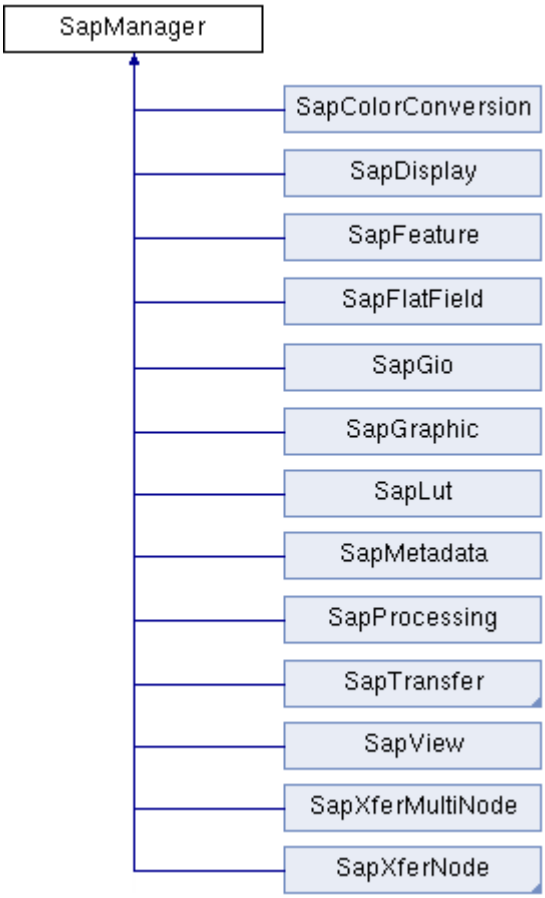
Remarks

Server name. If the returned value is an empty string, it does not necessarily mean that the object is invalid. In this case, use the ServerIndex property instead.

Demo/Example Usage

Not available

SapManager



The SapManager Class includes methods for describing the Sapera resources present on the system. It also includes error management capabilities.

With .NET, this class is declared abstract, which means it cannot be explicitly instantiated.

Namespace: DALSA.SaperaLT.SapClassBasic

SapManager Class Members

Properties

CommandTimeout	Timeout value used when waiting for completion of Sapera LT commands
DisplayStatusMode	Global reporting mode for messages and errors
EndResetContext	Application specific data for end of reset events
ErrorContext	Application specific data for error events
Initialized	Checks whether the Create method has succeeded for a derived object
LastStatusCode	Numeric value of the latest Sapera low-level error
LastStatusMessage	Description of the latest Sapera low-level error
ResetTimeout	Timeout value used when resetting a hardware device
ServerEventType	Currently registered event type for server related events
ServerNotifyContext	Application specific data for server related events
VersionInfo	Sapera LT version and licensing information

Methods

DetectAllServers	Detects GenCP cameras after a Sapera application has been started
DisplayMessage	Reports a custom message using the current reporting mode
Dispose	Frees unmanaged memory used internally by a Sapera .NET object
GetFormatType	Gets the data type corresponding to a Sapera data format
GetInstallDirectory	Gets the directory where a Sapera product is installed
GetResourceCount	Gets the number of Sapera resources of a specific type on a server
GetResourceIndex	Gets the index of a Sapera resource
GetResourceName	Gets the name of a Sapera resource
GetSerialNumber	Gets the serial number corresponding to a Sapera server
GetServerCount	Gets the number of available Sapera servers
GetServerIndex	Gets the index of a Sapera server
GetServerName	Gets the name of a Sapera server
GetServerType	Gets the type of a Sapera server
IsResourceAvailable	Checks whether a resource is available for use
IsSameLocation	Checks whether two SapLocation objects are the same
IsSameServer	Checks whether two SapLocation objects are located on the same server
IsServerAccessible	Checks if the resources for a server are accessible
IsSystemLocation	Checks whether a SapLocation object is located on the system server
ResetServer	Resets the hardware device associated with a specific server
WriteFile	Writes a file to non-volatile memory on the device

Events

EndReset	Notification that a server reset operation is finished
Error	Notification of Sapera LT .NET library errors
ServerNotify	Notification of server related events (except reset)
ServerFileNotify	Notification of file transfer event

SapManager Member Functions

The following are members of the SapManager Class.

SapManager.CommandTimeout Property

static int **CommandTimeout** (read/write)

Remarks

Timeout value (in milliseconds) used when waiting for completion of Sapera LT commands. The initial value for this property is 20000 (20 seconds).

If you need to control the timeout value used by the ResetServer method, use the ResetTimeout property instead.

Demo/Example Usage

GigE Sequential Grab Demo, Sequential Grab Demo, Camera Files Example

SapManager.DetectAllServers Method

static bool **DetectAllServers**(DetectServerType *type*);

Parameters

type	Specifies the type of server to detect. Possible values are:	
	DetectServerType.GenCP	Detect GenCP servers only.
	DetectServerType.All	Currently equivalent to GenCP only.

Remarks

Use this function to detect GenCP cameras after a Sapera application has been started. In a typical application device detection (discovery) is initiated during application startup. If a GenCP camera is connected after an application has been launched, it will not be detected automatically. Use this function to trigger the device discovery process.

Note that you must register the EventType.ServerNew event before calling this function. See the SapManager.ServerEventType property for details.

Demo/Example Usage

Find Camera example

SapManager.DisplayMessage Method

static void **DisplayMessage**(string *message*);

Parameters

message	Custom message to report

Remarks

Reports a custom message using the current reporting mode. See the DisplayStatusMode property for a description of the available reporting modes.

Note that, when the reporting mode is set to the Log Viewer, messages are logged as errors. It is possible to log these as informational instead by using the case insensitive '(Sapera app)' prefix at the beginning of each message. This prefix will be automatically stripped from the message before logging.

Demo/Example Usage

Not available

SapManager.DisplayStatusMode Property

static SapManager.StatusMode **DisplayStatusMode** (read/write)

Description

Global reporting mode for messages and errors. This mode is used by the DisplayMessage method, and also internally by the Sapera .NET library. It can be one of the following values:

StatusMode.Popup	Sends messages to a popup window
StatusMode.Log	Sends messages to the Sapera Log Server (can be displayed using the Sapera Log Viewer)
StatusMode.Event	Notifies application code through an event
StatusMode.Exception	Notifies application code through an exception
StatusMode.Custom	Error information is not reported, it is just stored internally

The initial value for this property is Popup.

For Event reporting mode, the SapManager.Error event is invoked whenever an error occurs. Also see the description of this event and of the ErrorContext property.

For Exception reporting mode, you get an exception of type SapLibraryException whenever an error occurs. This type is derived from SapException, which is the base type for all Sapera LT .NET exceptions This type is in turn derived from the .NET System.Exception type.

For Custom reporting mode, the only way to retrieve error information is by reading the value of the LastStatusCode and/or LastStatusMessage properties.

Note that, for all reporting modes except Exception, any Sapera .NET method returns **false** to indicate an error.

Demo/Example Usage

Camera Features Example, Find Camera Example

SapManager.Dispose Method

Description

Frees unmanaged memory used internally by a Sapera .NET object.

The following classes contains Dispose methods derived from the SapManager class:

- SapAcqDevice.Dispose Method
- SapFeature.Dispose Method
- SapLocation.Dispose Method

Demo/Example Usage

All demos and examples

SapManager.EndReset Event

static SapResetHandler **EndReset**

Description

Notifies the application that a server reset operation is finished. This operation is initiated by calling the ResetServer method. Use the EndResetContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

(for C#)

```
static void SapManager_EndReset(Object sender, SapResetEventArgs args)
```

(for VB.NET)

```
Sub SapManager_EndReset(ByVal sender As Object, ByVal args As SapResetEventArgs)
```

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

Demo/Example Usage

Not available

SapManager.EndResetContext Property

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

Description

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

Demo/Example Usage

Not available

SapManager.Error Event

static SapErrorHandler **Error**

Description

Notifies the application that Sapera.NET library error has occurred as a result of calling one of its methods. Use the ErrorContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

(for C#)

```
static void SapManager_Error(Object sender, SapErrorEventArgs args)
```

(for VB.NET)

```
Sub SapManager_Error(ByVal sender As Object, ByVal args As SapErrorEventArgs)
```

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

Demo/Example Usage

Not available

SapManager.ErrorContext Property

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

Description

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

Demo/Example Usage

Not available

SapManager.GetFormatType Method

static SapFormatType **GetFormatType**(SapFormat *format*);

Parameters

format Sapera data format

Remarks

Gets the data type corresponding to the specified Sapera data format as one of the following values:

SapFormatType.Unknown	Unable to determine data type
SapFormatType.Mono	Monochrome
SapFormatType.RGB	RGB color
SapFormatType.YUV	YUV color
SapFormatType.HSI	HSI color
SapFormatType.HSV	HSV color
SapFormatType.Color	Lookup table color data
SapFormatType.RGBA	RGB color with an additional component (alpha channel, infrared component, etc.)

Demo/Example Usage

Not available

SapManager.GetInstallDirectory Method

static string **GetInstallDirectory**(int *serverIndex*);
static string **GetInstallDirectory** (string *serverName*);
static string **GetInstallDirectory** (SapLocation *location*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
location	Valid SapLocation object

Remarks

Gets the directory where a Sapera product is installed.

For the system server, this corresponds to the Sapera installation directory, for example, **c:\Program Files\Teledyne DALSA\Sapera**.

For a server corresponding to a hardware device, this corresponds to the directory where the driver for the device is installed, .for example, **c:\Program Files\Teledyne DALSA\X64 Xcelera-CL PX4**.

Demo/Example Usage

Not available

SapManager.GetResourceCount Method

static int **GetResourceCount**(int *serverIndex*, SapManager.ResourceType *resourceType*);
static int **GetResourceCount**(string *serverName*, SapManager.ResourceType *resourceType*);
static int **GetResourceCount**(SapLocation *loc*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
serverName	Sapera server name

Remarks

Gets the number of resources of a specified type on a Sapera server. This only applies to static resources, that is, those attached to physical devices. Dynamic resources, like buffers, do not have a fixed count.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

IO Demo, Find Camera Example, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

SapManager.GetResourceIndex Method

static int **GetResourceIndex**(int *serverIndex*, SapManager.ResourceType *resourceType*, string *resourceName*);
static int **GetResourceIndex**(string *serverName*, SapManager.ResourceType *resourceType*, string *resourceName*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceName	Sapera resource name
serverName	Sapera server name

Remarks

Gets the index of a Sapera resource. Returns SapLocation.ResourceUnknown if the specified resource cannot be found.

The first form of this method looks for the resource of the specified name and type on the server specified by *index*. The second form uses the server name instead of the index.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server and resource names for that product.

Demo/Example Usage

Not available

SapManager.GetResourceName Method

static string **GetResourceName**(int *serverIndex*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static string **GetResourceName**(string *serverName*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static string **GetResourceName**(SapLocation *loc*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

Remarks

Gets the name of a Sapera resource of a specified type.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

Demo/Example Usage

Not available

SapManager.GetSerialNumber Method

static string **GetSerialNumber**(int *serverIndex*);
static string **GetSerialNumber**(string *serverName*);
static string **GetSerialNumber**(SapLocation *location*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object

Remarks

Gets a text representation of the serial number corresponding to the hardware device for the specified Sapera server. It consists of either the letter 'S' or 'H' followed by seven digits, for example, "S1234567".
Note that there is no serial number associated with the System server. Also, this function is only supported for frame grabbers and older Genie cameras (not Genie-TS). When using other camera servers (GigE-Vision or GenCP), you need a valid SapAcqDevice object from which the serial number can be retrieved through a named feature.

Demo/Example Usage

Not available

SapManager.GetServerCount Method

static int **GetServerCount**();
static int **GetServerCount**(SapManager.ResourceType *resourceType*);

Parameters

resourceType	Resource type to inquire, can be one of the following:	
	ResourceType.Acq	Frame grabber acquisition hardware
	ResourceType.AcqDevice	Camera acquisition hardware (for example, Genie)
	ResourceType.Display	Physical displays
	ResourceType.Gio	General inputs and outputs
	ResourceType.Graphic	Graphics engine

Remarks

Gets the number of available Sapera servers.

The first form of this method considers all servers, regardless of their resource type. In this case, the return value is at least 1, since the system server is always present. The second form returns the number of servers for the specified resource type only, so the return value may be equal to 0.

Demo/Example Usage

IO Demo, Find Camera Example

SapManager.GetServerIndex Method

static int **GetServerIndex**(string *serverName*);
static int **GetServerIndex**(SapLocation *location*);

Parameters

serverName	Sapera server name
location	Valid SapLocation object

Remarks

Gets the index of a Sapera server. Returns SapLocation.ServerIndexUnknown if the specified server cannot be found.

The first form of this method uses the server name. The second form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

IO Demo, GigE Auto-White Balance Example, GigE Camera LUT Example, Grab Console Example, Grab LUT Example

SapManager.GetServerName Method

static string **GetServerName**(int *serverIndex*);
static string **GetServerName**(SapLocation *loc*);
static string **GetServerName**(int *serverIndex*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Sapera server index
loc	Valid SapLocation object
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.

Remarks

Gets the name of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses an existing SapLocation object with valid server information. The third form only considers servers with at least one resource of the specified type. For example, index 1 corresponds to the second server with at least one acquisition device.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

Dual Acquisition Demo, IO Demo, Find Camera Example

SapManager.GetServerType Method

static SapManager.Server **GetServerType**(int *serverIndex*);
static SapManager.Server **GetServerType**(string *serverName*);

Parameters

serverIndex	Sapera server index	
serverName	Sapera server name	
Return Value	Can be one of the following:	
	Server.None	Server type cannot be determined
	Server.System	System server
	Server.Bandit3CV	Bandit-3 CV Express VGA frame grabber
	Server.X64CL	X64-CL acquisition board
	Server.X64CLiPRO	X64-CL iPro acquisition board
	Server.X64CLExpress	X64-CL-Express acquisition board
	Server.X64CLLX4	X64 Xcelera-CL LX4 acquisition board
	Server.X64CLPX4	X64 Xcelera-CL PX4 acquisition board
	Server.X64LVDS	X64-LVDS acquisition board
	Server.X64LVDSPX4	X64-LVDSPX4 acquisition board
	Server.X64LVDSVX4	X64-LVDSVX4 acquisition board
	Server.X64ANQuad	X64-AN Quad acquisition board
	Server.X64ANLX1	X64-ANLX1 acquisition board
	Server.PC2Vision	PC2-Vision acquisition board
	Server.PC2Comp	PC2-Comp Express acquisition board
	Server.PC2CamLink	PC2-CamLink acquisition board
	Server.Genie	Genie camera
	Server.AnacondaCL	Anaconda-CL vision processor
	Server.AnacondaLVDS	Anaconda-LVDS vision processor

Remarks

Gets the type of a Sapera server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Demo/Example Usage

Not available

SapManager.Initialized Property

bool **Initialized** (read-only)

Remarks

Checks whether the Create method has succeeded for an object derived from SapManager.

Calling the Destroy method resets this property to **false**.

Demo/Example Usage

Color Conversion Demo, FlatField Demo, Dual Acquisition Demo, GigE Camera Demo, GigE FlatField Demo, GigE Sequential Grab Demo, Grab Demo, Sequential Grab Demo, GigE Auto-White Balance Example,

SapManager.IsResourceAvailable Method

static bool **IsResourceAvailable**(int *serverIndex*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static bool **IsResourceAvailable**(string *serverName*, SapManager.ResourceType *resourceType*, int *resourceIndex*);
static bool **IsResourceAvailable**(SapLocation *location*, SapManager.ResourceType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
location	Valid SapLocation object

Return Value

Returns **true** if the specified resource is not already used, **false** otherwise

Remarks

Determines if a specific Sapera resource on a server is available. You may use this method, for example, before calling the SapAcqDevice.Create Method to avoid getting an error when the acquisition resource is already in use.

The first form of this method uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1. The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

Demo/Example Usage

Not available

SapManager.IsSameLocation Method

static bool **IsSameLocation**(SapLocation *firstLocation*, SapLocation *secondLocation*);

Parameters

firstLocation First valid SapLocation object

secondLocation Second valid SapLocation object

Remarks

Checks if the two specified SapLocation objects have the same server and resource information

Demo/Example Usage

Not available

SapManager.IsSameServer Method

static bool **IsSameServer**(SapLocation *firstLocation*, SapLocation *secondLocation*);

Parameters

firstLocation First valid SapLocation object

secondLocation Second valid SapLocation object

Remarks

Checks if the two specified SapLocation objects have the same server information

Demo/Example Usage

IsSameServer

SapManager.IsServerAccessible Method

static bool **IsServerAccessible**(int *serverIndex*);
static bool **IsServerAccessible**(string *serverName*);
static bool **IsServerAccessible**(SapLocation *location*);

Parameters

serverIndex	Index of Sapera server containing the resource
serverName	Name of Sapera server containing the resource
location	Valid SapLocation object

Return Value

Returns **true** if the resources for the server are accessible, **false** otherwise

Remarks

Checks if the resources belonging to a server are currently accessible. Although existing objects for these resources are still valid when their server becomes unaccessible, they must be left alone or destroyed (for example, SapAcqDevice.Destroy).

When a Sapera application starts, all detected servers are automatically accessible. However, Sapera camera devices (GigE-Vision and GenCP) can be connected and disconnected while a Sapera application is running. When such a device is connected for the first time, its server is automatically accessible. When the device is later disconnected, the server becomes unaccessible. If it is reconnected again, the server is once again accessible.

The first form of this method uses a server index. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object.

You may also determine accessibility of servers by registering an event handler for the ServerNotify event, and specifying the event type using the ServerEventType property.

Note that you should not use this method for devices which are always connected (for example, frame grabbers), since the return value may not correspond to the actual resource accessibility for the corresponding server.

Demo/Example Usage

Not available

SapManager.IsSystemLocation Method

static bool **IsSystemLocation**();
static bool **IsSystemLocation**(SapLocation *location*);

Parameters

location	Valid SapLocation object

Remarks

Check if the current application is running on the system server, or if the SapLocation object refers to this server.

Demo/Example Usage

Not available

SapManager.LastStatusCode Property

static SapStatus **LastStatusCode** (read-only)

Description

Numeric value of the latest Sapera low-level error. See the DisplayStatusMode property for a description of the available error reporting modes.

Note that each thread in a Sapera LT application has its own latest error code. This means that you cannot read this property to retrieve error information for a Sapera .NET function which has been called in another thread.

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

Demo/Example Usage

Not available

SapManager.LastStatusMessage Property

static string **LastStatusMessage** (read-only)

Description

Description of the latest Sapera low-level error. See the DisplayStatusMode property for a description of the available error reporting modes.

Note that each thread in a Sapera LT application has its own latest error description. This means that you cannot read this property to retrieve error information for a Sapera .NET function which has been called in another thread.

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

Demo/Example Usage

Not available

SapManager.ResetServer Method

static bool **ResetServer**(int *serverIndex*, bool *wait*);
static bool **ResetServer**(string *serverName*, bool *wait*);
static bool **ResetServer**(SapLocation *loc*, bool *wait*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
wait	Specifies whether this methgd should return immediately after resetting the specified server, or if it should wait for the server to be operational again

Return Value

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

Remarks

Resets the hardware frame grabber associated with a specific server.

The first form of this method uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

There are two ways to use this method:

wait = **true**	Returns only when the reset is complete, and the server is operational again
wait = **false** (EndReset event is not registered)	Returns immediately after resetting the server. The application program is then responsible for figuring out when the server is operational again.
wait = **false** (EndReset event is registered)	Returns immediately after resetting the server, and notifies the application by invoking the EndReset event when the server is operational again

You can read the values of ServerIndex and Context methods of the SapResetEventArgs class to retrieve the required information from the application event handler method.

Note that all other Sapera .NET objects must be destroyed before calling this method, otherwise application behavior is undefined. To reset the server, use the following sequence:

- Call Destroy on all Sapera .NET objects (SapTransfer, SapBuffer, SapAcquisition, ...)
- Call ResetServer
- Call Create for all needed Sapera .NET objects

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

Note: this method is only for use with frame grabbers; for cameras use the *DeviceReset* feature to reset the device.

Demo/Example Usage

Not available

SapManager.ResetTimeout Property

static int **ResetTimeout** (read/write)

Description

Timeout value (in milliseconds) used when resetting a hardware device. This value is used by the ResetServer method.

If you need to control the timeout value used when waiting for completion of Sapera LT commands, use the CommandTimeout property instead.

The initial value for this attribute is 20000 (20 seconds).

Demo/Example Usage

Not available

SapManager.ServerEventType Property

static SapManager.EventType **ServerEventType** (read-write)

Description

Registered event type for the ServerNotify event. One or more of the following values may be combined together using a bitwise OR operation:

EventType.ServerNew	A new device is connected while a Sapera application is already running
EventType.ServerDisconnected	The device corresponding to an existing server is disconnected. (Replaces EventType.ServerNotAccessible which is now deprecated.)
EventType.ServerConnected	The device corresponding to an existing, unaccessible server is reconnected. (Replaces EventType.ServerAccessible, which is now deprecated.)
EventType.ServerDatabaseFull	There is no room in the Sapera server database for a new device that has just been connected
EventType.ResourceInfoChanged	The information describing a resource (typically its label) has changed
EventType.ServerFile	The information about the progress of the file being transferred

In the event handler method, you can get the event type through the EventType property of the *args* argument. For all events except ServerDatabaseFull, you can get the index of the server through the ServerIndex property. For the ResourceInfoChanged event, you can get the index of the affected resource through the ResourceIndex property. For all events, you can access application specific data (originally specified by the SapManager.ServerNotifyContext property) through the Context property.

The ResourceInfoChanged event can only occur as a result of modifying the value of the 'DeviceUserID' feature through the SapAcqDevice class.

Note that server related events are only available when dealing with Sapera camera devices (GigE-Vision and GenCP), that can be connected and disconnected while a Sapera application is running.

Demo/Example Usage

Not available

SapManager.ServerFileNotify Event

static SapServerFileNotifyHandler **ServerFileNotify**

Description

Notifies the application of server related file transfer events. Use the ServerEventType property to set the events that the application needs to be notified of. Use the ServerNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

(for C#)

```
static void SapManager_ServerFileNotify(Object sender, SapServerNotifyEventArgs
args)
```

(for VB.NET)

```
Sub SapManager_ServerFileNotify(ByVal sender As Object, ByVal args As
SapServerNotifyEventArgs)
```

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

Demo/Example Usage

Not available

SapManager.ServerNotify Event

static SapServerNotifyHandler **ServerNotify**

Description

Notifies the application of server related events (except reset, which occurs through the EndReset event). Use the ServerEventType property to set the events that the application needs to be notified of. Use the ServerNotifyContext property to supply application specific data when the application event handler method is invoked. This data is then available through the Context property of the *args* argument.

The application event handler method is defined as follows:

(for C#)

```
static void SapManager_ServerNotify(Object sender, SapServerNotifyEventArgs args)
```

(for VB.NET)

```
Sub SapManager_ServerNotify(ByVal sender As Object, ByVal args As
SapServerNotifyEventArgs)
```

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

Demo/Example Usage

Not available

SapManager.ServerNotifyContext Property

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

Description

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

Demo/Example Usage

Not available

SapManager.VersionInfo Property

static SapManVersionInfo **VersionInfo** (read-only)

Description

Sapera LT version and licensing information. The SapManVersionInfo object contains the following read-only properties:

Major	Major version number. For example, if the version number is 6.30.01.0806, the value of this property is 6.
Minor	Minor version number. For example, if the version number is 6.30.01.0806, the value of this property is 30.
Revision	Revision number. For example, if the version number is 6.30.01.0806, the value of this property is 1.
Build	Build number. For example, if the version number is 6.30.01.0806, the value of this property is 806.
LicenseType	Sapera LT license type for the current installation, which can be one of LicenseType.Runtime, LicenseType.Evaluation, or LicenseType.FullSDK
EvalDaysRemaining	Number of days remaining in the evaluation period when the license type is LicenseType.Runtime, where a value equal to 0 means that the evaluation period has expired.

Demo/Example Usage

Not available

SapManager.WriteFile Method

static bool **WriteFile**(int *serverIndex*, String *localFilePath*, int *deviceFileIndex*);
static bool **WriteFile**(String *serverName*, String *localFilePath*, int *deviceFileIndex*);
static bool **WriteFile**(SapLocation *loc*, String *localFilePath*, int *deviceFileIndex*);

Parameters

serverIndex	*Sapera server index*
serverName	Sapera server name
loc	Valid SapLocation object
deviceFileName	Name of the device file. See the acquisition device User's Manual for the list of supported files.
deviceFileIndex	Index of the file. All indices in the range from 0 to the value returned by the GetFileCount method, minus 1, are valid.
localFilePath	Full directory path and filename on the host computer to save the file.

Return Value

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

Remarks

Writes a file to non-volatile memory on the device. Refer to the acquisition device's User's Manual for the list of supported files.

Demo/Example Usage

Camera Files Example

SapResetEventArgs

The SapResetEventArgs class contains the arguments to the application handler method for the SapManager.EndReset event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapResetEventArgs Class Members

Properties

Context	Application context associated with server reset events
ServerIndex	Sapera server index associated with the invocation of the application event handler.

SapResetEventArgs Member Properties

The following properties are members of the SapResetEventArgs Class.

SapResetEventArgs.Context Property

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

Description

Application context associated with server reset events. See the EndResetContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapResetEventArgs.ServerIndex Property

int **ServerIndex** (read-only)

Description

Sapera server index associated with the invocation of the application event handler.

Demo/Example Usage

Not available

SapServerFileNotifyEventArgs

The SapServerFileNotifyEventArgs class contains the arguments to the application handler method for the SapManager.ServerFileNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapServerNotifyEventArgs Class Members

Properties

Context	Application context associated with server events
EventType	Server event that triggered the invocation of the application event handler
FilePercentProgress	Returns the file transfer progress, as a percentage of the file size.

SapServerNotifyEventArgs Member Properties

The following properties are members of the SapServerNotifyEventArgs Class.

SapServerFileNotifyEventArgs.Context Property

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

Description

Application context associated with server events. See the ServerContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapServerFileNotifyEventArgs.EventType Property

SapManager.EventType **EventType** (read-only)

Description

Server event which triggered the invocation of the application event handler. See the SapManager.EventType property for the list of possible values.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.FilePercentProgress Property

int **FilePercentProgress** (read-only)

Description

Demo/Example Usage

Not available

SapServerNotifyEventArgs

The SapServerNotifyEventArgs class contains the arguments to the application handler method for the SapManager.ServerNotify event.

Namespace: DALSA.SaperaLT.SapClassBasic

SapServerNotifyEventArgs Class Members

Properties

Context	Application context associated with server events
EventType	Server event that triggered the invocation of the application event handler
ResourceIndex	Sapera resource index associated with the invocation of the application event handler
ServerIndex	Sapera server index associated with the invocation of the application event handler

SapServerNotifyEventArgs Member Properties

The following properties are members of the SapServerNotifyEventArgs Class.

SapServerNotifyEventArgs.Context Property

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

Description

Application context associated with server events. See the ServerContext property of the SapManager class for more details.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.EventType Property

SapManager.EventType **EventType** (read-only)

Description

Server event which triggered the invocation of the application event handler. See the SapManager.ServerEventType property for the list of possible values.

Demo/Example Usage

Not available

SapServerNotifyEventArgs.ResourceIndex Property

int **ResourceIndex** (read-only)

Description

Sapera resource index associated with the invocation of the application event handler

Demo/Example Usage

Not available

SapServerNotifyEventArgs.ServerIndex Property

int **ServerIndex** (read-only)

Description

Sapera resource index associated with the invocation of the application event handler

Demo/Example Usage

Not available

Appendix A: Support

Supported Operating Systems

- Windows 7® (32-bit and 64-bit)
- Windows 8® (32-bit and 64-bit)
- Windows 8.1 ® (32-bit and 64-bit)

Supported Teledyne DALSA Devices

Consult the Readme file included with Sapera Camera SDK for a list of currently supported Teledyne DALSA cameras. Refere to our support page www.teledynedalsa.com/imaging/support for the latest information or to contact technical support.

Tested 3rd Party Frame Grabber Compatability

The following table lists 3rd party frame grabbers tested with the Sapera Camera SDK.

3rd Party Frame Grabber	Notes
Silicon Software microEnable	OK
Epix PIXCI 1DB	OK
Matrox Radiant XCL Dual Full	OK
BitFlow CameraLink	Functions correctly if no other 3rd party frame grabbers are present.
National Instruments PCIe 1433	For 32-bit Windows: When installing the Sapera Camera SDK first, then the NI software: • In the registry, identify the directoy used by HKEY_LOCAL_MACHINE\Software\CameraLink\CLSERIALPATH key. • Copy the file clsercor.dll from c:\CameraLink\Serial to this directory. For 64-bit Windows When using the Sapera Camera SDK (order of installation is not important): • In the registry, identify the directory used by HKEY_LOCAL_MACHINE\Software\CameraLink\CLSERIALPATH key. • Create the directory identified by CLSERIALPATH, if it does not already exist. • Copy the file clsernat.dll from c:\Windows\System32 to this directory

Appendix B: Sapera Camera SDK Installations

Setup Types

Teledyne DALSA Installers

	**Note:** You must reboot after the installation of Sapera Camera SDK. However, to streamline the installation process, you may install Sapera Camera SDK (without rebooting), the required device drivers and then reboot.

Teledyne DALSA's installers can be started in two ways:

- **Normal Mode**
  This is the interactive mode provided by default. It is initiated by invoking the **setup.exe** program. The installation proceeds normally as if it was started from Windows Explorer or the Windows command line.
- **Silent Mode**
  This mode requires no user interaction. Any user input is provided through a “response” file. Nothing is displayed by the installer.

	Note: During driver installation, Windows Digital Signature and Logo Testing warnings can be safely ignored.

Silent Mode Installation

Silent Mode installation is recommended when integrating Teledyne DALSA products into your software installation. The silent installation mode allows the Sapera Camera SDK installation to proceed without the need for mouse clicks or other input from a user.

Two steps are required:

- Preparation of a response file to emulate a user.
- Invoking the Sapera Camera SDK installer with command options to use the prepared response file.

Creating a response file

The installer response file is created by performing a Sapera Camera SDK installation with a command line switch "-r". The response file is automatically named `setup.iss` which is saved in the \windows folder. One simple method is to execute the Sapera Camera SDK installer from within a batch file. The batch file will have one command line.

As an example, the command line is:

```
SaperaLTSetup -r
```

Running a Silent Mode Installation

A Sapera Camera SDK silent installation, whether done alone or within a larger software installation requires the Sapera Camera SDK executable and the generated response file `setup.iss`.

Execute the Sapera Camera SDK installer with the following command line:

`SaperaLTSetup -s -f1".\setup.iss"`

where the **`–s`** switch specifies the silent mode and the **`–f1`** switch specifies the location of the response file. In this example, the switch –f1".\setup.iss" specifies that the `setup.iss` file is in the same folder as the Sapera Camera SDK installer.

	Note: For Sapera LT 8.10, the installation process has been modified; a new prompt has been added for installing 'Teledyne Dalsa frame grabbers and CameraLink cameras' only, 'GigE-Vision cameras and the Sapera Network Imaging Package' only, or 'All acquisition components'. Therefore existing response files for previous versions need to be updated and replaced.

Appendix C: Sapera LT and GenICam

What is GenICam?

GenICam™ is an international standard that allows a single application programming interface (API) to control any compliant video source, regardless of its vendor, feature set, or interface technology (GigE Vision®, Camera Link®, etc.).

GenICam consists of five modules:

- GenApi: This module defines the format of an XML file that captures the features of a device. GenApi also specifies how to access and control the features. All GenICam-compliant devices must contain an XML file that conforms to this format.
- Standard Features Naming Convention (SFNC): This module standardizes the names of more than 220 commonly used camera features. To comply with GigE Vision, seven of the features are mandatory. The rest are either recommended or optional. Compliance with the naming convention is important for interoperability, as it frees application software from the complexity of situations where vendors call the same feature by different names, such as, 'Brightness' and 'Gain'.
- GenTL: This module defines a software interface for accessing image data from a generic transport layer.
- GenCP: This module provides a standardized packet based protocol to read and write device registers, and gives access to a GenApi compliant XML file present on the device

There are two levels of compliance to GenICam:

- GenICam-compliance: where a product either provides or interprets a compliant XML file.
- GenICam TL-compliance: where a product exposes a transport layer compatible with GenTL.

Currently, Teledyne DALSA offers several cameras with GenICam and GigEVision compliance.

Using Sapera LT with GenICam-compliant Devices

Sapera LT uses the SapAcqDevice and SapFeature classes to access the GenICam features of a device.

A SapAcqDevice object is created for each acquisition device and provides access to the list of features, events and files that are supported on the device. SapAcqDevice also allows the registering and unregistering of callback functions on an event.

Note, references to C++ functions in the following sections also apply to their .Net equivalents.

Features

A SapFeature object can be accessed for each feature on the device and provides more detailed information on the actual feature, such as its access mode, minimum and maximum values, enumerations, and so forth, as well as information used for integrating feature access into graphical user interfaces, such as the feature category.

Feature values can be read and written to using the SapAcqDevice::GetFeatureValue and SapAcqDevice::SetFeatureValue functions. To get more information on a feature, retrieve the SapFeature object for this specific feature using the SapAcqDevice::GetFeatureInfo function. See the Sapera++ – Modifying Camera Features sections for more information and examples on how to access and modify features.

Selectors

A selector is a fundamental concept of GenICamSFNC; it allows using a single feature to control multiple components of the same feature. For example the Gain feature might have three components: Red, Green, and Blue. The SapFeature::IsSelector, SapFeature::GetSelectedFeatureCount, SapFeature::GetSelectedFeatureName, SapFeature::GetSelectingFeatureIndex and corresponding GetSelecting functions allow the user to query information about the selector.

File Transfer

Sapera LT simplifies the transfer of files to and from devices with the SapAcqDevice::GetFileCount and SapAcqDevice::GetFileNameByIndex functions, which allow for the enumeration of the available device files. The SapAcqDevice::WriteFile and SapAcqDevice::ReadFile functions are used to transfer the file in and out of the device.

Notes on the Sapera LT GenICam Implementation

The following functions have GenICam specific notes about their implementation:

- SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode: only the *UpdateFeatureAuto* mode is implemented. Therefore, the SapAcqDevice::UpdateFeaturesFromDevice and SapAcqDevice::UpdateFeaturesToDevice functions are not implemented.
- SapFeature::GetPollingTime: GenICam does not provide polling information to the user, therefore this function always returns 0.
- SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile: The SapFeature class provides functionsto control which features are saved to the device configuration file. In GenICam, this is hardcoded by the device manufacturer in the device description file. Therefore, the SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile has no effect, and returns False when the value is read.
- SapFeature class: the retrieval of feature enumeration properties is currently not implemented; only the name and value can be retrieved.

Events

The SapAcqDevice object always provides two events; "*Feature Info Changed*" and "*Feature Value Changed*". These events are related to feature state changes and not the device. Since GenICam does not give information on what changed in the feature, only "Feature Info Changed" events are generated; the "Feature Value Changed" is never generated.

Type

GenAPI interface mapping to SapFeature types.

GenICam Interface	Sapera Type
IInteger	SapFeature::TypeInt64
IFloat	SapFeature::TypeDouble
IString	SapFeature::TypeString
IEnummeration	SapFeature::TypeEnum
ICommand	SapFeature::TypeBool (write only)
IBoolean	SapFeature::TypeBool
IRegister	SapFeature::TypeArray
ICategory	Not exported; the category is a property of the feature.
ISelector	The selector is a property of the feature regardless of its type.
IPort	This is the interface to the underlying transport technologies; it is not exported to the user.

You can retrieve the type of a feature using the SapFeature::GetType function (C++) or SapFeature.DataType Property (.NET). If the type returned is TypeArray, reading /writing to this feature must use a SapBuffer.

Currently the ICommand is mapped to a SapFeature::TypeBool. Setting any value will execute that action and return when the action is complete. One limitation of this mapping is that if the action takes more than the Sapera timeout, setting the value might return false even if the action succeeded.

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**.