DALSA • 7075 Place Robert-Joncas, Suite 142 • St-Laurent, Quebec, H4M 2Z2 • Canada
http://www.imaging.com

Sapera LT™

Basic Modules
Reference Manual

Edition 6.00

Part number OC-SAPM-BMR00

NOTICE

Printed on August 1, 2006

Document Number: OC-SAPM-BMR00
Printed in Canada

Contents

Introduction

Overview of the Manual

The *Sapera LT Basic Modules Reference Manual* is the main reference for the Sapera C API. The manual is divided in two: the first part presents an overview of how to use the API, the second part describes the API's functions and parameters.

The *Sapera Acquisition Parameters Reference Manual* complements the Sapera LT Basic Modules Reference manual with a description of all the acquisition parameters, capabilities and definitions.

Note that the *Sapera LT Pixel Processor Module Programmer's Manual* and the *Sapera LT CAB Programmer's Manual* ship only with the appropriate hardware.

This manual covers the following topics:

- **Sapera C API Overview**
  Information and examples on using the Sapera C API
- **Sapera Acquisition Modules API**
  Description of the Sapera Acquisition modules and their functions.
  Refer to the *Sapera LT Acquisition Parameters Reference Manual* for a description of the acquisition parameters and capabilities.
- **Sapera Basic Modules API**
  Description of the Sapera Basic modules and their functions.
- **Error Messages**
  Description of the error message returned by Sapera function calls.
- **Macro Definitions**
  Description of all Sapera supplied macros.
- **Data Definitions**
  Description of all data types used in Sapera.
- **Appendix A: Server Management**
  Descriptions of acquisition controls including camera parameters and capabilities.
- **Appendix B: File Formats**
  Description of all buffer, graphic font, and LUT file formats supported by Sapera LT.
- **DALSA Contact Information**
  Phone numbers, web site, and important email addresses.

About the Manual

This manual exists in printed, compiled HTML help, and Adobe Acrobat® (PDF) formats. The help and PDF formats make full use of hypertext cross-references and include links to the DALSA home page on the Internet located at **http://www.imaging.com**, accessed using any web browser.

Using the Manual

File names, directories, and Internet sites will be in bold text (e.g., **setup.exe**, **c:\windows, http://www.imaging.com**). Function parameters will be in italics (e.g., *hServer*).

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

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

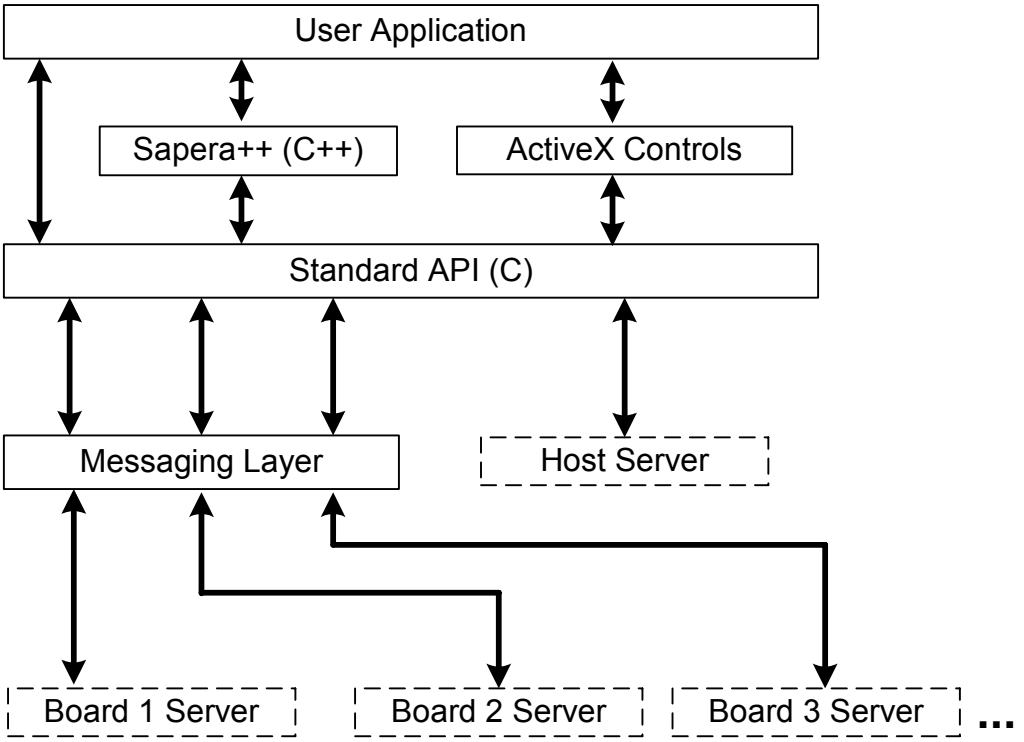
Sapera C API Overview

Sapera Architecture

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

Application Architecture

Whichever API is used (C, C++, or ActiveX), the Sapera modular architecture allows applications to be distributed on different Sapera Servers. Each server can run either on the host computer or on a DALSA board. Sapera calls are routed to different servers via the Sapera messaging layer in a fashion completely independent of the underlying hardware.



Definition of Terms

What is a server?

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

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

What is a static resource?

Resources attached to a physical device are called static resources. For example, a frame grabber can have an acquisition resource, display resource, and a processor resource. These resources can be manipulated to control a physical device through a Sapera Server.

What is a dynamic resource?

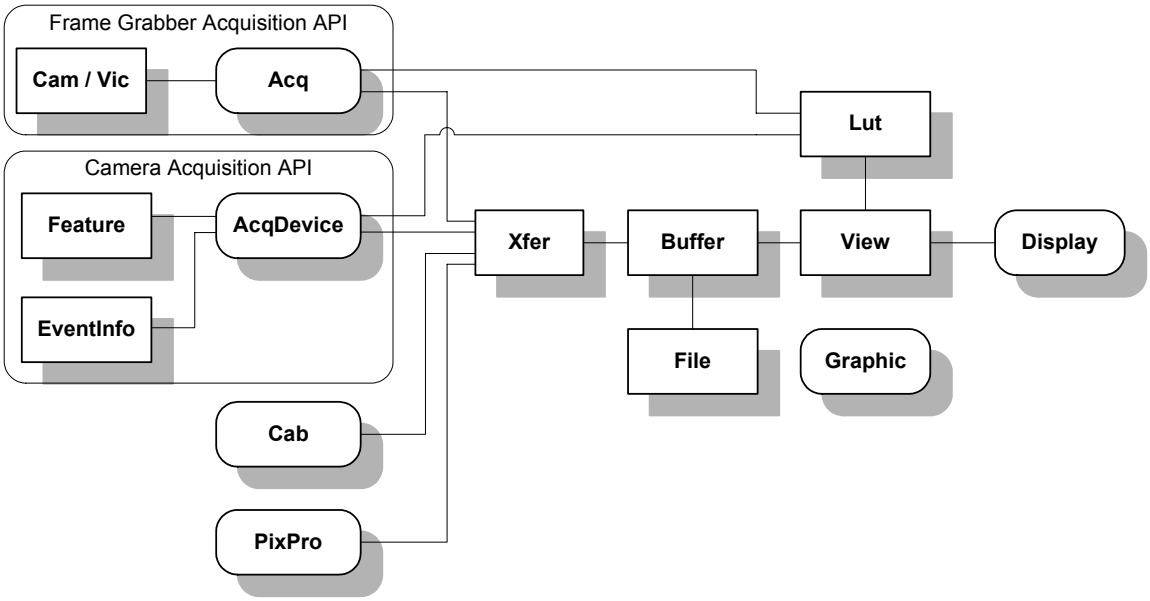
A dynamic resource is an abstract representation of data storage (such as a buffer, lookup table, object, etc.) or links that connect the data storage to static resources. Unlike static resources, dynamic resources are not dependent on physical devices; therefore, users on a specified server can freely 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. The complete Sapera API is composed of a series of modules organized in a particular architecture. These modules are described in the Description of Sapera Modules section.

Library Architecture

The block diagram below illustrates the Sapera Library architecture illustrating all the module interconnections. In this diagram, standard rectangles represent *dynamic resource modules* while rounded rectangles represent *static resource modules*.



Description of Sapera Modules

Below is a brief description of the modules (blocks) and their connections to other modules.

Acquisition Module (Acq)

The Acquisition module refers to a static resource. It is used to control an acquisition device that is present on any DALSA board that supports an acquisition section. The Acquisition module includes the functionality to read and write the acquisition parameters individually. Optionally, it can work in conjunction with both the Cam and Vic modules for grouping and storing parameters. The Transfer module, however, is required for synchronizing (starting or stopping) the acquisition process. This module is used by the Transfer module and uses the Cam and Vic modules.

Cam and Vic Modules

These two modules refer to dynamic resources. They are used for grouping and storing acquisition parameters related to the camera (cam) and video-input-conditioning (vic), respectively. Examples of *Cam* parameters include video format, number of input bits and the pixel clock, while examples of *Vic* parameters include brightness, contrast, and saturation.

Note: The *Cam* and *Vic* modules are strictly dedicated to the usage of cameras hooked to a frame grabber. To manipulate a GigE or DCAM camera refer to the *AcqDevice* module.

AcqDevice Module

The AcqDevice module refers to a static resource. It is used to control a DALSA camera device (e.g., a GigE camera). The AcqDevice module includes the functionality to read and write the feature values individually or all at once. The module also allows to save/restore the features to/from a file. The Transfer module, however, is required for synchronizing (starting or stopping) the acquisition process. This module is used by the Transfer module and uses the Feature module.

Feature Module

The Feature module refers to a dynamic resource. It is used to retrieve the feature information from the AcqDevice module. Each feature supported by the AcqDevice module provides a set of capabilities such as name, type, access mode, etc. that can be obtained through the feature module.

EventInfo Module

The EventInfo module refers to a dynamic resource. It is used to retrieve information from the events sent by the AcqDevice module. Each event supported by the AcqDevice module provides a set of parameters that can be accessed individually through the EventInfo module.

Transfer Module (Xfer)

The Transfer module refers to a dynamic resource. It is used to establish a connection and perform a data transfer between a source and a destination resource. For example, it is used to control an acquisition process by using the Acquisition resource as the source and a Buffer resource as the destination. The Transfer module uses the Acq, AcqDevice, Buffer, Cab, and Pixel Processor modules.

Buffer Module

The Buffer module refers to a dynamic resource. The module includes the functionality to manipulate a generic buffer that can be either one-dimensional (a vector) or two-dimensional (an image). Buffers are the base data storage resources of Sapera. The Buffer module is used by the Transfer, View and Graphic modules.

LookUp Table Module (Lut)

The LookUp Table module refers to a dynamic resource. The module includes the functionality to manipulate a generic LookUp table. The LookUp Table may be used as an input (in an acquisition process), an output (in a display process), or processing LookUp Table. The Lookup Table module is used by the Acq, AcqDevice, and View modules.

View Module

The View module refers to a dynamic resource. It is used to establish a connection and perform a data transfer between the Buffer and Display resources. For example, it is used to display a buffer by transferring its data from system memory to video memory. The View module uses the Buffer and Display modules.

File Module

The File module refers to a dynamic resource. It is used to exchange images between buffers and files. Various file formats are supported (e.g., TIFF, BMP, RAW, JPEG, AVI, and the DALSA Custom Format - CRC).

Display Module

The Display module refers to a static resource. It is used to control a display device that is present on the system display (your computer video card) or on any DALSA board supporting a display section. The Display module includes the functionality to read and write display parameters. In cases where the buffer is located outside video memory (system memory or off-screen memory), the View module transfers data to video memory. The Display module is used by the View module.

Graphic Module

The Graphic module refers to a static resource. It is used to control a graphic or processing device that is present in the system (your system CPU) or on any DALSA board supporting onboard graphics or other processing devices. The Graphic module includes the functionality to manipulate drawing shapes, vectors, and text within a buffer. The Graphic module uses the Buffer module.

DALSA Coreco Auxiliary Bus Module (Cab)

The DALSA Coreco Auxiliary Bus (CAB) module refers to a static resource. It is used to control the DALSA Coreco Auxiliary Bus device. The CAB is typically used for transferring data between two DALSA boards. This module is used by the Transfer module. For more information, consult the *Sapera LT CAB Programmer's Manual*.

Pixel Processor Module (PixPro)

The Pixel Processor module refers to a static resource. It controls the Pixel Processor daughter card that plugs into certain DALSA boards. The Pixel Processor is often used for applying simple pre-processing to an image. The Pixel Processor module is used by the Transfer module. For more information, consult the *Sapera LT Pixel Processor Module Programmer's Manual*.

The Three Sapera APIs

Three different APIs are available under Sapera:

- Sapera Standard API (based on C language)
- Sapera++ Classes (based on C++ language)
- Sapera LT ActiveX Controls (language independent)

The following sections will illustrate the standard **C API**. For C++ and ActiveX API information consult the *Sapera++ Programmer's Manual* and the *Sapera LT ActiveX Controls Programmer's Manual* respectively.

Header Files and Libraries (C API)

The following files are provided for building C applications with Sapera LT.

File Name	Description	Location
corapi.h	Main header file	**Sapera\Include**
corapi.lib	C library for Visual C++ .6.0 and up	**Sapera\Lib**
corapi.lib	C library for C++ Builder 6	**Sapera\Lib\Bc**

Creating a Sapera C Application

The following sections describe how to create a Sapera C application in Visual Studio 6.0, Visual Studio .NET 2003, Visual Studio 2005, and C++ Builder 6.

Visual Studio 6.0

To compile and link an application that uses the Sapera C library:

1. Include **corapi.h** in the program source code (it includes all other required headers).
2. Add **$(SAPERADIR)\Include** in *Project* | *Settings…* | *C/C++* | *Preprocessor* | *Additional include directories*.
3. Insert **$(SAPERADIR)\Lib\corapi.lib** in *Project* | *Add to Project* | *Files*.
4. In *Project* | *Settings…* | *C/C++* | *Code Generation* | *Use run-time library*, choose the option *Multithreaded DLL* (in release mode) or *Debug Multithreaded DLL* (in debug mode).

Note: Step 4 is required because the Sapera library is compiled with the *Multithreaded DLL* version of the **Microsoft runtime library**. If your application contains other DLL components, compile these DLLs with the same option.

Visual Studio .NET 2003 and Visual Studio 2005

To compile and link an application that uses the Sapera C library:

1. Include **corapi.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Include** in *Project* | *Properties* | *C/C++* | *General* | *Additional Include Directories*.
3. Insert **$(SAPERADIR)\Lib\corapi.lib** in *Project* | *Add Existing Item …*
4. 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).

C++ Builder 6

Follow the steps below to compile and link an application that uses the Sapera C library:

1. Include **corapi.h** in the program source code (it includes all other required headers).
2. Add **$(SAPERADIR)\Include** in *Project* | *Options…* | *Directories/Conditionals* | *Include path*.
3. Insert **$(SAPERADIR)\Lib\Bc\corapi.lib** in *Project* | *Add to Project …*

API Naming Conventions

The following describes naming conventions for API functions.

Functions

The API functions follow standard naming conventions. First, all API functions are prefixed with "Cor", derived from "Coreco". This prefix is followed by the module name, as described before in the architecture, and next by the function name. All functions also return an error code. Below is the syntax description with some examples.

```
CORSTATUS Cor<module name><function name>(...)
```

Examples:

```
CORSTATUS status;              // Status code
status = CorBufferClear(...)   // Clear function of Buffer module
status = CorXferStart(...)     // Start function of Transfer module
status = CorLutNormal(...)     // Generate a normal lookUp table
```

Handles

All API functions refer to a server and/or a module handle (see Working with Handles). The server handle is always named CORSERVER. The module handles use the following syntax:

COR<*module name*>

Examples:

```
CORSERVER hServer;   // Handle to a server
CORBUFFER hBuffer;   // Handle to a buffer
CORACQ hAcq;         // Handle to an acquisition
CORDISPLAY hDisplay; // Handle to a display
```

Capabilities and Parameters

Each resource may have a series of capabilities and parameters (see Capabilities and Parameters) that follow the syntax below:

For a capability number:

COR<*module name*>_CAP_<*capability name*>

For a parameter number:

COR<*module name*>_PRM_<*capability name*>

And for each of the possible values:

COR<*module name*>_VAL_<*capability name*>_<*value description*>

Examples:

```
CORACQ_CAP_CHANNEL         // Capability Channel of Acquisition module
CORACQ_PRM_CHANNEL         // Parameter Channel of Acquisition module

CORACQ_VAL_CHANNEL_SINGLE  // Single channel value for Acquisition module
CORACQ_VAL_CHANNEL_DUAL    // Dual channel value for Acquisition module
```

Enumerated Arguments

A function may have one or more enumerated arguments. The list of possible values for such arguments are as follows:

COR<*module name*>_<*value description*>

Example:

```
CORBUFFER_FILEFORMAT_BMP   // Bitmap file format in the buffer module
CORBUFFER_FILEFORMAT_TIFF  // TIFF file format in the buffer module
```

Working with Handles

Accessing resource data in Sapera can only be accomplished by calling an API function. Therefore, servers and resources are all assigned a handle. A handle is a structure containing the necessary information to access internal resource data. To get a handle to a resource involves two steps:

- Getting a server handle.
- Getting the resource handle.

Getting a Server Handle

There are three ways to get a server handle in Sapera:

- The default server, on which the current application is running, is obtained by calling the following function:

```
CORSERVER hServer;    // Declare a server handle

// Initialize Sapera API
CorManOpen();

// Get the default server handle
hServer = CorManGetLocalServer();
```

- You may also enumerate all of the currently available Sapera Servers, using for each an index that ranges from 0 to *nServer*-1, where *nServer* is the number of servers found by the API.

```
CORSTATUS status;     // Declare status code
UINT32 nCount;        // Declare a server count
UINT32 nIndex;        // Declare a server index
char szName[64];      // Declare a character string for returned name
CORSERVER hServer;    // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// Get the server count
status = CorManGetServerCount(&nCount);

// Get the server handle from an index
status = CorManGetServerByIndex(nIndex, szName, &hServer);
```

- You may also specify the exact server name.

```
CORSTATUS status;                  // Declare status code
CORSERVER hServer;                 // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// Get the server handle by specifying a name
status = CorManGetServerByName("X64-CL_1", &hServer);
```

Use the following function to release the server handle when you have finished using it:

```
CORSTATUS status;                  // Declare status code
CORSERVER hServer;                 // Declare a server handle

// Release the specified server handle
status = CorManReleaseServer(hServer);
```

A more comprehensive discussion on this topic is found in Appendix A: Server Management.

Getting the Resource Handle

The following describes getting static and creating dynamic handle resources.

Getting a Handle to a Static Resource

As noted in the Architecture Section, static resources are related to devices on a server. Therefore, their number depends on the specific server where they are located. Each static resource module includes a function to access the resource count, as in the following example:

```
CORSTATUS status;          // Declare status code
CORSERVER hServer;         // Declare a server handle
UINT32 nAcqCount;          // Declare a acquisition count
UINT32 nDisplayCount;      // Declare a display count

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

status = CorAcqGetCount(hServer, &nAcqCount);           // Get acquisition count
status = CorDisplayGetCount(hServer, &nDisplayCount);   // Get display count
```

You then obtain the resource handle by specifying an index ranging from 0 to *nxxxCount*-1. When the handle is no longer used, it must be released.

```
CORSTATUS status;     // Declare status code
CORSERVER hServer;    // Declare a server handle
CORACQ hAcq;          // Declare an acquisition handle
CORDISPLAY hDisplay;  // Declare an display handle

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Get resource handles
status = CorAcqGetHandle(hServer, 0, &hAcq);
status = CorDisplayGetHandle(hServer, 0, &hDisplay);

// Use them
...

// Release handles when finished
status = CorAcqRelease(hAcq);
status = CorDisplayRelease(hDisplay);

// Close Sapera API
status = CorManClose();
```

Creating a Handle for a Dynamic Resource

Because dynamic resources are not device-based their potential number is unlimited. Each dynamic resource has its own creation arguments. Below is an example showing the creation of a buffer and a lookup table.

```
CORSTATUS status;     // Declare status coed
CORSERVER hServer;    // Declare a server handle
CORBUFFER hBuffer;    // Declare a buffer handle
CORLUT hLut;          // Decalre a LUT handle

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Create resource handles
status = CorBufferNew(hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8, 0, &hBuffer);
status = CorLutNew(hServer, 256, CORLUT_VAL_FORMAT_UINT8, &hLut);

// Use them
...

// Free handles when finished
status = CorBufferFree(hBuffer);
status = CorLutFree(hLut);

// Close Sapera API
status = CorManClose();
```

Error Management

The following describes interpreting status codes and monitoring sapera errors.

Interpreting Status Codes

All Sapera functions return a status code. If the function executes successfully, it returns the CORSTATUS_OK status code. If an error is detected, the status code describes the nature and the level of the error within the called function. Some status codes also contain additional information related to the specific error. Furthermore, all status codes include a module identifier that indicates which module the function belongs to. The example below demonstrates how to get the different fields of the status code.

```
CORSTATUS status;     // Status code
UINT32 errorId;       // Error identifier
UINT32 errorInfo;     // Additional specific information
UINT32 errorLevel;    // Error level
UINT32 module;        // Module of the function called

// Initialize Sapera API
status = CorManOpen();

// Call an API function
status = CorXXX(...);

// Extract the status code's ID
// If the function succeeds will return CORSTATUS_OK,
// otherwise will return CORSTATUS_xxx.
// See Reference Manual for the complete list of error ID's

errorId = CORSTATUS_ID(status);

// Extract the status code's additional information
// This information is specific to the status code's ID
// Some status code don't support this field
// See Reference Manual for a detailed description of the values.

errorInfo = CORSTATUS_INFO(status);

// Extract the status code's level
// Will return one of the following values:
// CORSTATUS_LEVEL_FAT    Fatal error
// CORSTATUS_LEVEL_ERR    General error
// CORSTATUS_LEVEL_WRN    Warning
// CORSTATUS_LEVEL_INF    Information

errorLevel = CORSTATUS_LEVEL(status);

// Extract the module
// Will return one of the module identifier:
// CORSTATUS_MODULE_ACQ
// CORSTATUS_MODULE_BUFFER
// ...
// See Reference Manual for the complete list of modules

module = CORSTATUS_MODULE(status);

// Close Sapera API
status = CorManClose();
```

You can obtain an associated description string by calling the function **CorManGetStatusText**, which returns a string including a description of the status code.

```
CORSTATUS status;     // Status code
char szText[256];     // Status text

// Call an API function
status = CorXXX(...);

// Get the associated text string
CorManGetStatusText(status, szText, sizeof(szText), NULL, 0);
```

You can also obtain more detailed information by calling the function **CorManGetStatusTextEx**, which returns a string for each field of the status code.

```
CORSTATUS status;
char id[128], info[128], level[64], module[64];

// Call an API function
status = CorXXX(...);

// Get the associated text strings
CorManGetStatusTextEx(status, id, sizeof(id), info, sizeof(info), level,
sizeof(level), module, sizeof(module));
```

Monitoring Sapera Errors

The **logview.exe** utility program included with Sapera provides an easy way to view status code returned by API functions. **logview.exe** is a simple Windows program that includes a list box that stores the status code description strings as soon as they are logged in the API. Options allow you to modify the different fields for display.

It is recommended to start **LogView** before starting 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 **LogView** is not running. Therefore, it is possible to run it only when a problem occurs while running your application.

Capabilities and Parameters

Resources can be characterized by a set of capabilities and parameters. Together they define a resource's ability and current state.

What is a Capability?

A capability, as its name implies, is a value or set of values that describe what a resource can do. Capabilities are used to determine the possible valid values that can be applied to a resource's parameters. They are read-only.

Accessing a Capability

A capability can be obtained from a resource by using the Cor<*module name*>GetCap function. It has the following prototype:

```
CorxxxGetCap(CORxxx handle, UINT32 cap, void *value)
```

- *handle*: valid handle to a resource
- *cap*: valid capability of the resource
- *value*: buffer of proper size to store the capability value(s). The size of a capability can be obtained by using the macro CORCAP_GETSIZE(*cap*)

What is a Parameter?

A parameter describes a characteristic of a resource. It can be read/write or read-only.

Accessing a Parameter

A parameter can be read by using the Cor<*module name*>GetPrm function. It has the following prototype:

```
CorxxxGetPrm(CORxxx handle, UINT32 prm, void *value)
```

- *handle*: valid handle to a resource
- *prm*: valid parameter of the resource
- *value*: buffer of proper size to store the parameter value. The size (in bytes) of a parameter can be obtained by using the macro CORPRM_GETSIZE(*prm*)

You can write parameters with the Cor<*module name*>SetPrm and Cor<*module name*>SetPrmEx functions. They have the following prototypes:

```
CorxxxSetPrm(CORxxx handle, UINT32 prm, UINT32 value)
CorxxxSetPrmEx(CORxxx handle, UINT32 prm, const void *value)
```

- *handle*: valid handle to a resource
- *prm*: valid parameter of the resource
- *value*: buffer of proper size to store the parameter value. The size in bytes of a parameter can be obtained by using the macro CORPRM_GETSIZE(*prm*)

The "Ex" function is used to write to a parameter whose value is greater than four bytes.

Acquiring Images

Required Modules

You need three Sapera modules to initiate the acquisition process:

- **Acq or AcqDevice:** The "Acq" module is needed if you are using a frame grabber while the "AcqDevice" module is needed if you are using a camera directly connected to your PC, such as a GigE or DCAM camera.
- **Buffer**: Dynamic resource used to store the acquired data. The buffer must be allocated using the CORBUFFER_VAL_TYPE_CONTIGUOUS or CORBUFFER_VAL_TYPE_SCATTER_GATHER buffer type to enable the transfer (see Working with Buffers section for more information about contiguous memory and scatter-gather).
- **Transfer:** Dynamic resource used to link the acquisition to the buffer and to synchronize the acquisition operations.

Frame Grabber Acquisition Example

The example below demonstrates how to grab a live image into a buffer allocated in system memory, using the X64-CL board as an acquisition device.

As shown in this example, acquiring an image requires two files to configure the acquisition hardware: a CAM file and a VIC file. The former defines the characteristics of the camera whereas the latter defines how the camera and the acquisition hardware will be used. Refer to the CamExpert online help file for information on CAM and VIC video source parameter files.

Once the acquisition module is initialized using the CAM and VIC files, some parameters are retrieved from it (acquisition width, height, and format) and are used to create a compatible buffer.

Before starting the transfer, you must create a transfer path between the acquisition resource and the buffer resource. Furthermore, when requesting a transfer stop, you must call *CorXferWait* to wait for the transfer process to terminate completely.

```
// Transfer callback function: called each time a complete frame is transferred
//
CORSTATUS CCONV XferCallback (void *context, UINT32 eventType, UINT32 eventCount)
{
    // Display the last transferred frame
    CorViewShow(*(CORVIEW*) context);
    return CORSTATUS_OK;
}

//
// Example program
//
main()
{
    CORSTATUS status;     // Error code
    CORSERVER hSystem;    // System server handle
    CORSERVER hBoard;     // Board server handle
    CORCAM hCam;          // CAM handle
    CORVIC hVic;          // VIC handle
    CORACQ hAcq;          // Acquisition handle
    CORBUFFER hBuffer;    // Buffer handle
    CORXFER hXfer;        // Transfer handle
    CORVIEW hView;        // View handle
    CORDISPLAY hDisplay;  // Display handle
    UINT32 width, height, format;

    // Initialize Sapera API
    status = CorManOpen();

    // Get server handles (system and board)
    hSystem = CorManGetLocalServer();
    status = CorManGetServerByName("X64-CL_1", &hBoard);

    // Get acquisition handle
    status = CorAcqGetHandle(hBoard, 0, &hAcq);  // 0 = First instance

    // Create CAM/VIC handles
    status = CorCamNew(hSystem, &hCam);   // Camera
    status = CorVicNew(hSystem, &hVic);   // Video-Input-Conditionning

    // Load CAM/VIC parameters from file into system memory
    // The acquisition hardware is not initialized at this point
    status = CorCamLoad(hCam, "rs170.cca");
    status = CorVicLoad( hVic, "rs170.cvi");

    // Download the CAM/VIC parameters to the acquisition module
    // The acquisition hardware is now initialized
    status = CorAcqSetPrms(hAcq, hVic, hCam, FALSE);

    // Create a buffer compatible to acquisition
    status = CorAcqGetPrm(hAcq, CORACQ_PRM_SCALE_HORZ, &width);
    status = CorAcqGetPrm(hAcq, CORACQ_PRM_SCALE_VERT, &height);
    status = CorAcqGetPrm(hAcq, CORACQ_PRM_OUTPUT_FORMAT, &format);
    status = CorBufferNew(hSystem, width, height, format,
            CORBUFFER_VAL_TYPE_SCATTER_GATHER, &hBuffer);

    // Create a transfer handle to link acquisition to buffer
    status = CorXferNew(hBoard, hAcq, hBuffer, NULL, &hXfer);

    // Register a callback function on "End-Of-Frame" events
```

```
    status = CorXferRegisterCallback(hXfer, CORXFER_VAL_EVENT_TYPE_END_OF_FRAME,
             XferCallback, (void *)&hView);

    // Activate the connection between acquisition and buffer
    status = CorXferConnect(hXfer);

    // Get display handle
    status = CorDisplayGetHandle(hSystem, 0, &hDisplay);

    // Create a view handle and assign it to a HWND
    status = CorViewNew(hSystem, hDisplay, hBuffer, CORVIEW_VAL_MODE_AUTO_DETECT,
             &hView);
    status = CorViewSetPrm(hView, CORVIEW_PRM_HWND, -1); // -1: create new window

    // Start a continuous transfer (live grab)
    status = CorXferStart(hXfer, CORXFER_CONTINUOUS);

    printf("Press any key to stop grab\n");
    getchar(); // wait until a key has been hit

    // Stop the transfer and wait (timeout = 5 sec)
    status = CorXferStop(hXfer);
    status = CorXferWait(hXfer, 5000);

    // Break the connection between acquisition and buffer
    status = CorXferDisconnect(hXfer);

    printf("Press any key to terminate\n");
    getchar();

    // Release handles when finished (in the reverse order)
    status = CorViewFree(hView);
    status = CorDisplayRelease(hDisplay);
    status = CorXferFree(hXfer);
    status = CorBufferFree(hBuffer);
    status = CorVicFree(hVic);
    status = CorCamFree(hCam);
    status = CorAcqRelease(hAcq);

    // Close Sapera API
    status = CorManClose();

    return 0;
}
```

Modifying the Frame Grabber Parameters

The following describes how to modify frame grabber parameters individually or by group.

Modifying Parameters Individually

Acquisition parameters can be modified individually by using the *CorAcqSetPrm* and/or *CorAcqSetPrmEx* functions. When a new parameter value is requested, that value is verified against the current state of the acquisition module and the acquisition module capabilities. If the modification request is denied because the parameter is dependent on other parameters, then all the parameters in question must be modified by group, in which case you must refer to the next section.

```
CORSTATUS status;      // Error code
CORSERVER hSystem;     // System server handle
CORSERVER hBoard;      // Board server handle
CORACQ hAcq;           // Acquisition handle
UINT32 capSync;        // Sync capability (as an example)

// Initialize Sapera API
status = CorManOpen();

// Get server handles
hSystem = CorManGetLocalServer();
status = CorManGetServerByName("X64-CL_1", &hBoard);

// Get acquisition handle
status = CorAcqGetHandle(hBoard, 0, &hAcq);      // 0 = First instance

// Verify if sync on composite sync is supported
status = CorAcqGetCap(hAcq, CORACQ_CAP_SYNC, &capSync);
if (!status && (capSync & CORACQ_VAL_SYNC_COMP_SYNC))
{
    // Change the sync source to Composite Sync
    status = CorAcqSetPrm(hAcq, CORACQ_PRM_SYNC, CORACQ_VAL_SYNC_COMP_SYNC);
}


// Do something else
...


// Release handles when finished
status = CorAcqRelease(hAcq);

// Close Sapera API
status = CorManClose();
```

Modifying Parameters by Group

Acquisition parameters can be modified by group using the *CorAcqSetPrms* function. When a new set of values is requested, all modified parameters are verified against the given state and capabilities of the acquisition module.

```
CORSTATUS status;      // Error code
CORSERVER hSystem;     // System server handle
CORSERVER hBoard;      // Board server handle
CORACQ hAcq;           // Acquisition handle
CORCAM hCam;           // CAM handle
CORVIC hVic;           // VIC handle

// Initialize Sapera API
status = CorManOpen();

// Get server handles
hSystem = CorManGetLocalServer();
status = CorManGetServerByName("X64-CL_1", &hBoard);

// Get acquisition handle
status = CorAcqGetHandle(hBoard, 0, &hAcq);     // 0 = First instance

// Create a CAM resource (Camera)
status = CorCamNew( hSystem, &hCam);

// Create a VIC resource (Video-Input-Conditioning)
status = CorVicNew( hSystem, &hVic);

// Get current state of the acquisition module and lock parameters
status = CorAcqGetPrms(hAcq, hVic, hCam, TRUE);

// Modify parameters individually
status = CorVicSetPrm(hVic, CORVIC_PRM_CROP_WIDTH, 640);
status = CorVicSetPrm(hVic, CORVIC_PRM_CROP_HEIGHT, 480);
status = CorVicSetPrm(hVic, CORVIC_PRM_SCALE_HORZ, 640);
status = CorVicSetPrm(hVic, CORVIC_PRM_SCALE_VERT, 480);

// Apply the modified parameters on the acquisition module
status = CorAcqSetPrms(hAcq, hVic, hCam, TRUE);


// Do something else
...


// Release handles when finished
status = CorCamFree(hCam);
status = CorVicFree(hVic);
status = CorAcqRelease(hAcq);

// Close Sapera API
status = CorManClose();
```

Using an Input Lookup Table

An Input Lookup Table is first created using the LUT module API and then transferred to the acquisition module (if it has input lookup table capability). The example below illustrates the steps required.

```
CORSTATUS status;     // Error code
CORSERVER hSystem;    // System server handle
CORSERVER hBoard;     // Board server handle
CORACQ hAcq;          // Acquisition handle
CORLUT hLut;          // Lut handle
UINT32 nLut;          // Number of Acquisition LUT
UINT32 pixelDepth;    // Number of bits/pixel to acquire
UINT32 lutFormat;     // Acquisition LUT format
UINT32 entries;       // Total number of entries in the LUT

// Initialize Sapera API
status = CorManOpen();

// Get server handles
hSystem = CorManGetLocalServer();
status = CorManGetServerByName("X64-CL_1", &hBoard);

// Get acquisition handle
status = CorAcqGetHandle(hBoard, 0, &hAcq);    // 0 = First instance

// Check if the acquisition device has at least one lookup table available
status = CorAcqGetPrm(hAcq, CORACQ_PRM_LUT_MAX, &nLut);

if( nLut > 0)
{
   // Create a LUT resource
   // Get the pixel depth and current LUT format from the acquisition module
   status = CorAcqGetPrm(hAcq, CORACQ_PRM_PIXEL_DEPTH, &pixelDepth);
   status = CorAcqGetPrm(hAcq, CORACQ_PRM_LUT_FORMAT, &lutFormat);

   // Calculate the number of entries needed for the LUT
   entries = 1 << pixelDepth;

   // Create LUT resource
   status = CorLutNew(hSystem, entries, lutFormat, &hLut);

   // Initialize a reverse LUT
   status = CorLutReverse(hLut);

   // Load LUT to acquisition module LUT #0
   status = CorAcqSetLut(hAcq, hLut, 0);

   // Select LUT #0 as the active LUT
   status = CorAcqSetPrm(hAcq, CORACQ_PRM_LUT_NUMBER, 0);

   // Enable LUTs
   status = CorAcqSetPrm(hAcq, CORACQ_PRM_LUT_ENABLE, TRUE);

   // Release handles when finished
   status = CorLutFree(hLut);
}
status = CorAcqRelease(hAcq);

// Close Sapera API
status = CorManClose();
```

Camera Acquisition Example

The example below demonstrates how to grab a live image into a buffer allocated within system memory using the *Genie M640* camera as an acquisition device.

Acquiring an image can be performed either by using the camera default settings (feature values stored in the camera) or by loading a configuration file. The configuration file can be generated using *CamExpert*.

Once the AcqDevice module is initialized (with or without using a configuration file), some parameters are retrieved from it (acquisition width, height, and format) and are used to create a compatible buffer.

Before starting the transfer, you must create a transfer path between the AcqDevice resource and the buffer resource. Furthermore, when requesting a transfer stop, you must call *CorXferWait* to wait for the transfer process to terminate completely.

```
// Transfer callback function: called each time a complete frame is transferred
//
CORSTATUS CCONV XferCallback (void *context, UINT32 eventType, UINT32 eventCount)
{
    // Displays the last transferred frame
    CorViewShow(*(CORVIEW*) context);
    return CORSTATUS_OK;
}

//
// Example program
//
main()
{
    CORSTATUS status;        // Error code
    CORSERVER hSystemServer; // System server handle
    CORSERVER hAcqServer;    // Camera server handle
    CORACQDEVICE hAcqDevice; // Camera handle
    CORBUFFER hBuffer;       // Buffer handle
    CORXFER hXfer;           // Transfer handle
    CORVIEW hView;           // View handle
    CORDISPLAY hDisplay;     // Display handle
    UINT32 width, height, format;

    // Initialize Sapera API
    status = CorManOpen();

    // Gets server handles (system and camera)
    hSystemServer = CorManGetLocalServer();
    status = CorManGetServerByName("Genie_M640_1", &hAcqServer);

    // Gets camera handle
    status = CorAcqDeviceGetHandle(hAcqServer, 0, &hAcqDevice);  // 0 = First instance

    // Optionally loads a configuration file to program the camera
    // Otherwise, the camera will work with its default settings
    status = CorAcqDeviceLoadFeatures(hAcqDevice, "Genie_M640_Example.ccf");

    // Creates a buffer compatible with camera
    status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "Width", &width,
             sizeof(width));
    status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "Height", &height,
             sizeof(height));
    status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "SaperaBufferFormat",
             &format, sizeof(format));
    status = CorBufferNew(hSystemServer, width, height, format,
             CORBUFFER_VAL_TYPE_SCATTER_GATHER, &hBuffer);

    // Creates a transfer handle to link camera to buffer
    status = CorXferNew(hAcqServer, hAcqDevice, hBuffer, NULL, &hXfer);

    // Registers a callback function on "End-Of-Frame" events
    status = CorXferRegisterCallback(hXfer, CORXFER_VAL_EVENT_TYPE_END_OF_FRAME,
             XferCallback, (void *)&hView);

    // Activates the connection between camera and buffer
    status = CorXferConnect(hXfer);

    // Gets display handle
    status = CorDisplayGetHandle(hSystemServer, 0, &hDisplay);
```

```
    // Creates a view handle and assign it to a HWND
    status = CorViewNew(hSystemServer, hDisplay, hBuffer,
             CORVIEW_VAL_MODE_AUTO_DETECT,&hView);
    status = CorViewSetPrm(hView, CORVIEW_PRM_HWND, -1); // -1: create new window

    // Starts a continuous transfer (live grab)
    status = CorXferStart(hXfer, CORXFER_CONTINUOUS);

    printf("Press any key to stop grab\n");
    getch(); // wait until a key has been hit

    // Stops the transfer and waits (timeout = 5 sec)
    status = CorXferStop(hXfer);
    status = CorXferWait(hXfer, 5000);

    // Breaks the connection between acquisition and buffer
    status = CorXferDisconnect(hXfer);

    printf("Press any key to terminate\n");
    getch();

    // Releases handles when finished (in the reverse order)
    CorViewFree(hView);
    CorDisplayRelease(hDisplay);
    CorXferFree(hXfer);
    CorBufferFree(hBuffer);
    CorAcqDeviceRelease(hAcqDevice);

    // Close Sapera API
    status = CorManClose();

    return 0;
}
```

Modifying the Camera Features

The following describes how to modify camera features individually or by group.

Accessing Feature Information and Values

The following example shows how features of the camera can be accessed. Information such as type, range and access mode can be retrieved for each supported feature. The AcqDevice module 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 therefore be written to the camera all at once. The next section shows how to proceed in such a case.

```
//
// Callback Function
//
CORSTATUS CCONV CameraCallback(void *context, COREVENTINFO hEventInfo)
{
   CORSTATUS status;
   UINT32 eventCount;
   UINT32 eventIndex;
   char eventName[64];
   CORACQDEVICE hAcqDevice = (CORACQDEVICE) context;

   // Retrieve count, index and name of the received event
   status = CorEventInfoGetPrm(hEventInfo, COREVENTINFO_PRM_EVENT_COUNT, &eventCount);
   status = CorEventInfoGetPrm(hEventInfo, COREVENTINFO_PRM_EVENT_INDEX, &eventIndex);
   status = CorAcqDeviceGetEventNameByIndex(hAcqDevice, 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 = CorEventInfoGetPrm(hEventInfo, COREVENTINFO_PRM_FEATURE_INDEX,
               &featureIndex);
      status = CorAcqDeviceGetFeatureNameByIndex(hAcqDevice, featureIndex,
               featureName, sizeof(featureName));
   }

   return CORSTATUS_OK;
}

//
// Main Program
//
main()
{
   CORSTATUS status;
   CORSERVER hSystemServer, hAcqServer;
   CORACQDEVICE hAcqDevice;
   UINT32 featureCount, featureIndex;
   CORFEATURE hFeature;
   UINT32 value, min, max, inc;
   UINT32 enumCount, enumIndex, enumValue;
   char enumString[64];
   UINT32 lutIndex, lutNEntries, lutFormat;
   CORLUT hLut;
   UINT32 numEvents, eventIndex;
   char eventName[64];

   // Initialize Sapera API
   status = CorManOpen();

   // Get handle to the system server
   hSystemServer = CorManGetLocalServer();

   // Get handle to the camera server
   status = CorManGetServerByName("Genie_M640_1", &hAcqServer);

   // Get handle to the camera resource
   status = CorAcqDeviceGetHandle(hAcqServer, 0, &hAcqDevice);
```

```
// Get the number of features provided by the camera
status = CorAcqDeviceGetFeatureCount(hAcqDevice, &featureCount);

// Create an empty feature object (to receive information)
status = CorFeatureNew(hAcqServer, &hFeature);

//
// Example 1 : Browse through the feature list
//
for (featureIndex = 0; featureIndex < featureCount; featureIndex++)
{
   char featureName[CORPRM_GETSIZE(CORFEATURE_PRM_NAME)];
   UINT32 featureType;

   // Get information from current feature
   // Get feature object
   status = CorAcqDeviceGetFeatureInfoByIndex(hAcqDevice, featureIndex, hFeature);

   // Extract name and type from object
   status = CorFeatureGetPrm(hFeature, CORFEATURE_PRM_NAME, featureName);
   status = CorFeatureGetPrm(hFeature, CORFEATURE_PRM_TYPE, &featureType);

   // Get/set value from/to current feature
   switch (featureType)
   {
      // Feature is a 64-bit integer
   case CORFEATURE_VAL_TYPE_INT64:
      {
         UINT64 value;
         status = CorAcqDeviceGetFeatureValueByIndex(hAcqDevice, featureIndex,
                  &value, sizeof(value));
         value += 10;
         status = CorAcqDeviceSetFeatureValueByIndex(hAcqDevice, featureIndex,
                  &value, sizeof(value));
      }
      break;
      // Feature is a boolean
   case CORFEATURE_VAL_TYPE_BOOL:
      {
         BOOL value;
         status = CorAcqDeviceGetFeatureValueByIndex(hAcqDevice, featureIndex,
                  &value, sizeof(value));
         value = !value;
         status = CorAcqDeviceSetFeatureValueByIndex(hAcqDevice, featureIndex,
                  &value, sizeof(value));
      }
      break;
      // Other feature types
      // ...
   }
}

//
// Example 2 : Access specific feature (integer example)
//
// Get feature object
status = CorAcqDeviceGetFeatureInfoByName(hAcqDevice, "Gain", hFeature);

// Extract minimum, maximum and increment values
status = CorFeatureGetMin(hFeature, &min, sizeof(min));
status = CorFeatureGetMax(hFeature, &max, sizeof(max));
```

```
status = CorFeatureGetInc(hFeature, &inc, sizeof(inc));

// Read, modify and write value
status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "Gain", &value,
         sizeof(value));
value += 10;
status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "Gain", &value,
         sizeof(value));

//
// Example 3 : Access specific feature (enumeration example)
//
// Get feature object
status = CorAcqDeviceGetFeatureInfoByName(hAcqDevice, "ExposureMode", hFeature);

// Get number of items in enumeration
status = CorFeatureGetEnumCount(hFeature, &enumCount);

for (enumIndex = 0; enumIndex < enumCount; enumIndex++)
{
   // Get item string and value
   status = CorFeatureGetEnumString(hFeature, enumIndex, enumString,
            sizeof(enumString));
   status = CorFeatureGetEnumValue(hFeature, enumIndex, &enumValue);
}

// Read a value and get its associated string
status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "ExposureMode", &enumValue,
         sizeof(enumValue));
status = CorFeatureGetEnumStringFromValue(hFeature, enumValue, enumString,
         sizeof(enumString));

// Write a value corresponding to known string
status = CorFeatureGetEnumValueFromString(hFeature, "Manual", &enumValue);
status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "ExposureMode", &enumValue,
         sizeof(enumValue));

//
// Example 4 : Access specific feature (LUT example)
//
// Select a LUT and retrieve its size and format
status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "LUTNumberEntries",
         &lutNEntries, sizeof(lutNEntries));
status = CorAcqDeviceGetFeatureValueByName(hAcqDevice, "LUTFormat", &lutFormat,
         sizeof(lutFormat));

// Create and generate a compatible software LUT
status = CorLutNew(hSystemServer, lutNEntries, lutFormat, &hLut);
status = CorLutReverse(hLut);

// Write LUT values to camera
status = CorAcqDeviceSetFeatureDataByName(hAcqDevice, "LUTData", hLut);

//
// Example 5 : Callback management
//
// Browse event list
status = CorAcqDeviceGetEventCount(hAcqDevice, &numEvents);
for (eventIndex = 0; eventIndex < numEvents; eventIndex++)
{
   status = CorAcqDeviceGetEventNameByIndex(hAcqDevice, eventIndex, eventName,
            sizeof(eventName));
```

```
   }

   // Register event by name
   status = CorAcqDeviceRegisterCallbackByName(hAcqDevice, "Feature Value Changed",
            CameraCallback, hAcqDevice);

   // Modified a feature (Will trigger callback function)
   value = 150;
   status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "Gain", &value,
            sizeof(value));

   // Unregister event by name
   status = CorAcqDeviceUnregisterCallbackByName(hAcqDevice, "Feature Value Changed");

   // Release handles
   status = CorLutFree(hLut);
   status = CorFeatureFree(hFeature);
   status = CorAcqDeviceRelease(hAcqDevice);
   status = CorManReleaseServer(hAcqServer);
   status = CorManReleaseServer(hSystemServer);

   // Close Sapera API
   status = CorManClose();
}
```

Writing Feature Values by Group

When a series of features are tightly coupled it becomes almost impossible to modify those features without following a specific order. One example is the region-of-interest (ROI) where the four values (top, left, width and height) depend on each other. To circumvent this problem the AcqDevice module allows you to temporarily set the feature values in an “internal cache” and then downloads the values to the camera all at once. The following code illustrates the ROI example.

```
. . .

   CORSTATUS status;
   UINT32 value;

   // Set manual mode to update features
   status = CorAcqDeviceSetPrm(hAcqDevice, CORACQDEVICE_PRM_UPDATE_FEATURE_MODE,
      CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_MANUAL);

   // Set buffer top position value (in the internal cache only)
   value = 50;
   status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "OffsetY", &value,
      sizeof(value));

   // Set buffer left position value (in the internal cache only)
   value = 50;
   status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "OffsetX", &value,
      sizeof(value));

   // Set buffer width value (in the internal cache only)
   value = 300;
   status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "Width", &value,
      sizeof(value));

   // Set buffer height value (in the internal cache only)
   value = 300;
   status = CorAcqDeviceSetFeatureValueByName(hAcqDevice, "Height", &value,
      sizeof(value));

   // Write features value to the device (by reading values from the internal cache)
   status = CorAcqDeviceUpdateFeaturesToDevice(hAcqDevice);

   // Set back the automatic mode
   status = CorAcqDeviceSetPrm(hAcqDevice, CORACQDEVICE_PRM_UPDATE_FEATURE_MODE,
      CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_AUTO);

...
```

Displaying Images

Required Modules

The following three Sapera modules are required to initiate a display process:

- **Display**: Static resource based on an onboard display section.
- **Buffer**: Dynamic resource containing data to display. Several type options may be chosen when allocating the buffer to be compatible with the different display modes (see Working with Buffers on page 39 for more information about these options).
- **View**: Dynamic resource used to link the display to the buffer and to synchronize the display operations.

Display Example

The example below illustrates how to display an image contained within a system buffer to the computer VGA card. The buffer is transferred to the Windows desktop using the DIB mode (automatically detected by the View module). When using this mode, a Windows Device-Independent Bitmap (DIB) is first created before being sent to VGA memory. For more information on the View modes, see Modifying the View Parameters.

```
CORSTATUS status;     // Error code
CORSERVER hSystem;    // System server handle
CORDISPLAY hDisplay;      // Display handle
CORBUFFER hBuffer;    // Buffer handle
CORVIEW hView;        // View handle

// Initialize Sapera API
status = CorManOpen();

// Get system server handle
hSystem = CorManGetServer();

// Get display handle
status = CorDisplayGetHandle(hSystem, 0, &hDisplay);

// Create a 640x480/8-bit monochrome buffer in system memory
status = CorBufferNew(hSystem, 640, 480, CORBUFFER_VAL_FORMAT_UINT8,
CORBUFFER_VAL_TYPE_VIRTUAL, &hBuffer);

// Create a view handle
status = CorViewNew(hSystem, hDisplay, hBuffer, CORVIEW_VAL_MODE_AUTO_DETECT, &hView);

// Set HWND parameter to NULL to display image on the desktop
status = CorViewSetPrm(hView, CORVIEW_PRM_HWND, NULL);

// Display image in the desktop
status = CorViewShow(hView);

// Release handles when finished
status = CorViewFree(hView);         // Should be freed first
status = CorBufferFree(hBuffer);
status = CorDisplayRelease(hDisplay);

// Close Sapera API
status = CorManClose();
```

Modifying the View Parameters

The following describes view modes, source and destination windows, and zooming.

View Modes

Three viewing modes are available. Specifying CORVIEW_VAL_MODE_AUTO_DETECT when creating the View module will choose the appropriate mode, taking into account the given buffer.

- **DIB mode:** Used to display buffers of any pixel format. A View module can be created in DIB mode if the associated buffer is contiguous, scatter-gather, or virtual. DIB mode uses a device-independent bitmap to represent and transfer buffer data to the Display module.
- **BLT mode:** Used if the display device supports DirectDraw and if the buffer is an offscreen buffer. If the display adapter supports it, BLT mode will perform a fast data transfer from the buffer to the display memory. This mode is usually faster than the DIB mode, if the buffer has been allocated in video memory and if the transfer occurs within the display adapter, thus freeing the CPU or PCI bus of potential bottlenecks. Create offscreen buffers in video

memory using the same pixel format as the display adapter's current pixel format (for instance, RGB565 for a 65536 color configuration). For offscreen buffers in system memory, the CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN parameter supplies a list of pixel formats that DirectDraw can copy directly to the display memory. If the buffer's pixel format is not in this list, a software conversion will be performed.

- **Keyer mode:** Used if the display device supports DirectDraw and if the buffer is an overlay buffer. The display adapter's hardware can perform a color keying operation between the overlay buffer and the display memory using the keyer color defined by the CORVIEW_PRM_KEYER_COLOR parameters. The color keying mode is determined by the View module's CORVIEW_PRM_OVERLAY_MODE parameter.

Note: The DirectDraw Capabilities Detection tool may help you determine what color formats are supported by your VGA card as described in the *Sapera LT User's Manual*.

Source and Destination Windows and Zooming

The following are the View module's two reference windows:

- A source window that defines an area in the buffer to display.
- A destination window that defines a region on the display surface or in the target window's client area (if CORVIEW_PRM_HWND is not 0) where the source window region is displayed.

Upon the creation of a new View module, the source window is by default the same size as the whole buffer viewed and is positioned at its origin. The destination window matches the dimensions of the source window and is positioned at the origin of the display surface or the target window's client area. The dimensions and position of these windows can be modified using the CORVIEW_PRM_ROI_SRC_xxx and CORVIEW_PRM_ROI_DST_xxx parameters, if the CORVIEW_CAP_ROI_SRC and CORVIEW_CAP_ROI_DST capabilities are not 0 (for instructions on setting these parameters, see Displaying in a Windows Application). If these two windows have the same dimensions, no zooming is performed (the pixels are displayed as they are read in the buffer). If the destination window is a different size from the source window, the buffer elements are zoomed up or down (as appropriate) as they are displayed. The character of the zooming operation depends on the value of the CORVIEW_CAP_ZOOM_HORZ_METHOD and CORVIEW_CAP_ZOOM_VERT_METHOD capabilities. Zooming can be accomplished through pixel dropping or replication, interpolation, or by powers of 2. X and Y zoom methods are independent from each other.

Zooming may influence the View module's display speed (realtime refresh may not be possible).

Displaying in a Windows Application

The View module contains three callback functions, CorViewOnPaint, CorViewOnMove, and CorViewOnSize. They can be called in your Windows application's respective message handlers for WM_PAINT, WM_MOVE and WM_SIZE. Below is an example of a Windows application using the Visual C++'s MFC library. This is a dialog-based application whose dialog window is used to display the buffer content. The window's handle is passed as a parameter in the OnInitDialog handler to ensure that it is not null. The destination window is adjusted each time the dialog is resized. The source window corresponds to the buffer rectangle by default. The View module will scale the buffer contents into the dialog window because it is not adjusted.

```
CORSTATUS status;      // Error code
CORSERVER hSystem; // System server handle
CORDISPLAY hDisplay;       // Display handle
CORBUFFER hBuffer; // Buffer handle
CORVIEW hView;     // View handle

CCorViewDlg::CCorViewDlg()
{
// Initialize Sapera API
status = CorManOpen();

// Other initialization
...

// Get system server handle
hSystem = CorManGetServer();

// Get display handle
status = CorDisplayGetHandle(hSystem, 0, &hDisplay);

// Create a 640x480/8-bit monochrome buffer in system memory
status = CorBufferNew(hSystem, 640, 480, CORBUFFER_VAL_FORMAT_UINT8,
CORBUFFER_VAL_TYPE_VIRTUAL,
&hBuffer);

// Create a view handle
status = CorViewNew(hSystem, hDisplay, hBuffer, CORVIEW_VAL_MODE_AUTO_DETECT, &hView);
}

CCorViewDlg::~CCorViewDlg()
{
CORSTATUS status;      // Error code

// Release handles when finished
status = CorViewFree(hView);      // Should be freed first
status = CorBufferFree(hBuffer);
status = CorDisplayRelease(hDisplay);

// Close Sapera API
status = CorManClose();
}

BOOL CCorViewDlg::OnInitDialog()
{
// Call default handler
CDialog::OnInitDialog();
```

```
// Other initialization
...

// Set HWND parameter to window's handle
status = CorViewSetPrm(hView, CORVIEW_PRM_HWND, (UINT32)GetSafeHwnd());

return TRUE;
}

void CCorViewDlg::OnPaint()
{
if (IsIconic())
{
...
}
else
{
// Optionally call the default handler to paint a background
CDialog::OnPaint();

// Update view area
CorViewOnPaint(hView);
}
}

void CCorViewDlg::OnSize(UINT nType, int cx, int cy)
{
// Call default handler
CDialog::OnSize(nType, cx, cy);

// Fit destination window to window's client area
CRect cli;
GetClientRect(cli);
status = CorViewSetPrm(hView, CORVIEW_PRM_ROI_DST_LEFT, cli.left);
status = CorViewSetPrm(hView, CORVIEW_PRM_ROI_DST_TOP,  cli.top);
status = CorViewSetPrm(hView, CORVIEW_PRM_ROI_DST_WIDTH, cli.Width());
status = CorViewSetPrm(hView, CORVIEW_PRM_ROI_DST_HEIGHT, cli.Height());

// Update displayed area
CorViewOnSize(hView);
}

void CCorViewDlg::OnMove(int x, int y)
{
// Call default handler
CDialog::OnMove(x, y);

// Update displayed area
CorViewOnMove(hView);
}
```

Working with Buffers

Root and Child Buffers

A buffer is created in one of two ways: either as a root buffer (with no parent) or as a child buffer (with a parent). The parent of the child may also be a child itself, which allows you to build a buffer hierarchy with no restriction on the number of levels. A buffer can have more than one child buffer.

A child buffer shares the same memory space as its parent, and it defines an adjustable rectangular area within the root buffer. A child may be used by a processing function in order to process a region of interest. The example below shows how to create a root buffer with two child buffers.

Note: Child buffers must be freed before the root. If not, the root will return an error and will not be freed.

```
CORSTATUS status;                       // Status code
CORSERVER hServer;                      // Server handle
CORBUFFER hBuffer;                      // Root buffer handle
CORBUFFER hChildLeft, hChildRight;      // Child buffer handles

// Initialize Sapera API
status = CorManOpen();

// Get server handle
hServer = CorManGetServer();

// Create a 640x480/8-bit monochrome buffer
status = CorBufferNew(hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8, 0, &hBuffer);

// Create a child in the upper-left corner
status = CorBufferNewChild(hBuffer, 0, 0, 320, 240, &hChildLeft);

// Create a child in the upper-right corner
status = CorBufferNewChild(hBuffer, 320, 0, 320, 240, &hChildRight);

// Use buffers
...

// Free child buffers
status = CorBufferFree(hChildLeft);
status = CorBufferFree(hChildRight);

// Free root buffer
status = CorBufferFree(hBuffer);
```

```
// Close Sapera API
status = CorManClose();
```

Child buffer coordinates are accessed through four buffer parameters (*XMIN*, *YMIN*, *WIDTH*, and *HEIGHT*) that allow you to modify the position and size of the rectangle. The following example demonstrates several manipulations of the child buffers from the previous example.

```
// Swap buffers (left/right)
status = CorBufferSetPrm(hChildLeft, CORBUFFER_PRM_XMIN, 320);
status = CorBufferSetPrm(hChildRight, CORBUFFER_PRM_XMIN, 0);

// Set buffer as high as root
status = CorBufferSetPrm(hChildLeft, CORBUFFER_PRM_HEIGHT, 480);
status = CorBufferSetPrm(hChildRight, CORBUFFER_PRM_HEIGHT, 480);
```

Buffer Types

Various types of buffers can be created. The type of buffer created illustrates how it will be allocated and how it can be used with other modules, such as the Transfer or View modules.

Contiguous Memory Buffers

The buffer is allocated in contiguous memory. This means that the buffer is contained in a single, contiguous block of physical memory. The allocation mode allows the Transfer module to access the buffer through an efficient low-level process, for example, during an acquisition task. It is required to specify CORBUFFER_VAL_TYPE_CONTIGUOUS to allocate a buffer in contiguous memory when creating the buffer. Buffer size is limited by the amount of contiguous memory available which in turn is limited to one third of the total physical memory, up to 120MB. Use a scatter-gather buffer type to allocate large size buffers.

Scatter-Gather Memory Buffers

A buffer may be allocated in paged pool memory. This means that the buffer is composed of many 4K byte memory blocks (pages) that are locked in physical memory by the Buffer module. This particular allocation mode allows the Transfer module to access the buffer through an efficient low-level process, for example, during an acquisition task. It is required to specify CORBUFFER_VAL_TYPE_SCATTER_GATHER to allocate a scatter-gather buffer when creating a new buffer. Note that a scatter-gather buffer can be very large since it uses paged pool memory.

Virtual Buffers

Similar to a scatter-gather buffer except that pages of memory are not locked. This type of buffer permits the allocation of very large buffers; however, these buffers cannot be used as a source/destination for the transfer resource. It is required to specify CORBUFFER_VAL_TYPE_VIRTUAL to allocate a virtual buffer when creating the new buffer. This type of buffer may be used, for example, to store an image resulting from a processing operation. If you supply a contiguous scatter-gather or virtual buffer to CorViewNew, the View resource created will ensure that any pixel format can be displayed, sometimes at the expense of higher CPU utilization.

Offscreen and Overlay Buffers

These buffer types use DirectDraw (which must be installed on the computer) to exploit the hardware acceleration provided by the display adapter. Note that these buffers are subject to some restrictions. Before creating this buffer type, verify that the CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN or CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY parameters list the pixel formats that can be displayed efficiently without software conversion. The parameters depend on the display adapter and its current display mode (256 colors, 16, 24, or 32 bits). Note that a buffer created using any of those types can be used in low-level transfer processes, such as an acquisition task.

If the display device supports DirectDraw and CORBUFFER_VAL_TYPE_OFFSCREEN is specified when a buffer is created, the buffer will be allocated in system memory. The View module created using a buffer of this type tries to use the display adapter's hardware to copy the buffer's contents from system memory to video display memory. A system memory offscreen buffer can be created using any pixel format; however, calling *CorViewShow* with its corresponding view will take longer to execute if its pixel format is not listed in the CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN parameter.

Off-Screen Buffers in Video Memory

The buffer is allocated in offscreen video memory if CORBUFFER_VAL_TYPE_OFFSCREEN and CORBUFFER_VAL_TYPE_VIDEO is specified when creating the buffer (the two values should be ORed). The View module created using a buffer of this type uses the display adapter's hardware to perform a fast copy from video memory to video display memory. Typically, a buffer of this type is used when a graphical element is reused for several consecutive frames without modification. In this case, it is more efficient to keep this element in video memory and use the hardware to copy it to the appropriate position in each frame.

Note: If the display is in 256 color mode and 8-bit offscreen buffers are used, care should be taken to make certain that the buffers do not contain pixels with values within the 0-9 and 246-255 ranges. These values are reserved for Windows system colors and will not be displayed correctly.

Overlay Buffers

The buffer is allocated in video memory. Once a View module is created using this buffer and *CorViewShow* is initially called, the display adapter's overlay hardware will keep updating the display with the buffer's contents without additional *CorViewShow* calls. Note that the pixel format of an overlay buffer must be listed in the CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY parameter. Typically, overlay buffers will support more pixel formats (like YUV) than offscreen buffers. Color keying is supported by overlays as well. The behaviour of the overlay regarding key colors is determined by the CORVIEW_PRM_OVERLAY_MODE parameter of the View module resource associated with the buffer.

Dummy Buffers

No memory is allocated for a dummy buffer in order that it does not contain any data elements. However, all of its size and format parameters are still valid. This means that any Sapera functionality from other modules that need access to buffer data elements will not work. The only exception is the Transfer module, which may use dummy buffers as placeholders when no data is to be physically transferred.

Reading and Writing a Buffer

The following describes accessing simple buffer data as well as accessing buffer data using pointers.

Simple Buffer Data Access

The simplest way to read or write data to a buffer is by accessing it element by element. The CorBufferReadElement and CorBufferWriteElement functions are used to read and write a single elements to a buffer, respectively. The following examples demonstrate how to access data in an 8-bit monochrome buffer.

```
CORSTATUS status;     // Status code
CORSERVER hServer;    // Server handle
CORBUFFER hBuffer;    // Buffer handle
UINT8 value;          // Unsigned character to store 8-bit value

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Create a 640x480/8-bit monochrome buffer
status = CorBufferNew(hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8,
CORBUFFER_VAL_TYPE_VIRTUAL, &hBuffer);

// Write a constant value at a specific position
value = 0x80;
status = CorBufferWriteElement(hBuffer, 100, 200, &value, sizeof( value));

// Read back the value
status = CorBufferReadElement(hBuffer, 100, 200, &value, sizeof( value));

// Free buffer
status = CorBufferFree(hBuffer);

// Close Sapera API
status = CorManClose();
```

Accessing buffer data in this way is quite straightforward but, unfortunately, it considerably slows down access time. Alternately, you can access data by reading/writing an array of elements with only one function call through the Buffer module's CorBufferRead and CorBufferWrite functions. Below is a sample code illustrating the usage of these functions.

```
CORSTATUS status;     // Status code
CORSERVER hServer;    // Server handle
CORBUFFER hBuffer;    // Buffer handle
UINT8 *array;         // Character array to store 8-bit values
UINT32 size;          // Size of the array in bytes

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...
```

```
// Create a 640x480/8-bit monochrome buffer
status = CorBufferNew(hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8,
CORBUFFER_VAL_TYPE_VIRTUAL, &hBuffer);

// Create an array the same size as the buffer
size = 640 * 480 * sizeof( UINT8);
array  = (BYTE *) malloc(size);

// Fill array with values
...

// Write array to buffer
status = CorBufferWrite(hBuffer, 0, array, size);

// Read back array from buffer
status = CorBufferRead(hBuffer, 0, array, size);

// Free array and buffer
free(array);
status = CorBufferFree(hBuffer);

// Close Sapera API
status = CorManClose();
```

Buffer Data Access Using Pointers

Another way to access data stored in a buffer is to get a pointer to the buffer's memory by retrieving the value of its CORBUFFER_PRM_ADDRESS parameter. If the buffer has been allocated into video memory (i.e., an offscreen-video buffer or an overlay buffer), it must be locked before its address can be obtained. A buffer is locked by setting its CORBUFFER_PRM_LOCKED to a non-zero value.

```
CORSTATUS status;      // Status code
CORSERVER hServer;     // Server handle
CORBUFFER hBuffer;     // Buffer handle
UINT16 *dataPtr;       // pointer to buffer memory
UINT8  *basePtr;       // pointer to buffer memory
UINT32 pitch;          // width of buffer created
UINT32 i,j;

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Create a 640x480/16-bit RGB 565 buffer in video memory
// Display should also be 16 bits

status = CorBufferNew(hServer, 640, 480, CORBUFFER_VAL_FORMAT_RGB565,
CORBUFFER_VAL_TYPE_OFFSCREEN | CORBUFFER_VAL_TYPE_VIDEO, &hBuffer);

// Get the pitch of the surface
status = CorBufferGetPrm(hBuffer, CORBUFFER_PRM_PITCH, &pitch);

// Lock buffer since it is in video memory
status = CorBufferSetPrm(hBuffer, CORBUFFER_PRM_LOCKED, TRUE);
```

```
// Get address of the buffer's memory
status = CorBufferGetPrm(hBuffer, CORBUFFER_PRM_ADDRESS, &basePtr);

for(i=0;i<480;i++)
    {
    dataPtr = (UINT16*)(basePtr + i*pitch);
    for(j=0;j<640;j++)
    {
        // Process the line pointed to by dataPtr
        ...
    }
}

// Unlock buffer
status = CorBufferSetPrm(hBuffer, CORBUFFER_PRM_LOCKED, FALSE);

// note: at this point, dataPtr should not be used anymore

// Free buffer
status = CorBufferFree(hBuffer);

// Close Sapera API
status = CorManClose();
```

Additional buffer functions allow you to read and write specific data structures such as lines, rectangles, and dots.

Drawing Within Images

Overview

The Graphic module manages the graphic capabilities of the Sapera library. It permits you to define the graphic attributes and provides methods for drawing dots, lines, rectangles, and similar geometric shapes to a specified buffer. Graphic attributes include background color, foreground color, and operation mode.

In order to apply a drawing operator, the user must:

- Initialize the graphic subsystem.
- Set the graphic attributes.
- Specify a buffer where the shape or the vector representation will be drawn. Buffer formats currently supported by the Graphic module are listed below:

Supported Format	Corresponding Buffer Format
Unsigned 8 bits/pixel	CORBUFFER_VAL_FORMAT_UINT8
Unsigned 16 bits/pixel	CORBUFFER_VAL_FORMAT_UINT16
Signed 8 bits/pixel	CORBUFFER_VAL_FORMAT_INT8
Signed 16 bits/pixel	CORBUFFER_VAL_FORMAT_INT16
Color, 24 bits/pixel	CORBUFFER_VAL_FORMAT_RGB888
Color, 32 bits/pixel	CORBUFFER_VAL_FORMAT_RGB8888

- Use a graphic drawing operator.

Initializing the Graphic

Initializing the graphic consists of getting a handle to a graphic section of a specific server. Graphic attributes are set to default values at this stage.

```
CORSTATUS  status;     // Declare error code
CORSERVER  hServer;    // Declare a server handle
CORGRAPHIC hGra;       // Declare a graphic subsystem handle
```

```
// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Get graphic resource handle
status = CorGraphicGetHandle(hServer, 0, &hGra);


...

// Release graphic handle when finished
status = CorGraphicRelease(hGra);

// Close Sapera API
status = CorManClose();
```

Defining Graphic Attributes

Graphic attributes can be modified by the CorGraphicSetPrm and CorGraphicSetFont functions. The C code shown below illustrate an example of how to set graphic attributes.

```
// fontTab.h contains the description of a font created using the DALSA
// Font Generator Program
#include <fontTab.h>
...
CORSTATUS  status;     // Declare error code
CORSERVER  hServer;    // Declare a server handle
CORGRAPHIC hGra;       // Declare a graphic subsystem handle

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...


// Get graphic resource handle
status = CorGraphicGetHandle(hServer, 0, &hGra);

// Set the background color to black
status = CorGraphicSetPrm  ( hGra, CORGRAPHIC_PRM_BKCOLOR, 0);

// Set the foreground color to white
status = CorGraphicSetPrm  ( hGra, CORGRAPHIC_PRM_COLOR, 0xFFFFFF);

// Set the operation mode: when a drawing operator write a pixel in a buffer,
// it replaces its value by the foreground color
status = CorGraphicSetPrm  ( hGra, CORGRAPHIC_PRM_OPM, CORGRAPHIC_VAL_OPM_REP);

// Select a font that as been saved as a binary file and store on disk
// Next two function calls can be used interchangeably
status= CorGraphicSetPrm( hGra, CORGRAPHIC_PRM_FONTNAME, "C:\\Font.CFN");
status= CorGraphicSetFont( hGra, "C:\\Font.CFN", NULL, 0);
...
```

```
// Select a font that has been generated and saved as a text file using the
// DALSA Font Generator program and then included into this C source
// file (fontTab.h).
status= CorGraphicSetFont( hGra, "Font", fontTab, sizeof( fontTab));
...

// Release graphic handle when finished
status = CorGraphicRelease(hGra);

// Close Sapera API
status = CorManClose();
```

Drawing Shapes

The C code in the figure below illustrates how to draw a line and a rectangle in a specified buffer:

```
CORSTATUS  status;     // Declare error code
CORSERVER  hServer;    // Declare a server handle
CORGRAPHIC hGra;       // Declare a graphic subsystem handle
CORBUFFER  hBuffer;    // Declare a buffer

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Get graphic resource handle
status = CorGraphicGetHandle(hServer, 0, &hGra);

// Create a 24 bit/pixel color buffer
status  =  CorBufferNew( hServer, 640, 480, CORBUFFER_VAL_FORMAT_RGB888, 0, &hBuffer);

// Draw a line in the specified buffer
status = CorGraphicLine( hGra, hBuffer, 10,  10, 600, 400);

// Draw a rectangle in the specified buffer
status = CorGraphicRect( hGra, hBuffer, 10, 10, 600, 400, FALSE);

...

// Free up hBuffer
status= CorBufferFree( hBuffer);

// Release graphic handle when finished
status = CorGraphicRelease(hGra);

// Close Sapera API
status = CorManClose();
```

Drawing Vectors

The C code in the figure below illustrates how to draw a sample histogram vector in a specified buffer:

```
CORSTATUS  status;     // Declare error code
CORSERVER  hServer;    // Declare a server handle
CORGRAPHIC hGra;       // Declare a graphic subsystem handle
CORPRO hPro;           // Declare a processing subsystem handle
CORBUFFER  hBuffer;    // Declare a buffer
CORBUFFER  histVect;   // Declare a histogram buffer
FLOAT      histMean, histSd, histMin, histMax;

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Get graphic resource handle
status = CorGraphicGetHandle(hServer, 0, &hGra);

// Get processing resource handle
status = CorProGetHandle(hServer, 0, &hPro);


// Create a new unsigned 8 bit/pixel buffer
status  =  CorBufferNew( hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8, 0, &hBuffer);

// Create a histogram buffer
status  =  CorBufferNew( hServer, 256, 1, CORBUFFER_VAL_FORMAT_POINT, 0, &histVect);

...

// Compute the histogram
status = CorProHistogram( hPro, hBuffer, histVect, CORPRO_HIST_NORMAL, 256, (FLOAT)0,
(FLOAT)255);

// Compute the histogram min and max values
status = CorProStats( hPro, histVect, &histMean, &histSd, &histMin, &histMax);

// Draw a histogram vector in the specified buffer
status = CorGraphicDrawVector( hGra, hBuffer, histVect, (INT32)histMin,
(INT32)histMax, 256);

...

// Free up buffers
...

// Release graphic and processing handles when finished
...
// Close Sapera API
status = CorManClose();
```

Drawing Text

The C code in the figure below illustrates how to draw text in a specified buffer:

```
// Include font saved as a text file

#include <fontTab.h>
CORSTATUS  status;     // Declare error code
CORSERVER  hServer;    // Declare a server handle
CORGRAPHIC hGra;       // Declare a graphic subsystem handle
CORBUFFER  hBuffer;    // Declare a buffer

// Initialize Sapera API
status = CorManOpen();

// Get server handle
...

// Get graphic resource handle
status = CorGraphicGetHandle(hServer, 0, &hGra);


// Create a new unsigned 8 bit/pixel buffer
status = CorBufferNew( hServer, 640, 480, CORBUFFER_VAL_FORMAT_UINT8, 0, &hBuffer);

// Draw a text string in the specified buffer
status = CorGraphicText( hGra, hBuffer, 100, 100, "Text string");

...
// Loading a font from disk
status = CorGraphicSetPrm( hGra, CORGRAPHIC_PRM_FONTNAME, "C:\\newFont.cfn");
status = CorGraphicText( hGra, hBuffer, 100, 200, "Text string");


// Loading a font from fontTab array
status = CorGraphicSetFont( hGra, "New Font", fontTab, sizeof( fontTab));
status = CorGraphicText( hGra, hBuffer, 100, 300, "Text string");

// Free up buffers
...

// Release graphic and processing handles when finished
...
// Close Sapera API
status = CorManClose();
```

Sapera Frame Grabber Acquisition API

Acquisition Parameters & Capabilities

This section describes the functions of the Acquisition, Camera, and VIC Modules. The parameters and capabilities are described in the *Sapera LT Acquisition Parameters Reference Manual*.

Acquisition Functions

Function	Description
CorAcqDetectSync	*Auto detection of sync timings*
CorAcqFreeFlatfield	*Deallocate a flatfield resource from an acquisition device*
CorAcqGetCamIOControl	*Gets value of a custom camera I/O control*
CorAcqGetCap	*Gets acquisition capability value from an acquisition device*
CorAcqGetCount	*Gets the number of acquisition devices on a server*
CorAcqGetFlatfield	*Get the gain and offset values for a flatfield resource*
CorAcqGetHandle	*Gets a handle to an acquisition device*
CorAcqGetLut	*Gets input LUT values from an acquisition device*
CorAcqGetPrm	*Gets acquisition parameter value from an acquisition device*
CorAcqGetPrms	*Gets camera-dependent and VIC-dependent parameters from an acquisition device*
CorAcqGetSeialPortName	*Retrives the serial port name used by an acquisition device*
CorAcqNewFlatfield	*Create a flatfield resource for an acquisition device*
CorAcqRegisterCallback	*Register callback function for an acquisition resource*
CorAcqRelease	*Releases handle to an acquisition device*
CorAcqReset	*Resets an acquisition device*
CorAcqResetModule	*Resets the resources associated with the server's acquisition device(s)*
CorAcqSetCamIOControl	*Sets value of a custom camera I/O control*
CorAcqSetFlatfield	*Set the gain and offset values for a flatfield resource*
CorAcqSetLut	*Sets input LUT values for an acquisition device*

CorAcqSetPrm	*Sets a simple acquisition parameter of an acquisition device*
CorAcqSetPrmEx	*Sets a complex acquisition parameter of an acquisition device*
CorAcqSetPrms	*Sets camera-dependent and VIC-dependent parameters of an acquisition device*
CorAcqSoftwareTrigger	*Simulate a trigger to the acquisition device.*
CorAcqUnlock	*Unlocks acquisition parameters of an acquisition device*
CorAcqUnregisterCallback	*Unregister callback function for an acquisition resource*

CorAcqDetectSync

Detect and measure horizontal and vertical synchronization signal timings

Prototype CORSTATUS **CorAcqDetectSync** (CORACQ *hAcq*, PCORACQ_DETECT_SYNC *detectSync*);

Description This function performs an auto-detection of the input frame type (interlaced or progressive) and of its horizontal and vertical synchronization signal timings. Based on the return values, an appropriate camera file can be selected.

Input	`hAcq`	Acquisition resource handle
Output	`detectSync`	Pointer to a structure to receive the timing information (see also CORACQ_DETECT_SYNC in the *Sapera LT Acquisition Parameters Reference Manual*)

Return Value
CORSTATUS_ARG_NULL (if `detectSync` is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_NOT_IMPLEMENTED

Note If this function returns CORSTATUS_OK, but the returned horizontal and/or vertical sync period is set to 0, then the auto-detection function failed.

CorAcqFreeFlatfield

Deallocate a flatfield resource from an acquisition device

Prototype CORSTATUS **CorAcqFreeFlatfield** (CORACQ *hAcq*, UINT32 flatfieldNumber);

Description Deallocate a flatfield resource for an acquisition device that was allocated with CorAcqNewFlatfield.

Input	*hAcq*	Acquisition resource handle
	flatfieldNumber	The resource's flatfield number
Output	*None*	

Return Value
CORSTATUS_ARG_OUT_OF_RANGE
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY
CORSTATUS_NOT_AVAILABLE

See Also CorAcqNewFlatfield, CorAcqSetFlatfield and CorAcqGetFlatfield

CorAcqGetCamIOControl

Get value of a custom camera I/O control

Prototype	CORSTATUS **CorAcqGetCamIOControl**(CORACQ *hAcq*, PCSTR *label*, UINT32 **value*);	
Description	Gets the current value of a custom camera I/O control.	
Input	`hAcq`	Acquisition resource handle
	`label`	String specifying the label of the custom camera I/O control to get the value from.
Output	`value`	Current value of the custom camera I/O
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_NULL (if *label* or *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED	
Note	See Data Structures in the *Sapera LT Acquisition Parameters Reference Manual*, for retrieving correct values with specific macros.	

CorAcqGetCap

Get acquisition capability value from an acquisition device

Prototype	CORSTATUS **CorAcqGetCap**(CORACQ *hAcq*, UINT32 *cap*, void **value*);	
Description	Gets acquisition capability value from an acquisition device.	
Input	*hAcq*	Acquisition resource handle
	cap	Acquisition device capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_CAP_INVALID	

CorAcqGetCount

Get the number of acquisition devices on a server

Prototype	CORSTATUS **CorAcqGetCount**(CORSERVER *hServer*, UINT32 **count*);	
Description	Gets the number of acquisition devices available on a server.	
Input	*hServer*	Server handle
Output	*count*	Number of acquisition devices. The value of *count* is 0 when no acquisition device is available.
Return Value	CORSTATUS_NOT_IMPLEMENTED (when there is no Acquisition devices supported by the specified server) CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *count* is NULL)	

CorAcqGetFlatfield

Get the gain and offset values for a flatfield resource

Prototype	CORSTATUS CorAcqGetFlatfield (CORACQ hAcq, UINT32 flatfieldNumber, CORBUFFER hBufferGain, CORBUFFER hBufferOffset);	
Description	Reads the gain and offset values for each pixel from the allocated flatfield resource.	
Input	*hAcq*	Acquisition resource handle
	flatfieldNumber	Flatfield Number
Output	*hBufferGain*	Buffer resource handle. The buffer contains the flatfield gain values for each pixel in the flatfield resource.
	hBufferOffset	Buffer resource handle. The buffer contains the flatfield offset values for each pixel in the flatfield resource.
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_NOT_AVAILABLE CORSTATUS_ARG_INCOMPATIBLE	
See Also	CorAcqNewFlatfield, CorAcqFreeFlatfield and CorAcqSetFlatfield	

CorAcqGetHandle

Get a handle to an acquisition device

Prototype	CORSTATUS **CorAcqGetHandle**(CORSERVER *hServer*, UINT32 *index*, CORACQ **hAcq*);	
Description	Gets a handle to an acquisition device.	
Input	*hServer*	Server handle
	index	Specifies the acquisition device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorAcqGetCount.
Output	*hAcq*	Acquisition resource handle
Return Value	CORSTATUS_ARG_NULL CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_ACCESSIBLE CORSTATUS_RESOURCE_IN_USE	
See Also	CorAcqGetCount, CorAcqRelease	

CorAcqGetLut

Get input LUT values from an acquisition device

Prototype	CORSTATUS **CorAcqGetLut**(CORACQ *hAcq*, CORLUT *hLut*, UINT32 *lutNumber*);	
Description	Gets input LUT values from an acquisition device.	
Input	*hAcq*	Acquisition resource handle
	lutNumber	LUT number to get values from
Output	*hLut*	LUT resource handle
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INCOMPATIBLE_LUT CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_NOT_AVAILABLE	
Note	The LUT number value range is [0...CORACQ_PRM_LUT_MAX-1].	
See Also	CorAcqSetLut	

CorAcqGetPrm

Get acquisition parameter value from an acquisition device

Prototype	CORSTATUS **CorAcqGetPrm**(CORACQ *hAcq*, UINT32 *prm*, void **value*);	
Description	Gets acquisition parameter (simple and complex) value from an acquisition device. Make certain that *value* points to a data structure that matches the data type of the specified acquisition parameter. See Data Structures in the *Sapera LT Acquisition Parameters Reference Manual*, for retrieving correct values with specific macros.	
Input	*hAcq*	Acquisition resource handle
	prm	Acquisition parameter requested
Output	*value*	Current value of the requested parameter
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_PRM_INVALID	
See Also	CorAcqSetPrm and CorAcqSetPrmEx	

CorAcqGetPrms

Get camera-dependent and VIC-dependent parameter values from an acquisition device

Prototype	CORSTATUS **CorAcqGetPrms**(CORACQ *hAcq*, CORVIC *hVic*, CORCAM *hCam*, UINT32 *toLock*);
Description	Gets camera-dependent and VIC-dependent parameter values from an acquisition device. The VIC and camera resource handles can be any valid handle obtained through CorVicNew and CorCamNew respectively. If the acquisition parameters are locked (*toLock* = TRUE), then the acquisition parameters

cannot be modified by CorAcqSetPrms and CorAcqSetPrmEx. The only way to modify the parameters is to use CorAcqSetPrms with the same VIC and CAM handles that were used when locking the parameters. Refere to the CorAcqSetPrms for a more detailed explanation about the locking mechanism.

Input	*hAcq*	Acquisition resource handle
	toLock	Lock acquisition parameters (TRUE or FALSE)
Output	*hVic*	VIC resource handle
	hCam	Camera resource handle
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_PARAMETERS_LOCKED	
See Also	CorAcqSetPrms, CorAcqSetPrm, CorAcqSetPrmEx, CorVicNew and CorCamNew	

CorAcqGetSeialPortName

Retrives the serial port name used by an acquisition device

Prototype	CORSTATUS **CorAcqGetSerialPortName**(CORACQ *hAcq*, UINT32 *portNameLengthMax*, char* *szSerialPortName*);	
Description	Copies the name of the serial port used with the acquisition device.	
Input	*hAcq*	Acquisition resource handle
	portNameLengthMax	Maximum number of characters to copy
Output	*szSerialPortName*	Name of the serial port
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED	

CorAcqNewFlatfield

Allocate a flatfield resource for an acquisition resource

Prototype	CORSTATUS **CorAcqNewFlatfield** (CORACQ *hAcq*, UINT32 *pFlatfieldNumber);	
Description	Allocate a flatfield resource for an acquisition resource	
Input	*hAcq*	Acquisition resource handle
Output	*pFlatfieldNumber*	Pointer to a flatfield number
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_NOT_AVAILABLE	
See Also	CorAcqFreeFlatfield, CorAcqSetFlatfield and CorAcqGetFlatfield	

CorAcqRegisterCallback

Register callback function for an acquisition resource

Prototype	CORSTATUS **CorAcqRegisterCallback**(CORACQ *hAcq*, UINT32 *eventType*, PCORCALLBACK *callbackFct*, void *context);	
Description	Registers callback function for the specified acquisition resource.	
Input	*hAcq*	Acquisition resource handle
	eventType	Type of event to register. See CORACQ_PRM_EVENT_TYPE in the *Sapera LT Acquisition Parameters Reference Manual*.
	callbackFct	Callback function to register. The callback function is defined as follows: *CORSTATUS CCONV* *callback(void *context, UINT32 eventType, UINT32 eventCount);* When called, *context* will have the value specified at the callback function registration; *eventType* will contain the event(s) that trigged the call to the callback function; *eventCount* should increment by one at each call, with a starting value of 1. In case the acquisition resource cannot keep up because there are too many events to be signaled, *eventCount* will take non- consecutive values, indicating that events have been lost. See the Data Types section for the PCORCALLBACK definition.
	context	Context pointer passed to the callback function when called
Output	None	
Return Value	CORSTATUS_ARG_NULL CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_AVAILABLE CORSTATUS_RESOURCE_IN_USE	
Note	The values may be ORed if more than one event is desired. Moreover, when used, CORACQ_VAL_EVENT_TYPE_END_OF_LINE must be ORed with an *unsigned* integer representing the line on which the callback function has to be called while CORACQ_VAL_EVENT_TYPE_END_OF_NLINES must be ORed with an *unsigned* integer representing the number of lines after which the callback function has to be called.	
See Also	CorAcqUnregisterCallback	

CorAcqRelease

Release handle to an acquisition device

Prototype	CORSTATUS **CorAcqRelease**(CORACQ *hAcq*);	
Description	Releases handle to an acquisition device.	
Input	*hAcq*	Acquisition resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorAcqGetHandle	

CorAcqReset

Reset an acquisition device

Prototype	CORSTATUS **CorAcqReset**(CORACQ *hAcq*);
Description	Reset and restore the default acquisition parameter values of the specified acquisition device.
Input	*hAcq* Acquisition resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_PARAMETERS_LOCKED CORSTATUS_SOFTWARE_ERROR
See Also	CorAcqResetModule

CorAcqResetModule

Reset the resources associated with the server's acquisition device(s)

Prototype	CORSTATUS **CorAcqResetModule** (CORSERVER *hServer*);
Description	This releases all the resources (handle, memory) currently allocated. Before using this function, make certain that no other application is currently using any acquisition device resource. Proceed with caution when using this function.
Input	*hServer* Server handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorAcqReset

CorAcqSetCamIOControl

Set value of a custom I/O control

Prototype	CORSTATUS **CorAcqSetCamIoControl**(CORACQ hAcq, PCSTR label, UINT32 value)
Description	Sets the state of a custom I/O control that was previously defined by the Camera parameter CORACQ_PRM_CAM_IO_CONTROL. Refer to the Custom Camera Control I/O Description section in the *Sapera LT Acquisition Parameters Reference Manual*, for a discussion about custom I/O control.
Input	*hAcq* Acquisition resource handle *label* String specifying the label of the custom camera I/O control *value* Value to write to the custom camera I/O control
Output	None
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_NULL (if *label* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED CORSTATUS_PARAMETERS_LOCKED
Note	See the Data Structures section in the *Sapera LT Acquisition Parameters Reference Manual*, for building correct values with specific macros.

CorAcqSetFlatfield

Set the gain and offset values for a flatfield resource

Prototype	CORSTATUS **CorAcqGetFlatfield** (CORACQ hAcq, UINT32 flatfieldNumber, CORBUFFER hBufferGain, CORBUFFER hBufferOffset);	
Description	Writes the gain and offset values for each pixel to the allocated flatfield resource.	
Input	*hAcq*	Acquisition resource handle
	flatfieldNumber	Flatfield Number
	hBufferGain	Buffer resource handle. The buffer contains the flatfield gain values for each pixel in the flatfield resource.
	hBufferOffset	Buffer resource handle. The buffer contains the flatfield offset values for each pixel in the flatfield resource.
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_NOT_AVAILABLE CORSTATUS_ARG_INCOMPATIBLE	
See Also	CorAcqNewFlatfield, CorAcqFreeFlatfield and CorAcqGetFlatfield	

CorAcqSetLut

Set input LUT values for an acquisition device

Prototype	CORSTATUS **CorAcqSetLut**(CORACQ *hAcq*, CORLUT *hLut*, UINT32 *lutNumber*);	
Description	Sets input LUT values for an acquisition device.	
Input	*hAcq*	Acquisition resource handle
	hLut	LUT resource handle created with CorLutNew or CorLutNewFromFile.
	lutNumber	LUT number to write the values to. The LUT number value range is [0...CORACQ_PRM_LUT_MAX-1].
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_INCOMPATIBLE_LUT CORSTATUS_NO_MEMORY CORSTATUS_NOT_AVAILABLE	
See Also	CorAcqGetLut	

CorAcqSetPrm

Set a simple acquisition parameter of an acquisition device

Prototype	CORSTATUS **CorAcqSetPrm**(CORACQ *hAcq*, UINT32 *prm*, UINT32 *value*);
Description	Sets a simple acquisition device's parameter. A simple parameter is one that fits inside an

UINT32. If the parameter is complex, use CorAcqSetPrmEx.

Acquisition parameters are normally initialized all at once by CorAcqSetPrms. One may want to modify certain parameters at a later time, such as those related to camera swithing, brightness and contrast settings. In this case, CorAcqSetPrm (or CorAcqSetPrmEx) should used for faster execution.

Input

hAcq	Acquisition resource handle
prm	Acquisition parameter to set
value	New value of the parameter

Output None

Return Value

CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY
CORSTATUS_PARAMETERS_LOCKED
CORSTATUS_PRM_INVALID
CORSTATUS_PRM_INVALID_VALUE
CORSTATUS_PRM_MUTUALLY_EXCLUSIVE
CORSTATUS_PRM_NOT_AVAILABLE
CORSTATUS_ARG_OUT_OF_RANGE
CORSTATUS_PRM_READ_ONLY

Note See the section Data Structures in the *Sapera LT Acquisition Parameters Reference Manual*, for building correct values with specific macros.

See Also CorAcqGetPrm, CorAcqSetPrmEx and CorAcqSetPrms

CorAcqSetPrmEx

Set a complex acquisition parameter of an acquisition device

Prototype CORSTATUS **CorAcqSetPrmEx**(CORACQ *hAcq*, UINT32 *prm*, void **value*);

Description Sets a complex acquisition device's parameter. A complex parameter is greater in size than an UINT32. If the parameter size is UINT32, use either CorAcqSetPrm or CorAcqSetPrmEx.

Acquisition parameters are normally initialized all at once by CorAcqSetPrms One may want to modified certain parameters at a later time, such as those related to camera swithing, brightness and contrast settings. In this case, CorAcqSetPrmEx (or CorAcqSetPrm) should used for faster execution.

Input

hAcq	Acquisition resource handle
prm	Acquisition parameter to set
value	New value of the parameter

Output None

Return Value

CORSTATUS_ARG_NULL
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY
CORSTATUS_PARAMETERS_LOCKED
CORSTATUS_PRM_INVALID
CORSTATUS_PRM_INVALID_VALUE
CORSTATUS_PRM_MUTUALLY_EXCLUSIVE
CORSTATUS_PRM_NOT_AVAILABLE

CORSTATUS_ARG_OUT_OF_RANGE
CORSTATUS_PRM_READ_ONLY

See Also CorAcqGetPrm and CorAcqSetPrm

CorAcqSetPrms

Set camera-dependent and VIC-dependent parameter values of an acquisition device

Prototype CORSTATUS **CorAcqSetPrms**(CORACQ *hAcq*, CORVIC *hVic*, CORCAM *hCam*, UINT32 *toUnlock*);

Description Initializes an acquisition device with the specified camera-dependent and VIC-dependent parameter values.

The Sapera Development environment provides a Windows application called CameraExpert that allows the user to create/modify CAM and VIC files. Using CorVicLoad and CorCamLoad, the VIC and CAM resource handles can then be obtained from the CAM and VIC files respectively

If the acquisition parameters are locked (by a previous call to CorAcqGetPrms), then the same VIC and camera module handles that were used when locking the parameters initially must be used to set the parameters. Locking the parameters may allow faster hardware initialization depending on the actual device driver/hardware implementation because only the parameters that have changed since being locked will be reinitialized.

Also, in the specific case where the parameters are locked, the *toUnlock* flag is used to determined the state of the parameters upon returning from this function:
toUnlock = TRUE: unlock parameters
toUnlock = FALSE: keep parameters locked
Parameters can also be unlocked without any modifications by using CorAcqUnlock.

Input

hAcq	Acquisition resource handle
hVic	VIC resource handle
hCam	Camera resource handle
toUnlock	Unlock acquisition parameters (TRUE or FALSE)

Output None

Return Value
CORSTATUS_INVALID_HANDLE
CORSTATUS_PARAMETERS_LOCKED
CORSTATUS_PRM_INVALID_VALUE
CORSTATUS_PRM_MUTUALLY_EXCLUSIVE
CORSTATUS_PRM_NOT_AVAILABLE
CORSTATUS_ARG_OUT_OF_RANGE

See Also CorAcqGetPrms, CorAcqUnlock, CorVicNew and CorCamNew

CorAcqSoftwareTrigger

Simulate a trigger to the acquisition device

Prototype CORSTATUS **CorAcqSoftwareTrigger**(CORACQ *hAcq*, UINT32 triggerType);

Description Calling this function will simulate a hardware trigger to the acquisition device.

Input

hAcq	Acquisition resource handle

	UINT32	Type of trigger: CORACQ_CAP_SOFTWARE_TRIGGER.
Output	None	
Return Value	CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE	

CorAcqUnlock

Unlock acquisition parameters of an acquisition device

Prototype	CORSTATUS **CorAcqUnlock**(CORACQ *hAcq*, CORVIC *hVic*, CORCAM *hCam*);	
Description	Unlock the acquisition parameters previously locked by a call to CorAcqGetPrms.	
Input	*hAcq*	Acquisition device resource handle
	hVic	VIC resource handle
	hCam	Camera resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_PARAMETERS_LOCKED	
Note	The VIC and camera resource handles must be the same VIC and camera module handles that were used when locking the parameters with CorAcqGetPrms.	
See Also	CorAcqGetPrms and CorAcqSetPrms	

CorAcqUnregisterCallback

Unregister callback function for an acquisition resource

Prototype	CORSTATUS **CorAcqUnregisterCallback**(CORACQ *hAcq*, PCORCALLBACK *callbackFct*);	
Description	Unregisters the specified acquisition resource callback function.	
Input	*hAcq*	Acquisition resource handle
	callbackFct	Callback function to unregister. See Data Types section for the PCORCALLBACK definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorXferRegisterCallback	

Camera Module Functions

The Camera Module manages all parameters that define a camera, such as camera type, video timings, and sync info. These parameters are usually fixed for a given camera. Parameters may be saved to or retrieved from a file and used to configure an Acquisition Module.

This section describes the functions of the Camera Module. The parameters and capabilities are described in the *Sapera LT Acquisition Parameters Reference Manual*.

Function	Description
CorCamFree	*Releases handle to a camera resource*
CorCamGetPrm	*Gets camera parameter value from a camera resource*
CorCamLoad	*Loads camera parameters from a file into a camera resource*
CorCamNew	*Creates a new camera resource*
CorCamSave	*Saves to a file the camera parameters of a camera resource*
CorCamSetPrm	*Sets a simple camera parameter of a camera resource*
CorCamSetPrmEx	*Sets a complex camera parameter of a camera resource*

CorCamFree

Release handle to a camera resource

Prototype	CORSTATUS **CorCamFree**(CORCAM *hCam*);	
Description	Releases handle to a camera resource.	
Input	*hCam*	Camera resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorCamNew	

CorCamGetPrm

Get camera parameter value from a camera resource

Prototype	CORSTATUS **CorCamGetPrm**(CORCAM *hCam*, UINT32 *prm*, void **value*);	
Description	Gets a camera parameter value from a camera resource.	
Input	*hCam*	Camera resource handle
	prm	Camera parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID	

See Also CorCamSetPrm and CorCamSetPrmEx

CorCamLoad

Load camera parameters from a file into a camera resource

Prototype CORSTATUS **CorCamLoad**(CORCAM *hCam*, const char **filename*);

Description Fills the camera-dependent acquisition parameters from a file. The function does not initialize the actual hardware. The function CorAcqGetPrms must be called to perform the hardware initialization.

Input *filename* String specifying the path and filename

Output *hCam* Camera resource handle

Return Value CORSTATUS_ARG_NULL (if *filename* is NULL)
CORSTATUS_FILE_OPEN_ERROR
CORSTATUS_INVALID_HANDLE
CORSTATUS_NOT_IMPLEMENTED

See Also CorCamSave and CorCamNew

CorCamNew

Create a new camera resource

Prototype CORSTATUS **CorCamNew**(CORSERVER *hServer*, CORCAM **hCam*);

Description Allocates the data structure (dynamic resource) to store the camera parameters and returns the handle to it.

The data structure is allocated on the local server; that is, where the application is running (typically the Host computer). Camera parameters may be loaded from the "Camera file" using the CorCamLoad function. The *hCamc* handle is used by the CorAcqGetPrms function to initialize the acquisition device with the camera parameters.

Input *hServer* Server handle

Output *hCam* Camera resource handle

Return Value CORSTATUS_ARG_NULL (if *hCam* is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY

See Also CorCamFree, CorCamLoad and CorAcqGetPrms

CorCamSave

Save the camera parameters of a camera resource

Prototype CORSTATUS **CorCamSave**(CORCAM *hCam*, const char **filename*);

Description Saves to a file the camera-dependent acquisition parameters of a camera resource.

Input *hCam* Camera resource handle

filename String specifying the path and filename.

Output None

Return Value CORSTATUS_ARG_NULL (if *filename* is NULL)

CORSTATUS_FILE_WRITE_ERROR
CORSTATUS_INVALID_HANDLE
CORSTATUS_NOT_IMPLEMENTED

See Also CorCamLoad

CorCamSetPrm

Set a simple camera parameter of a camera resource

Prototype CORSTATUS **CorCamSetPrm**(CORCAM *hCam*, UINT32 *prm*, UINT32 *value*);

Description Sets a simple camera resource parameter. A simple parameter is one that can fit inside an UINT32. If the parameter is complex, use the function CorCamSetPrmEx.

The CorAcqSetPrms function must then be called to initialize the acquisition hardware with the CAM resource parameters. Alternatively, simple CAM parameters can be applied individually to the acquisition hardware using CorAcqSetPrm.

Input

hCam	Camera resource handle
prm	Camera parameter to modify
value	New value of the parameter

Output None

Return Value CORSTATUS_INVALID_HANDLE
CORSTATUS_PRM_INVALID

See Also CorCamGetPrm, CorCamSetPrmEx and CorAcqSetPrms

CorCamSetPrmEx

Set a complex camera parameter of a camera resource

Prototype CORSTATUS **CorCamSetPrmEx**(CORCAM *hCam*, UINT32 *prm*, void **value*);

Description Sets a complex camera resource parameter. A complex parameter is of size greater than an UINT32. If the parameter size is UINT32, use either CorCamSetPrm or CorCamSetPrmEx.

The CorAcqSetPrms function must then be called to initialize the acquisition hardware with the CAM resource parameters. Alternatively, complex CAM parameters may be applied individually to the acquisition hardware using CorAcqSetPrmEx.

Input

hCam	Camera resource handle
prm	Camera parameter to modify
value	New value of the parameter

Output None

Return Value CORSTATUS_ARG_NULL (if *value* is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_PRM_INVALID

See Also CorCamGetPrm, CorCamSetPrm and CorAcqSetPrms

VIC Module Functions

The VIC (Video Input Conditioning) Module manages all parameters that are camera-independent, such as brightness and contrast. Parameters may be saved to or retrieved from a file and used to configure the acquisition hardware through Acquisition Module functions.

This section describes the functions of the VIC Module. The parameters and capabilities are described in the *Sapera LT Acquisition Parameters Reference Manual*.

Function	Description
CorVicFree	Releases handle to a VIC resource
CorVicGetPrm	Gets VIC parameters value from a VIC resource
CorVicLoad	Loads VIC parameters from a file into a VIC resource
CorVicNew	Allocates the data structure to store the VIC parameters
CorVicSave	Saves to a file the VIC parameters of a VIC resource
CorVicSetPrm	Sets a simple VIC parameter of a VIC resource
CorVicSetPrmEx	Sets a complex VIC parameter of a VIC resource

CorVicFree

Release handle to a VIC resource

Prototype	CORSTATUS **CorVicFree**(CORVIC *hVic*);
Description	Releases the handle to a VIC resource.
Input	*hVic* VIC resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorVicNew

CorVicGetPrm

Get VIC parameter value from a VIC resource

Prototype	CORSTATUS **CorVicGetPrm**(CORVIC *hVic*, UINT32 *prm*, void **value*);
Description	Gets a VIC parameter value from a VIC resource.
Input	*hVic* VIC resource handle
	prm VIC parameter requested
Output	*value* Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID
See Also	CorVicSetPrm and CorVicSetPrmEx

CorVicLoad

Load VIC parameters from a file into a VIC resource

Prototype	CORSTATUS **CorVicLoad**(CORVIC *hVic*, const char **filename*);	
Description	Loads VIC parameters from a file into a VIC resource.	
Input	*filename*	String specifying the path and filename
Output	*hVic*	VIC resource handle
Return Value	CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_OPEN_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED	
See Also	CorVicSave	

CorVicNew

Create a new VIC resource handle

Prototype	CORSTATUS **CorVicNew**(CORSERVER *hServer*, CORVIC **hVic*);	
Description	Allocates the data structure (dynamic resource) to store the VIC parameters and returns the handle to it. The data structure is allocated on the local server, that is, where the application is running (typically the Host computer). The VIC parameters may be loaded from the "VIC file" using the CorVicLoad function. The *hVic* handle is used by the CorAcqGetPrms function to initialize the acquisition device with the VIC parameters.	
Input	*hServer*	Handle to the Server where the VIC resource (data structure) will be allocated.
Output	*hVic*	New VIC resource handle
Return Value	CORSTATUS_ARG_NULL (if *hVic* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
See Also	CorVicFree and CorVicLoad	

CorVicSave

Save to a file the VIC parameters of a VIC resource

Prototype	CORSTATUS **CorVicSave**(CORVIC *hVic*, const char **filename*);	
Description	Saves to a file the VIC resource parameters.	
Input	*hVic*	VIC resource handle
	filename	String specifying the path and filename
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_WRITE_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED	
See Also	CorVicLoad	

CorVicSetPrm

Set a simple VIC parameter of a VIC resource

Prototype	CORSTATUS **CorVicSetPrm**(CORVIC *hVic*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple VIC resource parameter. A simple parameter is one that fits inside an UINT32. If the parameter is complex, use CorVicSetPrmEx. The CorAcqSetPrms function must then be called to initialize the acquisition hardware with the VIC resource parameters. Alternatively, simple VIC parameters may be applied individually to the acquisition hardware using CorAcqSetPrm.	
Input	*hVic*	VIC resource handle
	prm	VIC parameter to modify
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID	
See Also	CorVicGetPrm, CorVicSetPrmEx, CorCamSetPrm and CorAcqSetPrms	

CorVicSetPrmEx

Set a complex VIC parameter of a VIC resource

Prototype	CORSTATUS **CorVicSetPrmEx**(CORVIC *hVic*, UINT32 *prm*, void **value*);	
Description	Sets a complex VIC resource parameter. A complex parameter has a size greater than an UINT32. If the parameter size is UINT32, use either CorVicSetPrm or CorVicSetPrmEx. The CorAcqSetPrms function must then be called to initialize the acquisition hardware with the VIC resource parameters. Alternatively, complex VIC parameters may be applied individually to the acquisition hardware using CorAcqSetPrmEx.	
Input	*hVic*	VIC resource handle
	prm	VIC parameter to modify
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID	
See Also	CorVicGetPrm, CorVicSetPrm, CorCamSetPrmEx and CorAcqSetPrms	

Sapera Camera Acquisition API

AcqDevice Module

The *AcqDevice* module functions provide read/write features from/to devices such as a GigE-Vision camera. The module also contains functions for sending commands and registering events to devices.

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

AcqDevice Parameters

ID	Parameters	Attribute
0x00	CORACQDEVICE_PRM_LABEL	Read-only
0x01	CORACQDEVICE_PRM_UPDATE_FEATURE_MODE	Read/write
0x02	CORACQDEVICE_PRM_CONFIG_NAME	Read/write
0x03	CORACQDEVICE_PRM_MODE_NAME	Read/write

CORACQDEVICE_PRM_LABEL

Description Acquisition device ID: Zero-terminated array of characters with a fixed size of 128 bytes.

Type CHAR[128]

Note This parameter is read-only.

CORACQDEVICE_PRM_CONFIG_NAME

Description Defines the configuration name to use when saving the device features using CorAcqDeviceSaveFeatures. 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”

When loading a configuration file using CorAcqDeviceLoadFeatures, this parameter is automatically updated.

Type CHAR[64]

See also CORACQDEVICE_PRM_MODE_NAME

CORACQDEVICE_PRM_MODE_NAME

Description Defines the mode name to use when saving the device features using CorAcqDeviceSaveFeatures. 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"

When loading a configuration file using CorAcqDeviceLoadFeatures, this parameter is automatically updated.

Type CHAR[64]

See also CORACQDEVICE_PRM_CONFIG_NAME

CORACQDEVICE_PRM_UPDATE_FEATURE_MODE

Description Defines the mode by which features are written to the device. In the automatic mode (CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_AUTO) every time a feature is set using the *CorAcqDeviceSetFeatureValue…* or *CorAcqDeviceSetFeatureData…* functions the feature value is immediately written to the device. In the manual mode (CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_MANUAL) each feature is temporarily cached until the *CorAcqDeviceUpdateFeaturesToDevice* is called to write all features to the device at once.

Type UINT32

Values CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_AUTO (0x00)
Each feature is written to the device individually without being cached.
CORACQDEVICE_VAL_UPDATE_FEATURE_MODE_MANUAL (0x01)
Each feature is temporarily cached until *CorAcqDeviceUpdateFeaturesToDevice* is called.

Note The default mode is automatic. However when setting a series of parameters that are related to each other (for instance a region-of-interest composed of four values) the manual mode is useful to avoid invalid temporary device states.

AcqDevice Functions

Function	Description
General Functions	
CorAcqDeviceResetModule	Resets the resources associated with the server's acquisition devices
CorAcqDeviceGetCount	Gets the number of acquisition devices on a server
CorAcqDeviceGetHandle	Gets a handle to an acquisition device with read/write access
CorAcqDeviceGetHandleReadOnly	Gets a handle to an acquisition device with read-only

	access
CorAcqDeviceRelease	Releases a handle to an acquisition device
CorAcqDeviceReset	Resets an acquisition device
Module Capability and Parameter Access Functions	
CorAcqDeviceGetCap	Gets a capability value from an acquisition device
CorAcqDeviceGetPrm	Gets a parameter value from an acquisition device
CorAcqDeviceSetPrm	Sets a simple parameter value to an acquisition device
CorAcqDeviceSetPrmEx	Sets a complex parameter value to an acquisition device
Feature Access Functions	
CorAcqDeviceGetFeatureCount	Returns the number of features supported by the acquisition device
CorAcqDeviceGetFeatureNameByIndex	Returns the name of a feature associated with a specified index
CorAcqDeviceGetFeatureIndexByName	Returns the index of a feature associated with a specified name
CorAcqDeviceIsFeatureAvailable	Returns whether or not a feature is supported by the acquisition device
CorAcqDeviceGetFeatureInfoByName	Returns information on a feature associated with a specified name
CorAcqDeviceGetFeatureInfoByIndex	Returns information on a feature associated with a specified index
CorAcqDeviceGetFeatureValueByName	Returns the value of a simple feature associated with a specified name
CorAcqDeviceGetFeatureValueByIndex	Returns the value of a simple feature associated with a specified index
CorAcqDeviceGetFeatureDataByName	Returns the value of a complex feature associated with a specified name
CorAcqDeviceGetFeatureDataByIndex	Returns the value of a complex feature associated with a specified index
CorAcqDeviceSetFeatureValueByName	Sets the value of a simple feature associated with a specified name
CorAcqDeviceSetFeatureValueByIndex	Sets the value of a simple feature associated with a specified index
CorAcqDeviceSetFeatureDataByName	Sets the value of an complex feature associated with a specified name
CorAcqDeviceSetFeatureDataByIndex	Sets the value of a complex feature associated with a specified index
CorAcqDeviceUpdateFeaturesToDevice	Sets all the features to the acquisition device at once
CorAcqDeviceUpdateFeaturesFromDevice	Gets all the features from the acquisition device at once
CorAcqDeviceLoadFeatures	Loads all the features from a configuration file
CorAcqDeviceSaveFeatures	Saves all (or a subset of) features to a configuration file.

Event Management Functions	
CorAcqDeviceGetEventCount	Returns the number of events supported by the acquisition device
CorAcqDeviceGetEventNameByIndex	Returns the name of an event associated with a specified index
CorAcqDeviceGetEventIndexByName	Returns the index of an event associated with a specified name
CorAcqDeviceIsEventAvailable	Returns whether or not an event is supported by the acquisition device
CorAcqDeviceRegisterCallbackByName	Registers a callback function for the event associated with a specified name
CorAcqDeviceRegisterCallbackByIndex	Registers a callback function for the event associated with a specified index
CorAcqDeviceUnregisterCallbackByName	Unregisters a callback function on the event associated with a specified name
CorAcqDeviceUnregisterCallbackByIndex	Unregisters a callback function on the event associated with a specified index
CorAcqDeviceIsCallbackRegisteredByName	Returns whether or not a callback function was registered on the event associated with a specified name
CorAcqDeviceIsCallbackRegisteredByIndex	Returns whether or not a callback function was registered on the event associated with a specified index

General Functions

The general functions are used to enumerate the acquisition devices in a system and enable/disable access to those devices.

CorAcqDeviceGetCount

Gets the number of server acquisition devices.

Prototype	CORSTATUS **CorAcqDeviceGetCount**(CORSERVER *server*, PUINT32 *count*);	
Description	Gets the number of acquisition devices available on a server.	
Input	*server*	Server handle
Output	*count*	Number of acquisition devices. The value of *count* is 0 when no acquisition device is available.
Return Value	CORSTATUS_NOT_IMPLEMENTED (when there is no Acquisition device supported by the specified server) CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *count* is NULL)	

CorAcqDeviceGetHandle

Gets a handle to an acquisition device with read/write access.

Prototype	CORSTATUS **CorAcqDeviceGetHandle**(CORSERVER *server*, UINT32 *index*, PCORACQDEVICE *acqDevice*);	
Description	Gets a handle to an acquisition device with read/write access. If the handle is already used by another application, use CorAcqDeviceGetHandleReadOnly to obtain read-only access to the device.	
Input	*server*	Handle to a server that contains an *AcqDevice* resource
	index	Specifies the acquisition device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorAcqDeviceGetCount.
Output	*acqDevice*	Acquisition resource handle
Return Value	CORSTATUS_ARG_NULL CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_ACCESSIBLE CORSTATUS_RESOURCE_IN_USE	
See Also	CorAcqDeviceGetCount, CorAcqDeviceRelease, CorAcqDeviceGetHandleReadOnly	

CorAcqDeviceGetHandleReadOnly

Gets a handle to an acquisition device with read-only access.

Prototype	CORSTATUS **CorAcqDeviceGetHandleReadOnly**(CORSERVER *server*, UINT32 *index*, PCORACQDEVICE *acqDevice*);	
Description	Gets a handle to an acquisition device with read-only access. If the handle is already used by another application use this function to obtain read-only access to the device. To know what functions of the *AcqDevice* module are accessible with this type of handle, refer to the function documentation.	
Input	*server*	Server handle
	index	Specifies the acquisition device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorAcqDeviceGetCount.
Output	*acqDevice*	Acquisition resource handle
Return Value	CORSTATUS_ARG_NULL CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_ACCESSIBLE	
See Also	CorAcqDeviceGetCount, CorAcqDeviceRelease, CorAcqDeviceGetHandle	

CorAcqDeviceRelease

Releases a handle to an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceRelease**(CORACQDEVICE *acqDevice*);
Description	Releases a handle to an acquisition device.

Input	*acqDevice* Acquisition resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorAcqDeviceGetHandle, CorAcqDeviceGetHandleReadOnly

CorAcqDeviceReset

Resets an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceReset**(CORACQDEVICE *acqDevice*);
Description	Reset and restore the default acquisition parameter values of the specified acquisition device.
Input	*acqDevice* Acquisition resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_SOFTWARE_ERROR
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceResetModule

CorAcqDeviceResetModule

Resets the resources associated with the server's acquisition devices.

Prototype	CORSTATUS **CorAcqDeviceResetModule**(CORSERVER *server*);
Description	Releases all resources (handle, memory) currently allocated. Before using this function, make certain that no other application is currently using any acquisition device resource. Proceed with caution when using this function.
Input	*hServer Server handle*
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceReset

Module Capability and Parameter Access Functions

The module capability and parameter access functions provide read/write capabilities and parameters for acquisition devices.

Note: *Capabilities* and *parameters* control the behavior of an *AcqDevice* module. They are different from *features* which describe the attributes of the hardware device itself.

CorAcqDeviceGetCap

Gets a capability value from an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceGetCap**(CORACQDEVICE *acqDevice*, UINT32 *cap*, void **value*);	
Description	Gets a capability value from an acquisition device. A capability defines the availability of the associated parameter.	
Input	*acqDevice*	Acquisition resource handle
	cap	Acquisition capability requested
Output	*value*	Value of the requested capability
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_CAP_INVALID	
See Also	CorAcqDeviceGetPrm	

CorAcqDeviceGetPrm

Gets a parameter value from an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceGetPrm**(CORACQDEVICE *acqDevice*, UINT32 *prm*, void **value*);	
Description	Gets a parameter (simple and complex) value from an acquisition device. Make certain that the data storage pointed to by *value* matches the data type of the specified parameter. See the description of each parameter for the required data storage type.	
Input	*acqDevice*	Acquisition resource handle
	prm	Acquisition parameter requested
Output	*value*	Current value of the requested parameter
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_PRM_INVALID	
See Also	CorAcqDeviceSetPrm and CorAcqDeviceSetPrmEx	

CorAcqDeviceSetPrm

Sets a simple parameter value to an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceSetPrm**(CORACQDEVICE *acqDevice*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple acquisition device parameter. A simple parameter is one that fits inside an UINT32. For complex parameters use CorAcqDeviceSetPrmEx. See the description of each parameter for the required data storage type.	
Input	*acqDevice*	Acquisition resource handle
	prm	Acquisition parameter to set
	value	New value of the parameter

Output	None
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_PRM_READ_ONLY
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceGetPrm, CorAcqDeviceSetPrmEx

CorAcqDeviceSetPrmEx

Sets a complex parameter value to an acquisition device.

Prototype	CORSTATUS **CorAcqDeviceSetPrmEx**(CORACQDEVICE *acqDevice*, UINT32 *prm*, const void **value*);	
Description	Sets a complex acquisition device's parameter. A complex parameter is greater in size than an UINT32. If the parameter size is UINT32, use either CorAcqDeviceSetPrm or CorAcqDeviceSetPrmEx. See the description of each parameter for the required data storage type.	
Input	*acqDevice*	Acquisition resource handle
	prm	Acquisition parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_PRM_READ_ONLY	
Note	This function is not valid for read-only handles.	
See Also	CorAcqDeviceGetPrm and CorAcqDeviceSetPrm	

Feature Access Functions

The feature access functions are used to enumerate the list of features supported by the device, get information associated with each of those features and read/write the feature values from/to the device.

CorAcqDeviceGetFeatureCount

Returns the number of features supported by the acquisition device.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureCount**(CORACQDEVICE *acqDevice*, PUINT32 *count*);	
Description	Returns the number of features supported by the acquisition device. Devices do not necessarily support the same feature set. For instance you can use this function to retrieve the number of features and then get information about those features using CorAcqDeviceGetFeatureInfo.	
Input	*acqDevice*	Acquisition resource handle
Output	*count*	Number of features
Return Value	CORSTATUS_OK	

CorAcqDeviceGetFeatureDataByIndex

Returns the value of a complex feature associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureDataByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, CORHANDLE *dataDest*);	
Description	Returns the value of a *complex type* feature associated with a specified index. A complex type feature is a feature that is contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For simple type features you must use the *CorAcqDeviceGetFeatureValueByIndex* instead. The *dataDest* handle must be allocated by the user prior to calling this function. For example if the feature is contained in a *CorBuffer* module, you must allocate the buffer using *CorBufferNew*. Attributes of the buffer such as *width*, *height* and *format* must be obtained through others features provided by your acquisition device. Upon calling this function the buffer's data from the device will be copied to your buffer object.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
Output	*dataDest*	Resource handle to store feature data
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetFeatureCount, CorAcqDeviceGetFeatureValueByIndex, CorAcqDeviceGetFeatureDataByName	

CorAcqDeviceGetFeatureDataByName

Returns the value of a complex feature associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureDataByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, CORHANDLE *dataDest*);

Description	Returns the value of a *complex type* feature associated with a specified name. A complex type feature is a feature that is contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For simple type features you must use the *CorAcqDeviceGetFeatureValueByName* instead. The *dataDest* handle must be allocated by the user prior to calling this function. For example if the feature is contained in a *CorBuffer* module, you must allocate the buffer using *CorBufferNew*. Attributes of the buffer such as *width*, *height* and *format* must be obtained through others features provided by your acquisition device. Upon calling this function the buffer's data from the device will be copied to your buffer object.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
Output	*dataDest*	Resource handle to store feature data
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceSetFeatureValueByName, CorAcqDeviceSetFeatureDataByIndex	

CorAcqDeviceGetFeatureIndexByName

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

Prototype	CORSTATUS **CorAcqDeviceGetFeatureIndexByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, PUINT32 *index*);	
Description	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 those features are accessed by index to increase performance.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
Output	*index*	Index of the feature associated with the specified name
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetFeatureNameByIndex	

CorAcqDeviceGetFeatureInfoByIndex

Returns information on a feature associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureInfoByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, CORFEATURE *feature*);	
Description	Returns information on a feature associated with a specified index. All information about the feature is stored in a *CorFeature* object. The *CorFeature* object contains the attributes of the feature such as name, type, range, etc. See the *CorFeature* module for more details.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
Output	*feature*	Feature resource handle to store feature information
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetFeatureCount, CorAcqDeviceGetFeatureInfoByName, Feature Module	

CorAcqDeviceGetFeatureInfoByName

Returns information on a feature associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureInfoByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, CORFEATURE *feature*);	
Description	Returns information on a feature associated with a specified name. All the information about the feature is stored in a *CorFeature* object. The *CorFeature* object contains the attributes of the feature such as name, type, range, etc. See the *CorFeature* module for more details.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
Output	*feature*	Feature resource handle to store feature information
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetFeatureInfoByIndex, Feature Module	

CorAcqDeviceGetFeatureNameByIndex

Returns the name of a feature associated with a specified index

Prototype	CORSTATUS **CorAcqDeviceGetFeatureNameByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, PSTR *name*, UINT32 *nameSize*);	
Description	Returns the name of a feature associated with a specified index. For instance you can use this function to display the name of all features supported by the device.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
	nameSize	Size (in bytes) of the buffer pointed to by *name*
Output	*name*	Name of the feature associated with the specified index
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetFeatureIndexByName	

CorAcqDeviceGetFeatureValueByIndex

Returns the value of a simple feature associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureValueByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, void **value*, UINT32 *valueSize*);	
Description	Returns the value of a *simple type* feature associated with a specified index. A simple type feature is a feature that is NOT contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For complex type features you must use the *CorAcqDeviceGetFeatureDataByIndex* instead. To determine the size of the buffer pointed to by *value* you must obtain the type of the feature using *CorAcqDeviceGetFeatureInfoByIndex*. Each feature type has its predetermined size.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
	valueSize	Size of the buffer pointed to by *value* (in bytes)

Output	*value*	Address of a buffer to store the feature value
Return Value	CORSTATUS_OK	
Note	Except for unitless features, each feature has its specific native unit, e.g. millisecond, KHz, tenth of degree, etc. This information is obtained through the CORFEATURE_PRM_SI_UNIT and CORFEATURE_PRM_SI_TO_NATIVE_EXP10 parameters.	
See Also	CorAcqDeviceGetFeatureCount ,CorAcqDeviceGetFeatureDataByIndex, CorAcqDeviceGetFeatureValueByName	

CorAcqDeviceGetFeatureValueByName

Returns the value of a simple feature associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceGetFeatureValueByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, void **value*, UINT32 *valueSize*);	
Description	Returns the value of a *simple type* feature associated with a specified name. A simple type feature is a feature that is NOT contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For complex type features you must use the *CorAcqDeviceGetFeatureDataByName* instead. To determine the size of the buffer pointed to by *value* you must obtain the type of the feature using *CorAcqDeviceGetFeatureInfoByName*. Each feature type has its predetermined size.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
	valueSize	Size of the buffer pointed to by *value* (in bytes)
Output	*value*	Address of a buffer to store the feature value
Return Value		
Note	Except for unitless features, each feature has its specific native unit, e.g. millisecond, KHz, tenth of degree, etc. This information is obtained through the CORFEATURE_PRM_SI_UNIT and CORFEATURE_PRM_SI_TO_NATIVE_EXP10 parameters.	
See Also	CorAcqDeviceGetFeatureDataByName, CorAcqDeviceGetFeatureValueByIndex	

CorAcqDeviceIsFeatureAvailable

Returns whether or not a feature is supported by the acquisition device.

Prototype	CORSTATUS **CorAcqDeviceIsFeatureAvailable**(CORACQDEVICE *acqDevice*, PCSTR *name*, PUINT32 *isAvailable*);	
Description	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.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
Output	*isAvailable*	TRUE if the feature is supported by the device. FALSE otherwise
Return Value	CORSTATUS_OK	

CorAcqDeviceLoadFeatures

Loads all device features from a configuration file.

Prototype	CORSTATUS **CorAcqDeviceLoadFeatures**(CORACQDEVICE *acqDevice*, PCSTR filename);
Description	Loads all the features from a configuration file. This function reads the features from a *Sapera LT's CCF File* and writes them to the device. The CCF file is generated by the *Sapera LT CamExpert Program*. For devices that support hardware persistence storage (see your Acquisition Device User's Manual) 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.
Input	*acqDevice* Acquisition resource handle *filename* Name of the configuration file to load features from
Output	None
Return Value	CORSTATUS_OK
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceSaveFeatures

CorAcqDeviceSaveFeatures

Saves all or a subset of features to a configuration file.

Prototype	CORSTATUS **CorAcqDeviceSaveFeatures**(CORACQDEVICE *acqDevice*, PCSTR *filename*);
Description	Saves all or a subset of the device features to a configuration file. This function reads the features from the device and writes them to a *Sapera LT's CCF File*. Not all features are saved. For example read-only features are not saved by default. To know what features are saved by default refer to CORFEATURE_PRM_SAVED_TO_CONFIG_FILE parameter. This parameter also allows you to control whether each individual feature is saved or not. This function is useful for devices that do not support hardware persistence storage (see the Acquisition Device User's Manual) in order to retreive the feature values after a shut down of the device.
Input	*acqDevice* Acquisition resource handle *filename* Name of the configuration file to save features to
Output	None
Return Value	CORSTATUS_OK
See Also	CorAcqDeviceLoadFeatures

CorAcqDeviceSetFeatureDataByIndex

Sets the value of a complex feature associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceSetFeatureDataByIndex**(CORACQDEVICE *acqDevice*,

UINT32 *index*, CORHANDLE *dataSrc*);

Description	Sets the value of a *complex type* feature associated with a specified index. A complex type feature is a feature that is contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For simple type features you must use *CorAcqDeviceSetFeatureValueByIndex* instead.	
	The *dataSrc* handle must be a valid handle containing the feature data to be set.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
	dataSrc	Resource handle that contains feature data
Output	None	
Return Value	CORSTATUS_OK	
Note	This function is not valid for read-only handles.	
See Also	CorAcqDeviceGetFeatureCount, CorAcqDeviceSetFeatureValueByIndex, CorAcqDeviceSetFeatureDataByName	

CorAcqDeviceSetFeatureDataByName

Sets the value of a complex feature associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceSetFeatureDataByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, CORHANDLE *dataSrc*);	
Description	Sets the value of a *complex type* feature associated with a specified name. A complex type feature is a feature that is contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For simple type features you must use *CorAcqDeviceSetFeatureValueByName* instead.	
	The *dataSrc* handle must be a valid handle containing the feature data to be set.	
Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
	dataSrc	Resource handle that contains feature data
Output	None	
Return Value	CORSTATUS_OK	
Note	This function is not valid for read-only handles.	
See Also	CorAcqDeviceSetFeatureValueByName CorAcqDeviceSetFeatureDataByIndex	

CorAcqDeviceSetFeatureValueByIndex

Sets the value of a simple feature associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceSetFeatureValueByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, const void **value*, UINT32 *valueSize*);
Description	Sets the value of a *simple type* feature associated with a specified index. A simple type feature is a feature that is NOT contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For complex type features you must use *CorAcqDeviceSetFeatureDataByIndex* instead.

To determine the size of the buffer pointed to by *value* you must obtain the feature type using *CorAcqDeviceGetFeatureInfoByIndex*. Each feature type has its predetermined size.

Input	*acqDevice*	Acquisition resource handle
	index	Index of the feature. Ranges from 0 to CorAcqDeviceGetFeatureCount.
	value	Address of a buffer that contains the feature value
	valueSize	Size of the buffer pointed to by *value* (in bytes)

Output None

Return Value CORSTATUS_OK

Notes This function is not valid for read-only handles.

Except for unitless features, each feature has its specific native unit, e.g. millisecond, KHz, tenth of degree, etc. This information is obtained through the CORFEATURE_PRM_SI_UNIT and CORFEATURE_PRM_SI_TO_NATIVE_EXP10 parameters.

See Also CorAcqDeviceGetFeatureCount
CorAcqDeviceSetFeatureValueByName
CorAcqDeviceSetFeatureDataByIndex

CorAcqDeviceSetFeatureValueByName

Sets the value of a simple feature associated with a specified name.

Prototype CORSTATUS **CorAcqDeviceSetFeatureValueByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, const void **value*, UINT32 *valueSize*);

Description Sets the value of a *simple type* feature associated with a specified name. A simple type feature is a feature that is NOT contained in a handle. See CORFEATURE_PRM_TYPE for a list of simple and complex types. For complex type features you must use *CorAcqDeviceSetFeatureDataByName* instead.

To determine the size of the buffer pointed to by *value* you must obtain the feature type using *CorAcqDeviceGetFeatureInfoByName*. Each feature type has its predetermined size.

Input	*acqDevice*	Acquisition resource handle
	name	Feature name. See device User's Manual for the list of supported features.
	value	Address of a buffer that contains the feature value
	valueSize	Size of the buffer pointed to by *value* (in bytes)

Output None

Return Value CORSTATUS_OK

Note This function is not valid for read-only handles.

Except for unitless features, each feature has its specific native unit, e.g. millisecond, KHz, tenth of degree, etc. This information is obtained through the CORFEATURE_PRM_SI_UNIT and CORFEATURE_PRM_SI_TO_NATIVE_EXP10 parameters.

See Also CorAcqDeviceSetFeatureValueByIndex, CorAcqDeviceSetFeatureDataByName

CorAcqDeviceUpdateFeaturesFromDevice

Gets all the features from the acquisition device at once.

Prototype	CORSTATUS **CorAcqDeviceUpdateFeaturesFromDevice**(CORACQDEVICE *acqDevice*);
Description	
Input	*acqDevice* Acquisition resource handle
Output	None
Return Value	Gets all the features from the acquisition device at once. This function must be used when the *feature update mode* is set to manual (See CORACQDEVICE_PRM_FEATURE_UPDATE_MODE). In this mode writing individual features is done to an internal cache. Calling this function resets the internal cache to the values currently present in the device. This is useful when a certain number of features have been set to the internal cache but you want to undo those settings.
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceUpdateFeaturesToDevice, CORACQDEVICE_PRM_UPDATE_FEATURE_MODE

CorAcqDeviceUpdateFeaturesToDevice

Sets all the features to the acquisition device at once.

Prototype	CORSTATUS **CorAcqDeviceUpdateFeaturesToDevice**(CORACQDEVICE *acqDevice*);
Description	Sets all the features to the acquisition device at once. This function must be used when the *feature update mode* is set to manual (See CORACQDEVICE_PRM_FEATURE_UPDATE_MODE). In this mode writing individual features is done to an internal cache. After all the required features have been set, call this function to write them all to the device.
Input	*acqDevice* Acquisition resource handle
Output	None
Return Value	CORSTATUS_OK
Note	This function is not valid for read-only handles.
See Also	CorAcqDeviceUpdateFeaturesFromDevice, CORACQDEVICE_PRM_UPDATE_FEATURE_MODE

Event Management Functions

The event management functions are used to enumerate the list of events supported by the device and register user callback functions on those events.

CorAcqDeviceGetEventCount

Returns the number of events supported by the acquisition device.

Prototype	CORSTATUS **CorAcqDeviceGetEventCount**(CORACQDEVICE *acqDevice*, PUINT32 *count*);	
Description	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 *CorAcqDeviceGetEventNameByIndex*.	
Input	*acqDevice*	Acquisition resource handle
Output	*count*	Number of events
Return Value	CORSTATUS_OK	

CorAcqDeviceGetEventIndexByName

Returns the index of an event associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceGetEventIndexByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, PUINT32 *index*);	
Description	Returns the index of an event associated with a specified name.	
Input	*acqDevice*	Acquisition resource handle
	name	Event name. See device User's Manual for the list of supported events.
Output	*index*	Index of the event associated with the specified name
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceGetEventIndexByName	

CorAcqDeviceGetEventNameByIndex

Returns the name of an event associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceGetEventNameByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, PSTR *name*, UINT32 *nameSize*);	
Description	Returns the name of an event associated with a specified index. The typical usage is converting an event index (retrieved from your callback information) to the corresponding name. You can then sort the events by name in order to call the appropriate event handling code.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the event. Ranges from 0 to CorAcqDeviceGetEventCount.
	nameSize	Size (in bytes) of the buffer pointed to by *name*
Output	*name*	Name of the event associated with the specified index

Return Value	CORSTATUS_OK
See Also	CorAcqDeviceGetEventIndexByName

CorAcqDeviceIsEventAvailable

Returns whether or not an event is supported by the acquisition device.

Prototype	CORSTATUS **CorAcqDeviceIsEventAvailable**(CORACQDEVICE *acqDevice*, PCSTR *name*, PUINT32 *isAvailable*);	
Description	Returns 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.	
Input	*acqDevice*	Acquisition resource handle
	name	Event name. See device User's Manual for the list of supported events.
Output	*isAvailable*	TRUE if the event is supported by the device. FALSE otherwise
Return Value	CORSTATUS_OK	

CorAcqDeviceIsCallbackRegisteredByIndex

Returns whether or not a callback function was registered on the event associated with a specified index.

Prototype	CORSTATUS **CorAcqDeviceIsCallbackRegisteredByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, PUINT32 *isRegistered*);	
Description	Returns whether or not a callback function was registered on the event associated with a specified index. For example use this function in a loop to determine if the callback function associated with the current index has to be unregistered.	
Input	*acqDevice*	Acquisition resource handle
	index	Index of the event. Ranges from 0 to CorAcqDeviceGetEventCount.
Output	*isRegistered*	TRUE if a callback function was registered on this event. FALSE otherwise
Return Value	CORSTATUS_OK	
See Also	CorAcqDeviceIsCallbackRegisteredByNameCorAcqDeviceIsCallbackRegisteredByName	

CorAcqDeviceIsCallbackRegisteredByName

Returns whether or not a callback function was registered on the event associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceIsCallbackRegisteredByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, PUINT32 *isRegistered*);	
Description	Returns whether or not a callback function was registered on the event associated with a specified name.	
Input	*acqDevice*	Acquisition resource handle
	name	Event name. See device User's Manual for the list of supported events.
Output	*isRegistered*	TRUE if a callback function was registered on this event. FALSE otherwise
Return Value	CORSTATUS_OK	

See Also CorAcqDeviceIsCallbackRegisteredByIndex

CorAcqDeviceRegisterCallbackByIndex

Registers a callback function on the event associated with a specified index.

Prototype CORSTATUS **CorAcqDeviceRegisterCallbackByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*, PCOREVENTINFOCALLBACK *callback*, void **context*);

Description Registers an event by associating a callback function to the specified index. When the event occurs in the acquisition device the specified callback function is called. The callback function provides information on the corresponding event (in the *COREVENTINFO* handle). Refer to the *EventInfo* module for more detail on the available information. The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

Input

acqDevice	Acquisition resource handle
index	Index of the event. Ranges from 0 to CorAcqDeviceGetEventCount.
callback	Address of a user callback function of the following form:

```
CORSTATUS CCONV MyCallback(void *context, COREVENTINFO
hEventInfo)
{
}
```

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

Output None

Return Value CORSTATUS_OK

Note This function is not valid for read-only handles.

See Also CorAcqDeviceRegisterCallbackByName, EventInfo Module

CorAcqDeviceRegisterCallbackByName

Registers a callback function on the event associated with a specified name.

Prototype CORSTATUS **CorAcqDeviceRegisterCallbackByName**(CORACQDEVICE *acqDevice*, PCSTR *name*, PCOREVENTINFOCALLBACK *callback*, void **context*);

Description Registers an event by associating a callback function to the specified name. When the event occurs in the acquisition device the specified callback function is called. The callback function provides information on the corresponding event (in the *COREVENTINFO* handle). Refer to the *EventInfo* module for more detail on the available information. The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

Input

acqDevice	Acquisition resource handle
name	Event name. See device User's Manual for the list of supported events.
callback	Address of a user callback function of the following form:

```
CORSTATUS CCONV MyCallback(void *context, COREVENTINFO
hEventInfo)
{
}
```

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

Output None

Return Value CORSTATUS_OK

Note This function is not valid for read-only handles.

See Also CorAcqDeviceRegisterCallbackByIndex, EventInfo Module

Example

```
CORSTATUS CCONV MyCallback(void *context, COREVENTINFO hEventInfo)
{
   // Retrieve "AcqDevice" handle through context pointer
   // Context could be anything else
   CORACQDEVICE hAcqDevice = (CORACQDEVICE) context;

   // Access information using CorEventInfo...functions
   // ...
}

main()
{
   // ...
   CorAcqDeviceRegisterCallbackByName(hAcqDevice,
   "FeatureValueChanged", MyCallback, hAcqDevice);
   // ...
   CorAcqDeviceUnregisterCallbackByName(hAcqDevice,
   "FeatureValueChanged");
   // ...
}
```

CorAcqDeviceUnregisterCallbackByIndex

Unregisters a callback function on the event associated with a specified index.

Prototype CORSTATUS **CorAcqDeviceUnregisterCallbackByIndex**(CORACQDEVICE *acqDevice*, UINT32 *index*);

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

Input *acqDevice* Acquisition resource handle

index Index of the event. Ranges from 0 to CorAcqDeviceGetEventCount.

Output None

Return Value CORSTATUS_OK

Note This function is not valid for read-only handles.

See Also CorAcqDeviceIsCallbackRegisteredByIndex, CorAcqDeviceIsCallbackRegisteredByName, CorAcqDeviceUnregisterCallbackByName

Example

```
// Unregisters all the callback functions
//
UINT32 eventCount, eventIndex;
CorAcqDeviceGetEventCount(hAcqDevice, &eventCount);
for (eventIndex = 0; eventIndex < eventCount; eventIndex++)
{
   BOOL isRegistered;
   CorAcqDeviceIsCallbackRegistered(hAcqDevice, eventIndex,
   &isRegistered);
   if (isRegistered)
   {
```

```
            CorAcqDeviceUnregisterCallbackByIndex(hAcqDevice,
            eventIndex);
        }
    }
```

CorAcqDeviceUnregisterCallbackByName

Unregisters a callback function on the event associated with a specified name.

Prototype	CORSTATUS **CorAcqDeviceUnregisterCallbackByName**(CORACQDEVICE *acqDevice*, PCSTR *name*);	
Description	Unregisters a callback function on the event associated with a specified name. Use this function to unregister a callback function for which you know the name.	
Input	*acqDevice*	Acquisition resource handle
	name	Event name. See device User's Manual for the list of supported events.
Output	None	
Return Value	CORSTATUS_OK	
Note	This function is not valid for read-only handles.	
See Also	CorAcqDeviceUnregisterCallbackByIndex	
Note	This function is not valid for read-only handles.	
See Also		

Feature Module

The *Feature* module is used to retrieve the feature information from the *AcqDevice* module. Each feature supported by the *AcqDevice* module provides a set of capabilities such as name, type, access mode, etc. that can be obtained through the feature module. The feature module is used by *CorAcqDeviceGetFeatureInfo…* functions.

Feature Parameters

ID	Parameters	Type	Access
0x00	CORFEATURE_PRM_NAME	64 characters String	R only
0x01	CORFEATURE_PRM_TYPE	Enumeration	R only
0x02	CORFEATURE_PRM_STANDARD	Boolean	R only
0x03	CORFEATURE_PRM_ACCESS_MODE	Enumeration	R only
0x04	CORFEATURE_PRM_POLLING_TIME	32-bit Integer	R only
0x05	CORFEATURE_PRM_TOOL_TIP	256 characters String	R only
0x06	CORFEATURE_PRM_DISPLAY_NAME	64 characters String	R only

0x07	CORFEATURE_PRM_REPRESENTATION	Enumeration	R only
0x08	CORFEATURE_PRM_SIGN	Enumeration	R only
0x09	CORFEATURE_PRM_SI_UNIT	32 characters String	R only
0x0A	CORFEATURE_PRM_CATEGORY	64 characters String	R only
0x0B	CORFEATURE_PRM_WRITE_MODE	32-bit Integer	R only
0x0C	CORFEATURE_PRM_SAVED_TO_CONFIG_FILE	Boolean	RW
0x0D	CORFEATURE_PRM_SI_TO_NATIVE_EXP10	32-bit Integer	R only
0x0E	CORFEATURE_PRM_VISIBILITY	Enumeration	R only

CORFEATURE_PRM_ACCESS_MODE

Description Determines the type of access for a feature. See the value descriptions below.

Type UINT32

Values

CORFEATURE_VAL_ACCESS_MODE_UNDEFINED
Undefined access mode

CORFEATURE_VAL_ACCESS_MODE_RW
The feature can be read and written. Most of the features are of this type.

CORFEATURE_VAL_ACCESS_MODE_RO
The feature can only be read. Certain type of features such as “Sensor Temperature” and “Sensor Resolution” cannot be written.

CORFEATURE_VAL_ACCESS_MODE_WO
The feature can only be written. Some features represent commands (or actions) such as “starting a calibration algorithm” for example. This kind of feature cannot be read back.

CORFEATURE_VAL_ACCESS_MODE_NP
The feature is not present. The feature is visible in the interface but is not implemented for this device.

CORFEATURE_VAL_ACCESS_MODE_NE
The feature is present but not enabled. Often used when a feature depends on another feature’s value.

CORFEATURE_PRM_CATEGORY

Description Represents the category of features the current feature belongs to. All the features are divided into categories to simplify the presentation of features coming from a large feature set.

Type 64 charaters string

Values String representing the name of the category

Note The categories are useful to present a categorized list of features in a graphical user interface.

CORFEATURE_PRM_DISPLAY_NAME

Description Represents the name of the feature in a more descriptive way than CORFEATURE_PRM_NAME. This name can be used for listing features in a graphical user interface.

Type 64 charaters string

Values String representing the display name of the feature

Note This name must not be used as an index to the *AcqDevice* module functions. Use CORFEATURE_PRM_NAME instead.

CORFEATURE_PRM_NAME

Description Represents the name of the feature. The name of a feature can be used as an index to the feature set in the *AcqDevice* module.

Type 64 charaters string

Values String representing the name of the feature

Note This string should not be used for display in a graphical user interface. Instead the CORFEATURE_PRM_DISPLAY_NAME provides a more descriptive name.

CORFEATURE_PRM_POLLING_TIME

Description Interval of time between two consecutive feature updates. Some read-only features such as "Sensor Temperature" for instance must be read from the device at a certain frequency in order to refresh to feature module. You can modify this value to either increase or decrease the reshresh rate.

Type UINT32

Values Positive integer

CORFEATURE_PRM_REPRESENTATION

Description Determines the mathematical representation of a integer or float feature. See possible options below.

Type UINT32

Values

CORFEATURE_VAL_REPRESENTATION_UNDEFINED
Undefined representation

CORFEATURE_VAL_REPRESENTATION_LINEAR
The feature follows a linear scale

CORFEATURE_VAL_REPRESENTATION_LOGARITHMIC
The feature follows a logarithmic scale

CORFEATURE_VAL_REPRESENTATION_BOOLEAN
The feature is a boolean (can have two values: zero or non-zero)

CORFEATURE_PRM_SAVED_TO_CONFIG_FILE

Description Specifies whether or not a feature is saved to the configuration file when calling *CorAcqDeviceSaveFeatures*. All the 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 behavoir using this parameter. For example a read-only feature such as "Sensor Temperature" is not saved by default. You set this parameter to TRUE to force the feature to be written to the configuration file.

Type Boolean

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

Note 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.

CORFEATURE_PRM_SIGN

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

Type UINT32

Values

CORFEATURE_VAL_SIGN_UNDEFINED
Sign is undefined

CORFEATURE_VAL_SIGN_SIGNED
The feature is a signed integer of float

CORFEATURE_VAL_SIGN_UNSIGNED
The feature is an unsigned integer of float

CORFEATURE_PRM_SI_TO_NATIVE_EXP10

Description Used to convert the value of a feature from SI unit (international system) to native unit (the unit 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.

Type 32-bit integer

Values The value is the exponent of a base 10. It can be negative or positive.

Example 1 You want to set the camera integration time to a known value in seconds. The "IntegrationDuration" feature is represented in microseconds. Therefore the current parameter value is 6. For instance if the desired integration time is 0.5 second you can compute the value to pass to *CorAcqDeviceSetFeatureValue*... as follows:

$$V_{NATIVE} = 0.5 * 10^{6} = 500000$$

Example 2 You want to monitor the temperature of the camera sensor. The "SensorTemperature" feature is reported in tenths of celcius degree. Therefore the current parameter value is 1. For instance if the feature value returned by *CorAcqDeviceGetFeatureValue…* is 205 then you can compute the temperature in celcius as follows:

$$V_{SI} = \frac{V_{NATIVE}}{10^{E}} = \frac{205}{10^{1}} = 20.5$$

Use the *CORFEATURE_SI_UNIT* parameter to retrieve the SI unit corresponding to the feature to monitor.

CORFEATURE_PRM_SI_UNIT

Description Describes the physical unit 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 unit used by the feature (the native unit) is NOT the SI unit directly but a multiple of it, e.g. the exposure time may be represented in microseconds instead of seconds. To convert the feature value to the SI unit you must use the *CORFEATURE_PRM_SI_TO_NATIVE_EXP10* conversion factor.

Type 32 charaters string

Values String representing the unit of the feature in the international system (SI)

Note Many features are unitless and therefore don't provide this information. In such a case an empty string is returned.

CORFEATURE_PRM_STANDARD

Description Determines 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.

Type Boolean

Values TRUE if the feature is standard, FALSE otherwise.

CORFEATURE_PRM_TOOL_TIP

Description Small text representing the explanation of the feature.

Type 64 charaters string

Values String representing the tool tip

Note This parameter can be used to implement tool tips in a graphical user interface.

CORFEATURE_PRM_TYPE

Description

Type UINT32

Values

CORFEATURE_VAL_TYPE_UNDEFINED Undefined type	Simple

CORFEATURE_VAL_TYPE_INT32 32-bit Integer	Simple
CORFEATURE_VAL_TYPE_INT64 64-bit Integer	Simple
CORFEATURE_VAL_TYPE_FLOAT 32-bit Floating-Point	Simple
CORFEATURE_VAL_TYPE_DOUBLE 64-bit Floating-Point	Simple
CORFEATURE_VAL_TYPE_BOOL Boolean	Simple
CORFEATURE_VAL_TYPE_ENUM Enumeration	Simple
CORFEATURE_VAL_TYPE_STRING ASCII character string	Simple
CORFEATURE_VAL_TYPE_BUFFER Buffer handle	Complex
CORFEATURE_VAL_TYPE_LUT LUT handle	Complex

Note If the type is *simple* the feature must be read/written using the *CorAcqDeviceGetFeatureValue…/ CorAcqDeviceSetFeatureValue…* functions. If the type is *complex* the feature must be read/written using the *CorAcqDeviceGetFeatureData…/ CorAcqDeviceSetFeatureData…* functions.

CORFEATURE_PRM_VISIBILITY

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

Type UINT32

Values

CORFEATURE_VAL_VISIBILITY_UNDEFINED
Undefined visibility level

CORFEATURE_VAL_VISIBILITY_BEGINNER
Specifies that the feature should be made visible to any user.

CORFEATURE_VAL VISIBILITY EXPERT
Specifies that the feature should be made visible to users with a certain level of expertise.

CORFEATURE_VAL VISIBILITY GURU
Specifies that the feature should be made visible to users with a high level of expertise.

CORFEATURE_VAL VISIBILITY INVISIBLE
Specifies that the feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features.

CORFEATURE_PRM_WRITE_MODE

Description Determines whether or not a feature can be modified when the transfer module is connected and/or acquiring. Some features like the buffer dimensions for example cannot be changed while data is being transfered to the buffer.

Type UINT32

Values

CORFEATURE_VAL_WRITE_MODE_UNDEFINED
Undefined write mode

CORFEATURE_VAL_WRITE_MODE_ALWAYS
The feature can always be written.

CORFEATURE_VAL WRITE_MODE_NOT_ACQUIRING
The feature can only be written when the transfer module is not acquiring. If the transfer is currently acquiring you must stop the acquisition using *CorXferStop / CorXferWait* before modifying the feature value.

CORFEATURE_VAL WRITE_MODE_NOT_CONNECTED
The feature can only be written when the transfer module is not connected. If the transfer is currently connected you must disconnect it using *CorXferDisconnect* before modifying the feature value.

Note Use this information to prevent an application from changing certain features when the transfer module is connected and/or acquiring.

Feature Functions

Function	Description
Creation/Destruction	
CorFeatureNew	Creates an empty feature object
CorFeatureFree	Destroys a feature object
General Parameters	
CorFeatureGetPrm	Gets a parameter value from a feature object
CorFeatureSetPrm	Sets a simple parameter value to a feature object
CorFeatureSetPrmEx	Sets a complex parameter value to a feature object
Integer/Float-specific Parameters	
CorFeatureGetMin	Returns the minimum acceptable value for a feature
CorFeatureGetMax	Returns the maximum acceptable value for a feature
CorFeatureGetInc	Returns the minimum acceptable increment for a feature
Enumeration-specific Parameters	
CorFeatureGetEnumCount	Returns the number of items in an enumeration
CorFeatureGetEnumString	Returns the enumeration string corresponding to a specified index
CorFeatureGetEnumValue	Returns the enumeration value corresponding to a specified index
CorFeatureIsEnumEnabled	Returns whether or not the enumeration item corresponding to a specified index is enabled
CorFeatureGetEnumStringFromValue	Returns the enumeration string corresponding to a specified enumeration value
CorFeatureGetEnumValueFromString	Returns the enumeration value corresponding to a specified enumeration string

Creation/Destruction

CorFeatureFree

Destroys a feature object.

Prototype CORSTATUS **CorFeatureFree**(CORFEATURE *feature*);

Description Destroys a feature object previously created using *CorFeatureNew*.

Input *feature* Feature handle

Output None

Return Value CORSTATUS_OK

See Also CorFeatureNew

CorFeatureNew

Creates an empty feature object.

Prototype	CORSTATUS **CorFeatureNew**(CORSERVER *server*, PCORFEATURE *feature*);	
Description	Creates an empty feature object. Upon creation the feature object is reset with null values. To fill-in a feature object call the *CorAcqDeviceGetFeatureInfo...* functions.	
Input	*server*	Handle to a server where to create the feature object. Must be the same as that of the *CorAcqDevice* object.
Output	feature	Feature handle
Return Value	CORSTATUS_OK	
See Also	CorFeatureFree, CorAcqDeviceGetFeatureInfoByIndex, CorAcqDeviceGetFeatureInfoByName	

General Parameters

The general parameters are those that apply to all feature types (they are not specific to any particular type). For example the *name* of a feature is a general parameter.

CorFeatureGetPrm

Gets a parameter value from a feature object.

Prototype	CORSTATUS **CorFeatureGetPrm**(CORFEATURE *feature*, UINT32 *prm*, void **value*);	
Description	Gets a parameter (simple and complex) value from a feature object. Make certain that the data storage pointed to by *value* matches the data type of the specified parameter. See the description of each parameter to know what kind of data storage it requires.	
Input	*feature*	Feature handle
	prm	Feature parameter requested
Output	*value*	Current value of the requested parameter
Return Value	CORSTATUS_OK	
See Also	CorFeatureSetPrm, CorFeatureSetPrmEx, Feature Parameters	

CorFeatureSetPrm

Sets a simple parameter value to a feature object.

Prototype	CORSTATUS **CorFeatureSetPrm**(CORFEATURE *feature*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple feature parameter. A simple parameter is one that fits inside an UINT32. If the parameter is complex, use CorFeatureSetPrmEx. See the description of each parameter to know what kind of data storage it requires.	
Input	*feature*	Feature handle
	prm	Feature parameter to set
	value	New value of the parameter

Output	None
Return Value	CORSTATUS_OK
See Also	CorFeatureGetPrm, CorFeatureSetPrmEx, Feature Parameters

CorFeatureSetPrmEx

Sets a complex parameter value to a feature object.

Prototype	CORSTATUS **CorFeatureSetPrmEx**(CORFEATURE *feature*, UINT32 *prm*, const void **value*);	
Description	Sets a complex feature parameter. A complex parameter is greater in size than an UINT32. If the parameter size is UINT32, use either CorFeatureSetPrm or CorFeatureSetPrmEx. See the description of each parameter to know what kind of data storage it requires.	
Input	*feature*	Feature handle
	prm	Feature parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetPrm, CorFeatureSetPrm, Feature Parameters	

Integer/Float-specific Parameters

The following paramters are specific to integer and float feature types.

CorFeatureGetInc

Returns the minimum acceptable increment for a feature.

Prototype	CORSTATUS **CorFeatureGetInc**(CORFEATURE *feature*, void **incVal*, UINT32 *valSize*);	
Description	Returns 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.	
Input	*feature*	Feature handle
	valSize	Size of the *incVal* buffer
Output	*incVal*	Address of the buffer to store the increment value
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetMin, CorFeatureGetMax	

CorFeatureGetMax

Returns the maximum acceptable value for a feature.

Prototype	CORSTATUS **CorFeatureGetMax**(CORFEATURE *feature*, void **maxVal*, UINT32 *valSize*);	
Description	Returns the maximum acceptable value for a feature. For integer and float types *maxVal* must point to a variable of the same type as the feature. For a string type the maximum length of the string (excluding the trailing null character) is returned and *maxVal* must point to a UINT32 variable.	
Input	*feature*	Feature handle
	valSize	Size of the *maxVal* buffer
Output	*maxVal*	Address of the buffer to store the maximum value
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetMin, CorFeatureGetInc	

CorFeatureGetMin

Returns the minimum acceptable value for a feature.

Prototype	CORSTATUS **CorFeatureGetMin**(CORFEATURE *feature*, void **minVal*, UINT32 *valSize*);	
Description	Returns the minimum acceptable value for a feature. For integer and float types *minVal* must point to a variable of the same type as the feature. For a string type the minimum length of the string (excluding the trailing null character) is returned and *minVal* must point to a UINT32 variable.	
Input	*feature*	Feature handle
	valSize	Size of the *minVal* buffer
Output	*minVal*	Address of the buffer to store the minimum value
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetMax, CorFeatureGetIncc	

Enumeration-specific Parameters

The following paramters are specific to the enumeration feature type.

CorFeatureGetEnumCount

Returns the number of items in an enumeration.

Prototype	CORSTATUS **CorFeatureGetEnumCount**(CORFEATURE *feature*, PUINT32 *count*);	
Description	Returns the number of items in an enumeration. Use this function along with *CorFeatureGetEnumString* and *CorFeatureGetEnumValue* to enumerate all the items contained within an enumeration feature.	
Input	*feature*	Feature handle

Output	*count*	Number of items in the enumeration
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetEnumString, CorFeatureGetEnumValue	

CorFeatureGetEnumString

Returns the enumeration string corresponding to a specified index.

Prototype	CORSTATUS **CorFeatureGetEnumString**(CORFEATURE *feature*, UINT32 *index*, PSTR *enumString*, UINT32 *stringSize*);	
Description	Returns the enumeration string corresponding to a specified index. Use this function along with *CorFeatureGetEnumCount* and *CorFeatureGetEnumValue* to enumerate all the items contained within an enumeration feature.	
Input	*feature*	Feature handle
	index	Index of the enumeration item. Ranges from 0 to *CorFeatureGetEnumCount* - 1
	stringSize	Size of the *enumString* buffer (in bytes)
Output	*enumString*	String associated with the item specified by *index*
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetEnumCount	

CorFeatureGetEnumStringFromValue

Returns the enumeration string corresponding to a specified enumeration value.

Prototype	CORSTATUS **CorFeatureGetEnumStringFromValue**(CORFEATURE *feature*, UINT32 *value*, PSTR *enumString*, UINT32 *stringSize*);	
Description	Returns the enumeration string corresponding to a specified enumeration value. For example you may use this function to retrieve the string corresponding to an enumeration value returned by the *CorAcqDeviceGetFeatureValue...* functions.	
Input	*feature*	Feature handle
	value	Value to look for in the enumeration items
	stringSize	Size of the *enumString* buffer (in bytes)
Output	*enumString*	String associated with the specified value
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetEnumValueFromString	

CorFeatureGetEnumValue

Returns the enumeration value corresponding to a specified index.

Prototype	CORSTATUS **CorFeatureGetEnumValue**(CORFEATURE *feature*, UINT32 *index*, PUINT32 *value*);	
Description	Returns the enumeration value corresponding to a specified index. Use this function along with *CorFeatureGetEnumCount* and *CorFeatureGetEnumString* to enumerate all the items contained within an enumeration feature.	
Input	*feature*	Feature handle
	index	Index of the enumeration item. Ranges from 0 to *CorFeatureGetEnumCount* - 1
Output	*value*	Value associated with the item specified by *index*
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetEnumCount	

CorFeatureGetEnumValueFromString

Returns the enumeration value corresponding to a specified enumeration string.

Prototype	CORSTATUS **CorFeatureGetEnumValueFromString**(CORFEATURE *feature*, PCSTR *enumString*, PUINT32 *value*);	
Description	Returns the enumeration value corresponding to a specified enumeration string. For example you may use this function to retrieve the value corresponding to a known enumeration string and then set this value to the device using the *CorAcqDeviceSetFeatureValue...* functions.	
Input	*feature*	Feature handle
	enumString	String to look for in the enumeration items
Output	*value*	Value associated with the specified string
Return Value	CORSTATUS_OK	
See Also	CorFeatureGetEnumStringFromValue	

CorFeatureIsEnumEnabled

Returns whether or not the enumeration item corresponding to a specified index is enabled.

Prototype	CORSTATUS **CorFeatureIsEnumEnabled**(CORFEATURE *feature*, UINT32 *index*, PUINT32 *enabled*);	
Description	Returns whether or not the enumeration item 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 determine the enable state of an item at a given time.	
Input	*feature*	Feature handle
	index	Index of the enumeration item. Ranges from 0 to *CorFeatureGetEnumCount* - 1
Output	*enabled*	TRUE if the item is enabled, FALSE otherwise.
Return Value	CORSTATUS_OK	

EventInfo Module

The *EventInfo* module is used to retrieve information from the events sent by the *AcqDevice* or the *Manager* module. Each event supported by these modules provides a set of parameters that can be accessed individually through the *EventInfo* module.

EventInfo Parameters

ID	Parameters	Attribute
0x00	COREVENTINFO_PRM_EVENT_COUNT	Integer 32-bit
0x01	COREVENTINFO_PRM_EVENT_INDEX	Integer 32-bit
0x02	COREVENTINFO_PRM_EVENT_TYPE	Integer 32-bit
0x03	COREVENTINFO_PRM_HOST_TIME_STAMP	Integer 64-bit
0x04	COREVENTINFO_PRM_AUX_TIME_STAMP	Integer 64-bit
0x05	COREVENTINFO_PRM_GENERIC_0	Integer 32-bit
0x06	COREVENTINFO_PRM_GENERIC_1	Integer 32-bit
0x07	COREVENTINFO_PRM_GENERIC_2	Integer 32-bit
0x08	COREVENTINFO_PRM_GENERIC_3	Integer 32-bit
0x09	COREVENTINFO_PRM_CUSTOM_DATA	void pointer
0x0a	COREVENTINFO_PRM_CUSTOM_SIZE	Integer 32-bit

ID	Parameter Aliases	Alias of
0x05	COREVENTINFO_PRM_FEATURE_INDEX	COREVENTINFO_PRM_GENERIC_0
0x05	COREVENTINFO_PRM_SERVER_INDEX	COREVENTINFO_PRM_GENERIC_0

COREVENTINFO_PRM_AUX_TIME_STAMP

Description Time stamp corresponding to the moment when the event occurred on the device. Note, not all devices support this time stamp.

Type UINT64

Values This value is specific to the device. See the device driver User's Manual for more information on the availability of this value and the associated unit.

See Also COREVENTINFO_PRM_HOST_TIME_STAMP

COREVENTINFO_PRM_CUSTOM_DATA

Description Address of a buffer containing data associated to a custom event. This parameter is supported only in special versions of certain device drivers.

Type void pointer

Note See the device driver User's Manual for a description of the supported custom events.

COREVENTINFO_PRM_CUSTOM_SIZE

Description Size of the custom data pointed to by COREVENTINFO_PRM_CUSTOM_DATA. This parameter is supported only in special versions of certain device drivers.

Type UINT32

Note See the device driver User's Manual for a description of the supported custom events.

COREVENTINFO_PRM_EVENT_COUNT

Description Number of events that have occurred since the callback function was registered.

Type UINT32

Values Positive integer starting at 1.

Note There is one separate count per module. For example the *AcqDevice* module event counter is independent of that of the *Manager* module.

COREVENTINFO_PRM_EVENT_INDEX

Description Index of the event contained in the EventInfo object. This parameter is used by the events sent by the *AcqDevice* module only.

Type UINT32

Values Ranges from 0 to *CorAcqDeviceGetEventCount* – 1

See Also CorAcqDeviceRegisterCallbackByName, CorAcqDeviceRegisterCallbackByIndex

COREVENTINFO_PRM_EVENT_TYPE

Description Type of event contained in the EventInfo object. This parameter is used by the events sent by the *Manager* module only.

Type UINT32

Values See the *CorManRegisterCallbackEx* function for a list of available events.

COREVENTINFO_PRM_FEATURE_INDEX

Description This parameter is an alias to the COREVENTINFO_PRM_GENERIC_0 parameter.

For example it is used by the "*FeatureInfoChanged*" and "*FeatureValueChanged*" events of the *AcqDevice* module. In this case it represents the index of the feature whose attributes or value have changed.

Type UINT32

Values Ranges from 0 to *CorAcqDeviceGetFeatureCount* – 1.

COREVENTINFO_PRM_GENERIC_0

Description	First generic parameter shared by the different events. You should use aliases instead when they are available. Otherwise refer to the device driver User's Manual for a list of events using generic parameters.
Type	UINT32
Values	Specific to the event
Note	For example COREVENTINFO_PRM_FEATURE_INDEX is an alias of this parameter.

COREVENTINFO_PRM_GENERIC_1

Description	Second generic parameter shared by the different events. You should use aliases instead when they are available. Otherwise refer to the device driver User's Manual for a list of events using generic paramters.
Type	UINT32
Values	Specific to the event

COREVENTINFO_PRM_GENERIC_2

Description	Third generic parameter shared by the different events. You should use aliases instead when they are available. Otherwise refer to the device driver User's Manual for a list of events using generic paramters.
Type	UINT32
Values	Specific to the event

COREVENTINFO_PRM_GENERIC_3

Description	Fourth generic parameter shared by the different events. You should use aliases instead when they are available. Otherwise refer to the device driver User's Manual for a list of events using generic paramters.
Type	UINT32
Values	Specific to the event

COREVENTINFO_PRM_HOST_TIME_STAMP

Description	Time stamp corresponding to the moment when the event occurred on the host.
Type	UINT64
Values	Under Windows, the value corresponding to the high-resolution performance counter is directly returned
Note	For more detail on how to convert this value to time units, refer to *QueryPerformanceCounter* in the Windows API documenation.

COREVENTINFO_PRM_SERVER_INDEX

Description	This parameter is an alias to the COREVENTINFO_PRM_GENERIC_0 parameter. It is used by the following events of the *Manager* module: CORMAN_VAL_EVENT_TYPE_SERVER_NEW CORMAN_VAL_EVENT_TYPE_SERVER_NOT_ACCESSIBLE CORMAN_VAL_EVENT_TYPE_SERVER_ACCESSIBLE It represents the index of the server that has been added or has changed its availability state.
Type	UINT32
Values	Ranges from 0 to *CorManGetServerCount* – 1
See Also	COREVENTINFO_PRM_EVENT_TYPE

EventInfo Functions

Function	Description
CorEventInfoGetPrm	Gets a parameter value from an EventInfo object

CorEventInfoGetPrm

Gets a parameter value from an EventInfo object

Prototype	CORSTATUS **CorEventInfoGetPrm**(COREVENTINFO *hEventInfo*, UINT32 *prm*, void* *pValue*);	
Description	Gets a parameter (simple and complex) value from an EventInfo object. Make certain that the data storage pointed to by *value* matches the data type of the specified parameter. See the description of each parameter to know what kind of data storage it requires.	
Input	*hEventInfo*	EventInfo handle
	prm	EventInfo parameter requested
Output	*value*	Current value of the requested parameter
Return Value	CORSTATUS_OK	
See Also	EventInfo Parameters	

Sapera Basic Modules API

Overview

The Basic Modules constitutes the core of the Sapera programming interface. This module provides everything to acquire, display, and access images. This section covers the following topics:

- **Buffer Module**
  Functions to create, read, and write memory buffers.
- **Counter Module**
  Functions to count internal or external events.
- **Display Module**
  Functions to control the display device.
- **File Module**
  Functions to create, read, write, load, and save files.
- **Graphic Module**
  Functions to perform graphic operations on buffer resources.
- **I/O Module**
  Functions to control a block of general inputs and outputs.
- **LUT Module**
  Functions to define Lookup Tables that can be used by the Acquisition, Display, and Processing modules.
- **Manager Module**
  Functions that allow an application to manage available static resources.
- **PCI Device Module**
  Functions to manage the configuration space of PCI devices.
- **Transfer Module**
  Functions to move data between various sources and destinations.
- **View Module**
  Functions to control a viewing window within the display device.
- **Error Messages**
  Descriptions of the Sapera error and status messages
- **Macro Definitions**
  Descriptions of the Sapera macros used to ensure code portability
- **Data Definitions**
  Descriptions of the Sapera data types used to ensure code portability

- **File Formats**
  Buffer file formats supported in the Sapera File Module.

Sapera Function General Return Codes

The following are general return codes that may be returned upon completion of a Sapera function:

- CORSTATUS_OK
  Returned upon the successful completion of a function.
- CORSTATUS_NO_MESSAGING_MEMORY
  Returned if there is not enough contiguous messaging memory available for sending or receiving a message. See "Configuring Sapera" in the *Sapera LT User's Manual* to learn how to increase Sapera messaging memory.
- CORSTATUS_NOT_IMPLEMENTED
  Returned if the function is not implemented on the server on which the function has been executed.
- CORSTATUS_TIMEOUT
  Returned if no answer has been received from the server on which the function has been executed.

The return values listed in the function descriptions are described in greater detail in the Error Messages section.

Buffer Module

The Buffer module functions create, read, and write memory buffers (located on the host system or onboard) of a given size and pixel format.

Capabilities

ID	Capability
0x00	CORBUFFER_CAP_PIXEL_DEPTH

CORBUFFER_CAP_PIXEL_DEPTH

Description	Specifies if the CORBUFFER_PRM_PIXEL_DEPTH parameter can be modified.
Type	UINT32
Values	TRUE, the CORBUFFER_PRM_PIXEL_DEPTH parameter can be modified. FALSE, the CORBUFFER_PRM_PIXEL_DEPTH parameter cannot be modified.
See Also	CORBUFFER_PRM_PIXEL_DEPTH

Parameters

ID	Parameters	Attribute
0x00	CORBUFFER_PRM_FORMAT	Read only
0x01	CORBUFFER_PRM_TYPE	Read only
0x02	CORBUFFER_PRM_DATASIZE	Read only
0x03	CORBUFFER_PRM_DATADEPTH	Read only
0x04	CORBUFFER_PRM_XMIN	Read/Write
0x05	CORBUFFER_PRM_YMIN	Read/Write
0x06	CORBUFFER_PRM_WIDTH	Read/Write
0x07	CORBUFFER_PRM_HEIGHT	Read/Write
0x08	CORBUFFER_PRM_ADDRESS	Read only
0x09	CORBUFFER_PRM_PHYSADDRESS	Read only
0x0a	CORBUFFER_PRM_ROOT	Read only
0x0b	CORBUFFER_PRM_PARENT	Read only
0x0c	CORBUFFER_PRM_MEM_WIDTH	Read only
0x0d	CORBUFFER_PRM_MEM_HEIGHT	Read only
0x0e	CORBUFFER_PRM_STATE	Read/Write
0x0f	CORBUFFER_PRM_SIGNED	Read only
0x10	CORBUFFER_PRM_NPAGES	Read only
0x11	CORBUFFER_PRM_PAGE	Read/Write
0x12	CORBUFFER_PRM_COUNTER_STAMP	Read/Write
0x13	CORBUFFER_PRM_SPACE_USED	Read/Write
0x14	Reserved	
0x15	Reserved	
0x16	Reserved	
0x17	Reserved	
0x18	CORBUFFER_PRM_LOCKED	Read/Write
0x19	CORBUFFER_PRM_PITCH	Read only
0x1a	CORBUFFER_PRM_PIXEL_DEPTH	Read/Write

CORBUFFER_PRM_ADDRESS

Description 32-bit address of the buffer

Type UINT32

Note This is a linear address that allows the requesting process direct access to the buffer's contents, and is only valid within this processes address space.
The address of a buffer allocated in video memory can only be obtained if the buffer

parameter CORBUFFER_PRM_LOCKED has been set to a non-zero value.

CORBUFFER_PRM_COUNTER_STAMP

Description Count at which a specific event occurred.

Type CORCOUNT

See Also See CorXferLinkCounterStamp to link a counter device to a buffer resource.

CORBUFFER_PRM_DATADEPTH

Description Number of bits per buffer element.

Type UINT32

CORBUFFER_PRM_DATASIZE

Description Size of one buffer element (in bytes).

Type UINT32

Values This value does depend on the buffer format.

CORBUFFER_PRM_FORMAT

Description Buffer data format. Defines a single buffer element's type.

Type UINT32

Values For a detailed description of the values, see CorBufferNew

CORBUFFER_PRM_HEIGHT

Description Height of the buffer region of interest (ROI).

Type UINT32

Values Range within [0...MEM_HEIGHT]. Applies only to child buffers. For information on creating child buffers, see CorBufferNewChild

CORBUFFER_PRM_LOCKED

Description Defines if a buffer has been locked.

Type UINT32

Values Boolean. A non-zero value indicates the buffer is locked.

Note This parameter is used only for buffers created in video memory (i.e., off-screen video buffers or overlay buffers). The buffer address can only be obtained if it is locked, otherwise the address could become invalid if another application (like a screen saver, full-screen DOS prompt, or pressing CTRL-ALT-DEL) takes control of the video memory.

Insure that the buffer is only locked when necessary. If a buffer is used for processing, lock it just before its address is obtained and then unlock it once the processing is done.

CORBUFFER_PRM_MEM_HEIGHT

Description Buffer height. Defines the number of rows in the buffer.

Type UINT32

CORBUFFER_PRM_MEM_WIDTH

Description Buffer width. Defines the number of pixels per line in the buffer.

Type UINT32

CORBUFFER_PRM_NPAGES

Description Defines the number of pages within the buffer.

Type UINT32

CORBUFFER_PRM_PAGE

Description Defines the buffer's active page for a multi-pages buffer.

Type UINT32

CORBUFFER_PRM_PARENT

Description Child buffer's parent handle.

Type CORBUFFER

Note Applies only to child buffers, created as described in CorBufferNewChild

CORBUFFER_PRM_PHYSADDRESS

Description Physical address of the buffer.

Type UINT32

Note The buffer physical address can be used to allow PCI devices access to the buffer contents. This address can be mapped to a linear address (using CorManMapBuffer) by a process running on the host, so as to give the process access to the buffer's contents.
The 32-bit address of a buffer allocated in video memory can only be obtained if the buffer's CORBUFFER_PRM_LOCKED has been set to a non-zero value.

CORBUFFER_PRM_PITCH

Description Buffer's memory pitch

Type UINT32

Values Contains the offset (in bytes) between two pixels in the same column on two consecutive lines, within the same buffer.

Note When an off-screen or overlay buffer is created, the memory region allocated may be wider than the requested width (for alignment purposes, hardware optimization, etc.). In this case, each buffer line is padded with extra bytes to get a buffer that is CORBUFFER_PRM_PITCH bytes wide. Care should be taken not to access these extra bytes, especially if the buffer has been allocated in video memory.

CORBUFFER_PRM_PIXEL_DEPTH

Description The buffer's number of significant bits per component. By default this value is initialized to the maximum. Refer to Data Formats section to know the maximum pixel depth value for a specific format. This maximum value may also be obtained using the **CorManGetPixelDepthMax** function.

Type UINT32

Values Unsigned integer ranging from **CorManGetPixelDepthMin**(format) to **CorManGetPixelDepthMax**(format), where *format* is given by CORBUFFER_PRM_FORMAT.

Note An example usage of this parameter is when acquiring into a 16-bit monochrome buffer using a 10-bit camera. The CORBUFFER_PRM_PIXEL_DEPTH may be manually set to 10 in order to keep the information within the buffer. If the buffer is saved (using the File module) under the DALSA Format (CRC), the pixel depth value is stored. When processing the buffer the pixel depth value may be used to process the significant bits only (e.g., creating a LUT with the minimum required size).

CORBUFFER_PRM_ROOT

Description Root buffer's handle.

Type CORBUFFER

Values Contains the buffer handle on top of the hierarchy. For information on creating child buffers, see CorBufferNewChild

CORBUFFER_PRM_SIGNED

Description Sign of the buffer elements.

Type UINT32

Values CORBUFFER_VAL_FORMAT_UNSIGNED
CORBUFFER_VAL_FORMAT_SIGNED

CORBUFFER_PRM_SPACE_USED

Description Space used in the buffer. This parameter is automatically updated by the transfer resource when transferring variable length data streams in a buffer. This parameter is also valid for a fixed length data stream, in which case it is computed directly from the buffer dimensions and data format.

Type UINT32

Values Unsigned integer representing the size in bytes of the last data stream transferred in the buffer.

CORBUFFER_PRM_STATE

Description Buffer state. Defines if the buffer is empty or full. If the buffer state is full then the state can also indicate an overflow condition.

This parameter may be used as a flag in an automated processing task. The EMPTY state indicates that new data can be written to the buffer while the FULL state indicates that data has not been processed. The OVERFLOW state indicates errors such as limited data bandwidth. This parameter can also be used in conjunction with the transfer module while cycling through a list of buffers.

For more information on using this parameter, see CORXFER_PRM_CYCLE_MODE. To ensure code portability, you should use the macros CORBUFFER_STATE_IS_EMPTY (UINT32 state), CORBUFFER_STATE_IS_FULL(UINT32 state), and CORBUFFER_STATE_IS_OVERFLOW(UINT32 state) to check the buffer state.

Type UINT32

Values CORBUFFER_VAL_STATE_EMPTY
Specifies that the buffer is ready to receive new data.
CORBUFFER_VAL_STATE_FULL
Specifies that the buffer contains unprocessed data.
CORBUFFER_VAL_STATE_OVERFLOW
Specifies that the buffer contains incomplete data due to insufficient data bandwidth.

CORBUFFER_PRM_TYPE

Description Defines the attributes of the buffer memory.

Type UINT32

Values For a detailed description of possible values, see CorBufferNew.

CORBUFFER_PRM_WIDTH

Description Buffer's region of interest (ROI) width in pixels.

Type UINT32

Values Unsigned integer. Range within [0...MEM_WIDTH]. Applies only to child buffers.
For information on creating child buffers, see CorBufferNew.

CORBUFFER_PRM_XMIN

Description Buffer's region of interest (ROI) origin along the X axis.

Type UINT32

Values Unsigned integer. Range within [0...MEM_WIDTH-1]. Applies only to child buffers.
For information on creating child buffers, see CorBufferNew.

CORBUFFER_PRM_YMIN

Description Buffer's region of interest (ROI) origin along the Y axis.

Type UINT32

Values Unsigned integer. Range within [0...MEM_HEIGHT-1]. Applies only to child buffers.
For information on creating child buffers, see CorBufferNew.

Macros

This section describes macros specific to the Sapera buffer module. They should always be used in your applications to ensure code portability.

CORBUFFER_FORMAT_INDEX (UINT32 format)

Definition Gets buffer format index

Return Value Unsigned integer

CORBUFFER_FORMAT_DATADEPTH (UINT32 format)

Definition Gets buffer data depth in bits

Return Value Unsigned integer

CORBUFFER_FORMAT_DATASIZE (UINT32 format)

Definition Gets buffer data size in bytes

Return Value Unsigned integer

CORBUFFER_FORMAT_IS_SIGNED (UINT32 format)

Definition Checks buffer sign

Return Value CORDATA_FORMAT_SIGNED if buffer format is signed, CORDATA_FORMAT_UNSIGNED otherwise

CORBUFFER_FORMAT_NPAGES (UINT32 format)

Definition Gets the number of pages needed to represent the buffer data format.

Return Value Unsigned integer ranging from 1 to n

Info CORBUFFER_VAL_FORMAT_RGBP8, CORBUFFER_VAL_FORMAT_RGBP16, and CORBUFFER_VAL_FORMAT_HSIP8 are currently the only multi-page buffer format supported (3 pages, one per component). All other buffer formats (monochrome and packed color) have a single page.

CORBUFFER_STATE_IS_EMPTY (UINT32 state)

Definition Checks if buffer state is empty

Return Value TRUE if buffer state is empty, FALSE otherwise

CORBUFFER_STATE_IS_FULL(UINT32 state)

Definition Checks if a buffer state is full

Return Value TRUE if buffer state is full, FALSE otherwise

Functions

Function	Description
CorBufferBayerConvert	*Converts a Bayer-encoded image to an RGB image*
CorBufferBayerWhiteBalance	*Calculates the white balance coefficients used by the Bayer filter*
CorBufferClear	*Clears contents of a buffer resource*
CorBufferClearEx	*Clears contents of a buffer resource*
CorBufferClearBlack	*Clears contents of a buffer resource with the value corresponding to black according to the buffer format*
CorBufferConvertFormat	*Transfers images from the source buffer to the destination buffer and performs pixel format conversion if required*
CorBufferCopy	*Copies contents of a buffer resource into another buffer resource*
CorBufferCopyRect	*Copies a rectangular area of a buffer resource into another buffer resource*
CorBufferFree	*Frees handle to a buffer resource*
CorBufferGetPrm	*Gets buffer parameter value from a buffer resource*
CorBufferLoad	*(Obsolete) Loads an image from a file into a buffer resource*
CorBufferMap	*Maps a buffer using physical memory into the current process virtual address space*
CorBufferMapEx	*Map a region of a buffer using physical memory into the current process virtual address space*
CorBufferMergeComponents	*Merges source buffers into the different components of a color buffer*
CorBufferNew	*Creates in a specified server's memory a new buffer resource*
CorBufferNew1D	*Creates in a specified server's memory a new 1D buffer resource*
CorBufferNew2D	*Creates in a specified server's memory a new 2D buffer resource*
CorBufferNewChild	*Creates a new buffer resource within an existing buffer resource*
CorBufferNew1DChild	*Creates a new buffer resource within an existing 1D buffer resource*
CorBufferNew2DChild	*Creates a new buffer resource within an existing 2D buffer resource*
CorBufferNewEx	*Creates from a file and in a specified server's memory a new buffer resource*
CorBufferNew1DEx	*Creates from a file and in a specified server's memory a new 1D buffer resource*
CorBufferNew2DEx	*Creates from a file and in a specified server's memory a new 2D buffer resource*
CorBufferNewShared	*Creates in a server's memory a new buffer resource of the specified type that can be shared with other processes running on this server*
CorBufferNewSharedEx	*Creates in a specified server's memory a new buffer resource that references a previously allocated shared buffer.*
CorBufferRead	*Reads a series of elements from a buffer resource*

CorBufferReadDots	*Reads a set of elements from a buffer resource*
CorBufferReadElement	*Reads an element from a buffer resource*
CorBufferReadElementEx	*Reads an element from a buffer resource*
CorBufferReadLine	*Reads a set of linearly positioned elements from a buffer resource*
CorBufferReadRect	*Reads a set of elements forming a rectangular area from a buffer resource*
CorBufferSave	*(Obsolete) Saves to a file the contents of a buffer resource*
CorBufferSetPrm	*Sets a simple buffer parameter of a buffer resource*
CorBufferSetPrmEx	*Sets a complex buffer parameter of a buffer resource*
CorBufferSplitComponents	*Splits a color buffer into each of its components*
CorBufferUnmap	*Unmaps the buffer physical memory from the current process virtual address space*
CorBufferWrite	*Writes a series of elements into a buffer resource*
CorBufferWriteDots	*Writes a set of elements into a buffer resource*
CorBufferWriteElement	*Writes an element into a buffer resource*
CorBufferWriteElementEx	*Writes an element into a buffer resource*
CorBufferWriteLine	*Writes a set of linearly positioned elements into a buffer resource*
CorBufferWriteRect	*Writes a set of elements forming a rectangular area into a buffer resource*

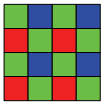
CorBufferBayerConvert

Converts a Bayer-encoded image to an RGB image.

Prototype CORSTATUS **CorBufferBayerConvert**(CORBUFFER *src*, CORBUFFER *dst*, UINT32 *options*, CORDATA *wb*, CORLUT *lut*);

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

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



The missing color channel values are determined using neighbouring pixel values for the color channel in question either by linear interpolation (CORBUFFER_VAL_BAYER_METHOD_1) or by one of the advanced methods (CORBUFFER_VAL_BAYER_METHOD_2 or METHOD_3). The advanced methods are more computationally expensive than the interpolation method but gives better image quality when the input image contains many strong edges.

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

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

The output lookup table (*lut*) may be used to apply a color correction after the filtering. A commonly used correction is gamma (*CorLutGamma* function of LUT module).

Input

src Input buffer handle. The input buffer format must be one of the following:
CORBUFFER_VAL_FORMAT_UINT8 CORBUFFER_VAL_FORMAT_UINT16

dst Output buffer handle. The output buffer format can be one of the following:
CORBUFFER_VAL_FORMAT_RGB888
CORBUFFER_VAL_FORMAT_RGB8888

If the input format is CORBUFFER_VAL_FORMAT_UINT16, the output buffer format can also be CORBUFFER_VAL_FORMAT_RGB101010.

options This value must contain one of the **alignment options** listed below. The alignment mode must correspond to the upper left 2x2 square of your camera's bayer scheme. If the input buffer is a child, the alignment mode is internally recalculated with respect to the upper left corner.

CORBUFFER_VAL_BAYER_ALIGN_GB_RG 

CORBUFFER_VAL_BAYER_ALIGN_BG_GR 

CORBUFFER_VAL_BAYER_ALIGN_RG_GB 

CORBUFFER_VAL_BAYER_ALIGN_GR_BG 

The value must be ORed with one of the **filtering options** listed below. The filtering option specifies the method used for calculating the pixel values of the three color components.

CORBUFFER_VAL_BAYER_METHOD_1	This technique, based on bi-linear interpolation, is fast but tends to smooth image edges.
CORBUFFER_VAL_BAYER_METHOD_2	This advanced technique is better for preserving image edges. However it works well only when the image has a strong green content. If not, a little amount of noise may be visible in objects.
CORBUFFER_VAL_BAYER_METHOD_3	This advanced technique is almost as good as method 2 for preserving the edges but is independent of the image green content. Little color artifacts of 1 pixel may be visible in edges.

wb White balance coefficients. Can be calculated by *CorBufferBayerWhiteBalance* or set manually as follows:

```
CORDATA wb;
wb.frgb.red = <Red Gain>
wb.frgb.green = <Green Gain>
wb.frgb.blue = <Blue Gain>
```

If no white balance is required, all gains must be set to 1.0.

lut LUT handle. color lookup Table applied after the filtering for color adjustment, e.g., gamma correction. The number of entries required by the LUT must be 2^N, where N is the buffer's pixel depth (CORBUFFER_PRM_PIXEL_DEPTH). The LUT format must be one of the following according to the output format: CORLUT_VAL_FORMAT_COLORNI8 CORLUT_VAL_FORMAT_COLORNI16

If no correction is required, can be set to CORHANDLE_NULL.

Output *None*

Return Value CORSTATUS_INCOMPATIBLE_BUFFER
CORSTATUS_INCOMPATIBLE_LUT
CORSTATUS_ARG_INVALID_VALUE

See Also CorBufferBayerWhiteBalance, CORBUFFER_PRM_PIXEL_DEPTH, CorLutNew and CorLutGamma

CorBufferBayerWhiteBalance

Calculates the white balance coefficients used by the Bayer filter.

Prototype CORSTATUS **CorBufferBayerWhiteBalance**(CORBUFFER *src*, UINT32 *options*, PCORDATA *pWB*);

Description Calculates the white balance coefficients used by CorBufferBayerConvert on a Bayer-encoded input image. The input buffer should be a region-of-interest (ROI) of a Bayer-encoded image containing a uniformly illuminated white region. The intensity of the pixels should be as high as possible but not saturated. The coefficients are calculated as follows:

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

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

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

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

Input

src Input buffer handle. The input buffer format must be one of the following:
CORBUFFER_VAL_FORMAT_UINT8 CORBUFFER_VAL_FORMAT_UINT16

options Used for selecting alignment. This value must contain one of the following:

CORBUFFER_VAL_BAYER_ALIGN_GB_RG 

CORBUFFER_VAL_BAYER_ALIGN_BG_GR 

CORBUFFER_VAL_BAYER_ALIGN_RG_GB 

CORBUFFER_VAL_BAYER_ALIGN_GR_BG 

Output *pWB* Address of a CORDATA structure to store the coefficients.

Return Value CORSTATUS_INCOMPATIBLE_BUFFER
CORSTATUS_ARG_INVALID_VALUE

See Also CorBufferBayerConvert and CorBufferNewChild

CorBufferClear

Clear contents of a buffer resource

Prototype CORSTATUS **CorBufferClear**(CORBUFFER hB*uffer*, UINT32 *value*);

Description	Clears contents of a buffer resource by writing a color value to all buffer elements. The *value* parameter will be read as an UINT8, UINT16, or an UINT32 according to the buffer's element size. If the element size is larger than an UINT32, use CorBufferClearEx. For certain formats, such as UYVY and YUY2, the color corresponding to black is not 0. Therefore, make certain that the color value assigned to *value* is defined according the pixel format of the buffer. CorBufferClearBlack
Input	*hBuffer* Buffer resource handle *value* Color value
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorBufferClearEx and CorBufferClearBlack

CorBufferClearEx

Clear contents of a buffer resource

Prototype	CORSTATUS **CorBufferClearEx**(CORBUFFER *hBuffer*, const *void *value*, UINT32 *size*);
Description	Clears contents of a buffer resource by writing a color value to all buffer elements. Required for an element size larger then UINT32. For an element size less than or equal to UINT32, use either CorBufferClear (suggested) or CorBufferClearEx.
Input	*hBuffer* Buffer resource handle *value* Color value *size* Element size
Output	None
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE
See Also	CorBufferClear and CorBufferClearEx

CorBufferClearBlack

Clear buffer resource contents with the corresponding black value as per the buffer format

Prototype	CORSTATUS **CorBufferClearBlack**(CORBUFFER *hBuffer*);
Description	Clears buffer resource contents to the corresponding black color. The color value is automatically determined according to the buffer format. For certain formats such as UYVY and YUY2, the color corresponding to black is not 0. Thus calling this function instead of CorBufferClear with a value of 0, guarantees that the buffer will appear black.
Input	*hBuffer* Buffer resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

See Also CorBufferClear and CorBufferClearEx

CorBufferConvertFormat

Transfer image from source buffer to destination buffer and perform pixel format conversion if required

Prototype CORSTATUS **CorBufferConvertFormat**(CORBUFFER *hSrc*, CORBUFFER *hDst*, UINT32 *options*);

Description Transfers image from the source buffer to the destination buffer and performs required pixel format conversion.

Input

hSrc Source buffer resource handle

options Any of the following options can be specified:

CORBUFFER_CONVERT_RANGE_CLIP
When a source pixel value is outside the destination buffer format's range, it gets clipped to the nearest valid value. This is the default method.

CORBUFFER_CONVERT_RANGE_REMAP
The source buffer format is mapped to the destination buffer's format. When the source and destination pixel depth are different (e.g., when converting from MONO16 to MONO8) the buffer CORBUFFER_PRM_PIXEL_DEPTH value is used to determine how the pixel values are remapped. This option cannot be used when the destination is floating-point.

Output *hDst* Destination buffer resource handle

Return Value
CORSTATUS_ARG_INVALID
CORSTATUS_INCOMPATIBLE_BUFFER
CORSTATUS_INVALID_HANDLE
CORSTATUS_NOT_IMPLEMENTED

See Also CorBufferMergeComponents and CorBufferSplitComponents

CorBufferCopy

Copy contents of a buffer resource into another buffer resource

Prototype CORSTATUS **CorBufferCopy**(CORBUFFER *hSrc*, UINT32 *x*, UINT32 *y*, CORBUFFER *hDst*);

Description Copies the source buffer to location (*x*,*y*) of the destination buffer. When the source buffer is larger than the destination buffer, only the section of the source that fits into the destination is copied. Buffers do not have to be located on the same server.

Input

hSrc Buffer resource handle (source)

x Horizontal offset in destination buffer

y Vertical offset in destination buffer

hDst Buffer resource handle (destination)

Output None

Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE
See Also	CorBufferCopyRect

CorBufferCopyRect

Copy a rectangular area of a buffer resource into another buffer resource

Prototype	CORSTATUS **CorBufferCopyRect**(CORBUFFER *hSrc*, UINT32 *xSrc*, UINT32 *ySrc*, UINT32 *width*, UINT32 *height*, CORBUFFER hD*st*, UINT32 *xDst*, UINT32 *yDst*);	
Description	Copies a rectangular area of the source buffer, defined by (*xSrc*, *ySrc*, *width*, *height*), to the location (*xDst*, *yDst*) in the destination buffer. When the source area is larger than the destination buffer, only the section of the source that fits the destination is copied. Buffers do not have to be located on the same server.	
Input	*hSrc*	Source buffer resource handle
	xSrc	Horizontal offset of rectangle in source buffer
	ySrc	Vertical offset of rectangle in source buffer
	width	Horizontal length of the rectangle
	height	Vertical length of the rectangle
	hDst	Destination buffer resource handle
	xDst	Horizontal offset in destination buffer
	yDst	Vertical offset in destination buffer
Output	None	
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	For 1-bit data buffers, *xSrc*, *xDst* and *width* must be a multiple of 8.	
See Also	CorBufferCopy	

CorBufferFree

Free the handle to a buffer resource

Prototype	CORSTATUS **CorBufferFree**(CORBUFFER *hBuffer*);	
Description	Free the handle and all resources used by a buffer resource. Any child buffers **MUST** be freed before calling CorBufferFree for the parent buffer.	
Input	*hBufer*	Buffer resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_RESOURCE_LINKED	
See Also	CorBufferNew, CorBufferNewChild and CorBufferNewEx	

CorBufferGetPrm

Gets buffer parameter value from a buffer resource

Prototype	CORSTATUS **CorBufferGetPrm**(CORBUFFER *hBuffer*, UINT32 *prm*, void **value*);	
Description	Get the buffer parameter value from a buffer resource. See the section Parameters for a descriptive list of all buffer parameters.	
Input	*hBuffer*	Buffer resource handle
	prm	Buffer parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID	
See Also	CorBufferSetPrm, CorBufferSetPrmEx and Parameters Section	

CorBufferLoad

Load a image file into a buffer resource (obsolete)

Prototype	CORSTATUS **CorBufferLoad**(CORBUFFER *hBuffer*, const char **filename*, UINT32 *options*);	
Description	This function is obsolete and should not be used for new applications. Use the file module API function CorFileLoad instead.	
Input	*hBuffer*	Buffer resource handle
	filename	String specifying the path and filename
	options	CORFILE_VAL_FORMAT_BMP (Windows Bitmap file format) CORFILE_VAL_FORMAT_CRC (DALSA raw file format)
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_OPEN_ERROR CORSTATUS_FILE_READ_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID_VALUE	
See Also	CorBufferSave	

CorBufferMap

Map a buffer using physical memory into the virtual address space of the current process.

Prototype	CORSTATUS **CorBufferMap**(CORBUFFER *hBuffer*);	
Description	Maps a buffer using physical memory into the current process virtual address space	
Input	*hBuffer*	Buffer resource handle
Notes	This function should only be used for a buffer that has been created using the CORBUFFER_VAL_TYPE_UNMAPPED buffer type. For mapping a region, see the CorBufferMapEx function.	

Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_RESOURCE_IN_USE CORSTATUS_NO_MEMORY
See Also	CorBufferMapEx, CorBufferUnmap

CorBufferMapEx

Map a region of a buffer using physical memory into the virtual address space of the current process.

Prototype	CORSTATUS **CorBufferMapEx**(CORBUFFER *hBuffer, UINT64 offset, UINT32 size*);	
Description	Map a region of a buffer using physical memory into the current process virtual address space	
Input	*hBuffer*	Buffer resource handle
	offset	Offset in byte from the origin
	size	Size in byte of the region to be mapped
Notes	This function should only be used for a buffer that has been created using the CORBUFFER_VAL_TYPE_ UNMAPPED buffer type.	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_ARG_INVALID_VALUE CORSTATUS_RESOURCE_IN_USE CORSTATUS_NO_MEMORY	
See Also	CorBufferUnmap	

CorBufferMergeComponents

Merge source buffers into the different components of a color buffer

Prototype	CORSTATUS **CorBufferMergeComponents**(CORBUFFER *hCompA*, CORBUFFER *hCompB*, CORBUFFER *hCompC*, CORBUFFER *hDst*, UINT32 *options*);	
Description	Merges source buffers into the different components of a color buffer.	
Input	*hCompA*	Source buffer resource handle (can be set to NULL).
	hCompB	Source buffer resource handle (can be set to NULL).
	hCompC	Source buffer resource handle (can be set to NULL).
	options	Any of the following options can be specified: CORBUFFER_CONVERT_RANGE_CLIP When a source pixel's value is outside the destination buffer format's range, it gets clipped to the nearest valid value. This is the default method. CORBUFFER_CONVERT_RANGE_REMAP The source buffer's format is mapped to the destination buffer's format. When the source's and the destination's pixel depth are different (e.g., when converting from MONO16 to MONO8) the buffers' CORBUFFER_PRM_PIXEL_DEPTH value is used to determine how the pixel values are remapped. This option cannot be used when the destination is floating-point.
Output	*hDst*	Destination buffer resource handle.

Return Value	CORSTATUS_ARG_INVALID CORSTATUS_INCOMPATIBLE_BUFFER CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_IMPLEMENTED
Note	All the source buffers should have the same format, dimensions, and pixel depth.
See Also	CorBufferConvertFormat and CorBufferSplitComponents

CorBufferNew

Create in a specified server's memory a new buffer resource

Prototype	CORSTATUS **CorBufferNew**(CORSERVER *hServer*, UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *type*, CORBUFFER **hBuffer*);	
Description	Creates in a server's memory a new buffer resource of the specified type, format, and size.	
Input	*hServer*	Server handle
	width	Width of new buffer in pixels
	height	Height of new buffer in pixels
	format	See Data Formats for detailed format descriptions. **Monochrome** and **Unsigned integer** CORBUFFER_VAL_FORMAT_MONO1 CORBUFFER_VAL_FORMAT_MONO8 CORBUFFER_VAL_FORMAT_MONO16 CORBUFFER_VAL_FORMAT_MONO32 **Integer (monochrome with sign)** CORBUFFER_VAL_FORMAT_INT8 CORBUFFER_VAL_FORMAT_INT16 CORBUFFER_VAL_FORMAT_INT32 **Color** CORBUFFER_VAL_FORMAT_RGB5551 CORBUFFER_VAL_FORMAT_RGB565 CORBUFFER_VAL_FORMAT_RGB888 CORBUFFER_VAL_FORMAT_RGB8888 CORBUFFER_VAL_FORMAT_RGB101010 CORBUFFER_VAL_FORMAT_RGB161616 CORBUFFER_VAL_FORMAT_RGB16161616 CORBUFFER_VAL_FORMAT_RGBP8 CORBUFFER_VAL_FORMAT_RGBP16 CORBUFFER_VAL_FORMAT_UYVY CORBUFFER_VAL_FORMAT_YUY2 CORBUFFER_VAL_FORMAT_YVYU CORBUFFER_VAL_FORMAT_YUYV CORBUFFER_VAL_FORMAT_Y411 CORBUFFER_VAL_FORMAT_Y211 CORBUFFER_VAL_FORMAT_YUV CORBUFFER_VAL_FORMAT_HSV CORBUFFER_VAL_FORMAT_HSI CORBUFFER_VAL_FORMAT_HSIP8

Other

CORBUFFER_VAL_FORMAT_FLOAT
CORBUFFER_VAL_FORMAT_COMPLEX
CORBUFFER_VAL_FORMAT_POINT
CORBUFFER_VAL_FORMAT_FPOINT

type The following six buffer types are valid (See the "*Sapera LT User's Manual*" for further information).

CORBUFFER_VAL_TYPE_CONTIGUOUS
The buffer is allocated in Contiguous Memory. The buffer data is contained in a single memory block (no segmentation). Allocated buffers can be used as source and destination for the transfer resource.

CORBUFFER_VAL_TYPE_SCATTER_GATHER
The buffer is allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list can be constructed. This type allows allocation of very large sized buffers used as source and destination for the transfer resource. Note that the maximum amount of memory that can be allocated with respect to available memory on the computer depends upon the operating system and the application(s) used.

CORBUFFER_VAL_TYPE_VIRTUAL
Similar to a scatter-gather buffer except that the memory pages are not locked. This type allows allocation of very large buffers, but they cannot be used as source or destination for the transfer resource.

CORBUFFER_VAL_TYPE_ UNMAPPED
The buffer is allocated in system memory. Pages are allocated but not mapped into the process virtual address space. Before trying to retrieve the buffer virtual address, the buffer's memory has to be mapped into the process virtual address space by calling either the CorBufferMap or the CorBufferMapEx function. This buffer type can be logically ORed with either CORBUFFER_VAL_TYPE_SCATTER_GATHER or CORBUFFER_VAL_TYPE_CONTIGUOUS to create a source buffer or destination buffer for the transfer resource.

CORBUFFER_VAL_TYPE_OFFSCREEN
The buffer is allocated in system memory. The Display Module view created using this buffer type may use the display adapter's hardware to copy the system memory buffer to the display memory. A system memory off-screen buffer can be created using any pixel format, but calling CorViewShow with its corresponding view will take longer to execute if its pixel format is not listed in the CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN parameter.

CORBUFFER_VAL_TYPE_VIDEO
The buffer is allocated in off-screen video memory. The view created using a buffer of this type uses the display adapter's hardware to perform a fast copy from video memory to video memory. Typically, a buffer of this type is used when a graphical element is reused for several consecutive frames without modification. In this case, it is more efficient to keep this element in video memory and use the display hardware to copy it to the appropriate position in each frame.

CORBUFFER_VAL_TYPE_OVERLAY
This buffer is allocated in video memory. Once you create a view using this

buffer and call CorViewShow once, the display adapter's overlay hardware will keep updating the display with the buffer's contents without additional CorViewShow calls. The pixel format of an overlay buffer must be listed in the CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY parameter. Typically, overlay buffers will support more pixel formats (like YUV) than off-screen buffers. Also, color keying is supported for overlays. The behavior of the overlay regarding key colors is determined by the CORVIEW_PRM_MODE parameter.

CORBUFFER_VAL_TYPE_DUMMY
No memory is allocated for a dummy buffer in order that it does not contain any data elements. This type of buffer may be used as a placeholder by the Transfer Module when no data is to be physically transferred.

Output *hBuffer* Buffer resource handle

Return Value
CORSTATUS_ARG_INVALID_VALUE
CORSTATUS_ARG_NULL (if *hBuffer* is NULL)
CORSTATUS_DDRAW_ERROR
CORSTATUS_DDRAW_NOT_AVAILABLE
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY

Notes
Type CORBUFFER_VAL_TYPE_MONO1
(1-bit data depth) buffers are created with the following restrictions:
- Width must be a multiple of 8
- Type must be CONTIGUOUS, SCATTER_GATHER, or VIRTUAL

Type CORBUFFER_VAL_TYPE_OFFSCREEN, CORBUFFER_VAL_TYPE_VIDEO, & CORBUFFER_VAL_TYPE_OVERLAY
utilize Windows and VGA hardware Direct Draw support to create and display buffers in off-screen VGA surfaces. Under different combinations of Windows version, DirectX version, and the VGA driver, the following three conditions may cause failure with Direct Draw in the creation of off-screen surface buffers.
- Having a screen saver activated.
- Opening a full screen Command session (DOS prompt).
- Pressing CTL-ALT-DEL which hides the current Desktop and displays a Windows menu.
Properly written Sapera applications should present any Direct Draw errors as a windows message box to the user. As part of the message, the above points can be presented as solutions to the problem. It is suggested that these conditions be avoided on the application target system. Review and test each condition to understand the behavior in your environment.

CorBufferNew1D

Create in a specified server's memory at a given address a new 1D buffer resource

Prototype CORSTATUS **CorBufferNew1D**(CORSERVER *hServer*, UINT32 *width*, UINT32 *format*, UINT32 *type*, CORBUFFER **hBuffer*);

Description Creates in a specified server's memory a new one-dimensional buffer resource of the specified type, format, and size.

Input *hServer* Server handle
width Width of new child buffer in pixels
format Format of new buffer. See CorBufferNew.

	type	Type of new buffer.
Output	*hBuffer*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if hBuffer is NULL) CORSTATUS_DDRAW_ERROR CORSTATUS_DDRAW_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
See Also	CorBufferFree	

CorBufferNew2D

Create in a specified server's memory at a given address a new 2D buffer resource

Prototype	CORSTATUS **CorBufferNew2D**(CORSERVER *hServer*, UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *type*, CORBUFFER **hBuffer*);	
Description	Creates in a specified server's memory a new two-dimensional buffer resource of the specified type, height, format, and size.	
Input	*hServer*	Server handle
	width	Width of new child buffer in pixels
	height	Height of new child buffer in lines
	format	Format of new buffer. See CorBufferNew.
	type	Type of new buffer
Output	*hBuffer*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL(if hBuffer is NULL) CORSTATUS_DDRAW_ERROR CORSTATUS_DDRAW_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
See Also	CorBufferFree	

CorBufferNewChild

Create a new buffer resource within an existing buffer

Prototype	CORSTATUS **CorBufferNewChild**(CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *width*, UINT32 *height*, CORBUFFER **hChild*);	
Description	Creates a new buffer resource as a sub-area of an existing parent buffer. The sub-area rectangle (specified by *x*, *y*, *width* and *height*) must not exceed any of the parent's borders.	
Input	*hBuffer*	Buffer resource handle
	x	Horizontal offset between parent and child's top left corners.
	y	Vertical offset between parent and child's top left corners.
	width	Width of new child buffer in pixels.

	height	Height of new child buffer in pixels.
Output	*hChild*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_NULL (if *hChild* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
Note	The child buffer is not a separate copy of the parent buffer. A 1-bit data child buffer must have the same dimensions as the parent buffer.	
See Also	CorBufferFree	

CorBufferNew1DChild

Create a new 1D buffer resource within an existing 1D buffer

Prototype	CORSTATUS **CorBufferNew1DChild**(CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *width*, CORBUFFER **hChild*);	
Description	Creates a new one dimensional buffer resource as a sub-area of an existing parent buffer. The sub-area (specified by *x and*, *width*) must not exceed any of the parent's borders.	
Input	*hBuffer*	Buffer resource handle
	x	Horizontal offset between parent and child's top left corners.
	width	Width of new child buffer in pixels.
Output	*hChild*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_NULL (if *hChild* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
Note	The child buffer is not a separate copy of the parent buffer. A 1-bit data child buffer must have the same dimensions as the parent buffer.	
See Also	CorBufferFree	

CorBufferNew2DChild

Create a new 2D buffer resource within an existing 2D buffer

Prototype	CORSTATUS **CorBufferNew2DChild**(CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *width*, UINT32 *height*, CORBUFFER **hChild*);	
Description	Creates a new two dimensional buffer resource as a sub-area of an existing parent buffer. The sub-area (specified by *x*, *y*, *width* and *height*) must not exceed any of the parent's borders.	
Input	*hBuffer*	Buffer resource handle
	x	Horizontal offset between parent and child's top left corners.
	y	Vertical offset between parent and child's top left corners.
	width	Width of new child buffer in pixels.
	height	Height of new child buffer in pixels.
Output	*hChild*	Buffer resource handle

Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_NULL (if *hChild* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY
Note	The child buffer is not a separate copy of the parent buffer. A 1-bit data child buffer must have the same dimensions than its parent.
See Also	CorBufferFree

CorBufferNewEx

Create in a specified server's memory at a given address a new buffer resource

Prototype CORSTATUS **CorBufferNewEx**(CORSERVER *hServer*, UINT32 UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *type*, UINT32 *physAddress*, UINT32 *virtualAddress*, CORBUFFER **hBuffer*);

Description Creates in a specified server's memory (virtual or physical address) a new buffer resource of the specified type, format, and size.

If the physical address is specified as 0, the virtual address must not be 0, and that memory is not considered as contiguous. If the physical address is not specified as 0 then the virtual address should be specified as 0. In this case, the virtual address is determined automatically from the physical address and read by getting the CORBUFFER_PRM_ADDRESS parameter.

Input	*hServer*	Server handle
	width	Width of new child buffer in pixels.
	height	Height of new child buffer in pixels.
	format	Format of new buffer, see CorBufferNew
	type	New buffer type values: CORBUFFER_VAL_TYPE_SCATTER_GATHER or CORBUFFER_VAL_TYPE_CONTIGUOUS.
	physAddress	Physical address of new buffer.
	virtualAddress	Virtual address of new buffer.
Output	*hBuffer*	Buffer resource handle

Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *hBuffer* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY
See Also	CorBufferFree and CORBUFFER_PRM_ADDRESS

CorBufferNew1DEx

Create in a specified server's memory at a given address a new 1D buffer resource

Prototype CORSTATUS **CorBufferNewEx**(CORSERVER *hServer*, UINT32 UINT32 *width*, UINT32 *format*, UINT32 *type*, UINT32 *physAddress*, UINT32 *virtualAddress*, CORBUFFER **hBuffer*);

Description Creates in a specified server's memory (virtual or physical address) a new one dimensional buffer resource of the specified type, format, and size.

If the physical address is specified as 0, the virtual address must not be 0, and that memory is not considered as contiguous. If the physical address is not specified as 0 then the virtual address should be specified as 0. In this case the virtual address is determined automatically from the physical address and read by getting the CORBUFFER_PRM_ADDRESS parameter.

Input	*hServer*	Server handle
	width	Width of new child buffer in pixels.
	format	Format of new buffer, see CorBufferNew
	type	New buffer type values: CORBUFFER_VAL_TYPE_SCATTER_GATHER or CORBUFFER_VAL_TYPE_CONTIGUOUS.
	physAddress	Physical address of new buffer.
	virtualAddress	Virtual address of new buffer.
Output	*hBuffer*	Buffer resource handle

Return Value
CORSTATUS_ARG_INVALID_VALUE
CORSTATUS_ARG_NULL (if *hBuffer* is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY

See Also CorBufferFree and CORBUFFER_PRM_ADDRESS

CorBufferNew2DEx

Create in a specified server's memory at a given address a new 2D buffer resource

Prototype CORSTATUS **CorBufferNew2DEx**(CORSERVER *hServer*, UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *type*, UINT32 *physAddress*, UINT32 *virtualAddress*, CORBUFFER **hBuffer*);

Description Creates in a specified server's memory (virtual or physical address) a new two dimensional buffer resource of the specified type, format, and size.

If the physical address is specified as 0, the virtual address must not be 0, and that memory is not considered as contiguous. If the physical address is not specified as 0 then the virtual address should be specified as 0. In this case the virtual address is determined automatically from the physical address and read by getting the CORBUFFER_PRM_ADDRESS parameter.

Input	*hServer*	Server handle
	width	Width of new child buffer in pixels.
	height	Height of new child buffer in pixels.

	format	Format of new buffer, see CorBufferNew.
	type	New buffer type value. CORBUFFER_VAL_TYPE_SCATTER_GATHER or CORBUFFER_VAL_TYPE_CONTIGUOUS.
	physAddress	Physical address of new buffer.
	virtualAddress	Virtual address of new buffer.
Output	*hBuffer*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *hBuffer* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
See Also	CorBufferFree and CORBUFFER_PRM_ADDRESS	

CorBufferNewShared

Create in a specified server's memory a new buffer resource that can be shared between processes running on this server.

Prototype	CORSTATUS **CorBufferNew**(CORSERVER *hServer*, UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *type*, PCSTR name, CORBUFFER **hBuffer*);	
Description	Creates in a server's memory a new buffer resource of the specified type, format, and size that can be shared with other processes running on this server	
Input	*hServer*	Server handle
	width	Width of new buffer in pixels
	height	Height of new buffer in pixels
	format	Format of new buffer, see CorBufferNew.
	type	The following buffer types are valid. (See the "*Sapera LT User's Manual*" for further information). CORBUFFER_VAL_TYPE_SCATTER_GATHER The buffer is allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list can be constructed. This type allows allocation of very large sized buffers used as source and destination for the transfer resource. Note that the maximum amount of memory that can be allocated with respect to available memory on the computer depends upon the operating system and the application(s) used. CORBUFFER_VAL_TYPE_VIRTUAL Similar to a scatter-gather buffer except that the memory pages are not locked. This type allows allocation of very large buffers, but they cannot be used as source or destination for the transfer resource.
	name	Name to be given to the shared buffer. This name has to be unique in the system since it will be used to reference a specific shared buffer resource
Output	*hBuffer*	Buffer resource handle
Return Value	CORSTATUS_ARG_INVALID_VALUE	

CORSTATUS_ARG_NULL (if either *hBuffer* or name is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY

Notes Type CORBUFFER_VAL_TYPE_MONO1
(1-bit data depth) buffers are restricted to a width that must be a multiple of 8.

See Also CorBufferFree

CorBufferNewSharedEx

Create in a specified server's memory a new buffer resource that references a previously allocated shared buffer

Prototype CORSTATUS **CorBufferNewSharedEx**(CORSERVER *hServer*, UINT32 *type PCSTR name,* CORBUFFER **hBuffer*);

Description Creates in a specified server's memory a new buffer resource that references a previously allocated shared buffer.

Input

HServer Server handle

Type The following buffer types are valid.
(See the "*Sapera LT User's Manual*" for further information).

CORBUFFER_VAL_TYPE_SCATTER_GATHER
The buffer is allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list can be constructed. This type allows allocation of very large sized buffers used as source and destination for the transfer resource. Note that the maximum amount of memory that can be allocated with respect to available memory on the computer depends upon the operating system and the application(s) used.

CORBUFFER_VAL_TYPE_VIRTUAL
Similar to a scatter-gather buffer except that the memory pages are not locked. This type allows allocation of very large buffers, but they cannot be used as source or destination for the transfer resource.

name Name of a previously allocated shared buffer.

Output *hBuffer* Buffer resource handle

Return Value
CORSTATUS_ARG_INVALID_VALUE
CORSTATUS_ARG_NULL (if *hBuffer* is NULL)
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY

See Also CorBufferNewShared and CorBufferFree

CorBufferRead

Read a series of elements from a buffer resource

Prototype CORSTATUS **CorBufferRead**(CORBUFFER *hBuffer*, UINT32 *offset*, void **array*, UINT32 *size*);

Description Reads a consecutive series of elements continuously from the specified buffer resource. Elements are read to the end of the buffer line and then continued from the beginning of the next line, until *length* elements have been read. Elements are then copied into a one-dimensional destination array.

Input

hBuffer	Buffer resource handle
offset	Offset to seek within the buffer prior to read (in pixels)
size	Size of transfer (*number of elements* × CORBUFFER_PRM_DATASIZE bytes). For 1-bit data buffers, size should be *((number of elements + 7) >> 3)* bytes.

Output

array	Array which can accommodate the requested size (*number of elements* × CORBUFFER_PRM_DATASIZE). For 1-bit data buffers, the array size should be *((number of elements + 7) >> 3)* bytes.

Return Value CORSTATUS_ARG_NULL (if *array* is NULL)
CORSTATUS_ARG_OUT_OF_RANGE
CORSTATUS_INVALID_HANDLE

Note Reading elements from video memory buffers may be very slow.
For 1-bit data buffers, the offset must be a multiple of 8.

See Also CorBufferWrite

CorBufferReadDots

Read a set of elements from a buffer resource

Prototype CORSTATUS **CorBufferReadDots**(CORBUFFER *hBuffer*, UINT32 *xStart*, UINT32 *yStart*, const UINT8 **dirs*, UINT32 *nDirs*, void **array*, UINT32 *size*);

Description Reads *nDirs* elements to an *array* from a set of locations in a buffer resource as defined by *xStart, yStart* and the list of directions given by *dirs.*

Input

hBuffer	Buffer resource handle
xStart	Horizontal position of first element (dot) to read
yStart	Vertical position of first element (dot) to read
dirs	Series of bytes defining the path to follow. Each byte represents the direction to the next adjacent element to read, as indicated in the following table.

Value	1	2	3	4	5	6	7	8
Direction	E	NE	N	NW	W	SW	S	SE

A value of zero or greater than 8 is ignored; the previous element is read again.

nDirs	Amount of given directions (or dots to read)
size	Size of transfer (*nDirs*×CORBUFFER_PRM_DATASIZE bytes)

Output

array	Array which can accommodate the requested number of elements (*nDirs*×CORBUFFER_PRM_DATASIZE)

Return Value CORSTATUS_ARG_NULL (if *array* is NULL)
CORSTATUS_ARG_OUT_OF_RANGE
CORSTATUS_INVALID_HANDLE

Note	Reading elements from video memory buffers may be very slow. Function not supported by 1-bit data buffers.
See Also	CorBufferWriteDots

CorBufferReadElement

Read an element from a buffer resource

Prototype	CORSTATUS **CorBufferReadElement**(CORBUFFER *hBuffer*, UINT32 *xPos*, UINT32 *yPos*, void **element*, UINT32 *size*);	
Description	Reads a single element at (*xPos,yPos*) from a buffer resource.	
Input	*hBuffer*	Buffer resource handle
	xPos	Horizontal position of the element in the buffer
	yPos	Vertical position of the element in the buffer
	size	Size of transfer (CORBUFFER_PRM_DATASIZE bytes)
Output	*element*	Current value of the element
Return Value	CORSTATUS_ARG_NULL (if *element* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	Reading elements from video memory buffers may be very slow.	
See Also	CorBufferWriteElement	

CorBufferReadElementEx

Read an element from a buffer resource

Prototype	CORSTATUS **CorBufferReadElementEx**(CORBUFFER *hBuffer*, UINT32 *xPos*, UINT32 *yPos*, CORDATA **element*);	
Description	Reads an element from a buffer resource and pack its value into the CORDATA argument.	
Input	*hBuffer*	Buffer resource handle
	xPos	Horizontal position of the element in the buffer
	yPos	Vertical position of the element in the buffer
Output	*element*	Current value of the element. See Data Types for the *CORDATA* definition.
Return Value	CORSTATUS_ARG_NULL (if *element* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	Reading elements from video memory buffers may be very slow.	
See Also	CorBufferWriteElementEx	

CorBufferReadLine

Read a set of linearly positioned elements from a buffer resource

Prototype	CORSTATUS **CorBufferReadLine**(CORBUFFER *hBuffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*, UINT32 **nElements*, void **array*, UINT32 *size*);	
Description	Read elements in a line within a buffer, from position (*x1*,*y1*) to position (*x2*,*y2*), and copies them into an array.	
Input	*hBuffer*	Buffer resource handle
	x1	Horizontal position of first element to read
	y1	Vertical position of first element to read
	x2	Horizontal position of last element to read
	y2	Vertical position of last element to read
	size	Size of transfer (MAX(ABS(x2-x1), ABS(y2-y1) + 1) × CORBUFFER_PRM_DATASIZE bytes)
Output	*array*	Array which can accommodate the requested number of elements (MAX(ABS(x2-x1), ABS(y2-y1) + 1) × CORBUFFER_PRM_DATASIZE)
	nElements	Number of elements read
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *number of elements* or *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	Reading elements from video memory buffers may be very slow. Function not supported by 1-bit data buffers.	
See Also	CorBufferWriteLine	

CorBufferReadRect

Read a set of elements forming a rectangular area from a buffer resource

Prototype	CORSTATUS **CorBufferReadRect**(CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *width*, UINT32 *height*, void **array*, UINT32 *size*);	
Description	Reads the elements of a rectangular area from a buffer resource into an array.	
Input	*hBuffer*	Buffer resource handle
	x	Horizontal offset in source buffer
	y	Vertical offset in source buffer
	width	Horizontal length of the rectangle (must be 2 or greater)
	height	Vertical length of the rectangle (must be 2 or greater)
	size	Size of transfer (*width*×*height*×CORBUFFER_PRM_DATASIZE bytes) For 1-bit data buffers, *size* should be *((width×height) >> 3)* bytes
Output	*array*	Array which can accommodate the requested number of elements (*width*×*height*×CORBUFFER_PRM_DATASIZE) For 1-bit data buffers, the *array* size should be *((width×height) >> 3)* bytes
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	

Note	Reading elements from video memory buffers may be very slow. For 1-bit data buffers, *x* and *width* must be in multiples of 8.
See Also	CorBufferWriteRect

CorBufferSave

Save to a file the content of a buffer resource (obsolete)

Prototype	CORSTATUS **CorBufferSave**(CORBUFFER *hBuffer*, const char **filename*, UINT32 *options*);	
Description	This function is obsolete and should not be used for new applications. Use the file module API function CorFileSave instead.	
Input	*hBuffer*	Buffer resource handle
	filename	String specifying the path and filename
	options	CORFILE_VAL_FORMAT_BMP (Windows Bitmap file format)
		CORFILE_VAL_FORMAT_CRC (DALSA raw file format)
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_CREATE_ERROR CORSTATUS_FILE_WRITE_ERROR CORSTATUS_INVALID_HANDLE	
See Also	CorBufferLoad	

CorBufferSetPrm

Set a simple buffer parameter of a buffer resource

Prototype	CORSTATUS **CorBufferSetPrm**(CORBUFFER *hBuffer*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple parameter of a buffer resource. A simple parameter fits inside an UINT32.	
Input	*hBuffer*	Buffer resource handle
	prm	Buffer parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY	
Note	For complex parameters, use CorBufferSetPrmEx. See the Parameters section.	
See Also	CorBufferGetPrm and CorBufferSetPrmEx	

CorBufferSetPrmEx

Set a complex buffer parameter of a buffer resource

Prototype	CORSTATUS **CorBufferSetPrmEx**(CORBUFFER *hBuffer*, UINT32 *prm*, const void **value*);
Description	Sets a complex parameter of a buffer resource. A complex parameter is greater than an UINT32.
Input	*hBuffer* Buffer resource handle *prm* Buffer parameter to set *value* New value of the parameter
Output	None
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY
Note	If the parameter size is UINT32, either CorBufferSetPrm or CorBufferSetPrmEx can be used.
See Also	CorBufferGetPrm, CorBufferSetPrm and Parameters section

CorBufferSplitComponents

Split a color buffer into each of its components

Prototype	CORSTATUS **CorBufferSplitComponents**(CORBUFFER *hSrc*, CORBUFFER *hCompA*, CORBUFFER *hCompB*, CORBUFFER *hCompC*, UINT32 *options*);
Description	Splits a color buffer into each of its components.
Input	*hSrc* Source buffer resource handle. *options* Any of the following options can be specified: CORBUFFER_CONVERT_RANGE_CLIP When a source pixel's value is outside the destination buffer format's range, it gets clipped to the nearest valid value. This is the default method. CORBUFFER_CONVERT_RANGE_REMAP The source buffer's format is mapped to the destination buffer's format. When the source and the destination pixel depths are different (e.g., when converting from MONO16 to MONO8) the buffers' CORBUFFER_PRM_PIXEL_DEPTH value is used to determine how the pixel values are remapped. This option cannot be used when the destination is floating-point.
Output	*hCompA* Destination buffer resource handle (can be set to NULL). *hCompB* Destination buffer resource handle (can be set to NULL). *hCompC* Destination buffer resource handle (can be set to NULL).
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_INCOMPATIBLE_BUFFER CORSTATUS_NOT_IMPLEMENTED CORSTATUS_INVALID_HANDLE

Note	All the destination buffers should have the same format, dimensions, and pixel depth.
See Also	CorBufferConvertFormat and CorBufferMergeComponents

CorBufferUnmap

Unmap the buffer physical memory from the current process virtual address space

Prototype	CORSTATUS **CorBufferUnmap**(CORBUFFER *hBuffer*);
Description	Unmap the buffer physical memory from the current process virtual address space
Input	*hBuffer* Buffer resource handle
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY
See Also	CorBufferMap and CorBufferMapEx

CorBufferWrite

Write a series of elements to a buffer resource

Prototype	CORSTATUS **CorBufferWrite**(CORBUFFER *hBuffer*, UINT32 *offset*, const void **array*, UINT32 *size*);	
Description	Writes a series of elements from a one-dimensional source array to the buffer resource.	
Input	*hBuffer*	Buffer resource handle
	offset	Offset to seek within the buffer prior to write (in pixels)
	array	Array which contains the elements to be written: (*number of elements*×CORBUFFER_PRM_DATASIZE) bytes. For 1-bit data buffers, the array size should be *((number of elements + 7) >> 3)* bytes
	size	Size of transfer (*number of elements*×CORBUFFER_PRM_DATASIZE) bytes For 1-bit data buffers, the array size is *((number of elements + 7) >> 3)* bytes
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	For 1-bit data buffers, *offset* must be a multiple of 8.	
See Also	CorBufferRead	

CorBufferWriteDots

Write at specific locations a series of elements to a buffer resource

Prototype	CORSTATUS **CorBufferWriteDots**(CORBUFFER *hBuffer*, UINT32 *xStart*, UINT32 *yStart*, const UINT8 **dirs*, UINT32 *nDirs*, const void **array*, UINT32 *size*);
Description	Writes *nDirs* elements from an *array* to the set of locations in a buffer resource as defined by *xStart, yStart* and the list of directions given by *dirs.*
Input	*hBuffer* Buffer resource handle

	xStart	Horizontal position of first element (dot) to write
	yStart	Vertical position of first element (dot) to write
	dirs	Series of bytes defining the path to follow. Each byte represents the direction to the next adjacent element to write, as indicated in the following below:

Value	1	2	3	4	5	6	7	8
Direction	E	NE	N	NW	W	SW	S	SE

A value of zero or greater than 8 is ignored; the previous element is written again.

	nDirs	Amount of given directions (or dots to read)
	array	Array which contains the elements to be written (*nDirs*×CORBUFFER_PRM_DATASIZE)
	size	Size of transfer (*nDirs*×CORBUFFER_PRM_DATASIZE bytes)
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *dirs* or *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	Not supported by 1-bit data buffers.	
See Also	CorBufferReadDots	

CorBufferWriteElement

Write an element to a buffer resource

Prototype	CORSTATUS **CorBufferWriteElement**(CORBUFFER *hBuffer*, UINT32 *xPos*, UINT32 *yPos*, const void **element*, UINT32 *size*);	
Description	Writes the value pointed to by *element*, to location (*xPos*, *yPos*) in the *buffer resource*.	
Input	*hBuffer*	Buffer resource handle
	xPos	Horizontal position of the element in the buffer
	yPos	Vertical position of the element in the buffer
	element	New value for the specified location (CORBUFFER_PRM_DATASIZE)
	size	Number of bytes to write corresponding to (CORBUFFER_PRM_DATASIZE bytes)
Return Value	CORSTATUS_ARG_NULL (if *element* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
See Also	CorBufferReadElement	

CorBufferWriteElementEx

Write an element to a buffer resource

Prototype	CORSTATUS **CorBufferWriteElementEx**(CORBUFFER *hBuffer*, UINT32 *xPos*, UINT32 *yPos*, CORDATA *element*);	
Description	Writes the value in *element*, to location (*xPos*, *yPos*) in the buffer resource.	
Input	*hBuffer*	Buffer resource handle
	xPos	Horizontal position of the element in the buffer
	yPos	Vertical position of the element in the buffer
	element	New value for the specified location. See Data Types for CORDATA definition.
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
See Also	CorBufferReadElementEx	

CorBufferWriteLine

Write a set of linearly positioned elements into a buffer resource

Prototype	CORSTATUS **CorBufferWriteLine**(CORBUFFER *hBuffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*, UINT32 **nElements*, const void **array*, UINT32 *size*);	
Description	Writes a line into a buffer, from element at position (*x1*,*y1*) to element at position (*x2*,*y2*). The new value of each line element is specified by the contents of *array*. If position (*x2*,*y2*) is outside the buffer boundary the line write ends. The number of elements written is stored in *nElements*, under all conditions.	
Input	*hBuffer*	Buffer resource handle
	x1	Horizontal position of first element to be written
	y1	Vertical position of first element to be written
	x2	Horizontal position of last element to be written
	y2	Vertical position of last element to be written
	array	Array containing the elements to be written (MAX(ABS(x2-x1), ABS(y2-y1) + 1) × CORBUFFER_PRM_DATASIZE)
	size	Number of bytes to write: (MAX(ABS(x2-x1), ABS(y2-y1) + 1) × CORBUFFER_PRM_DATASIZE bytes)
Output	*nElements*	Number of elements written
Return Value	CORSTATUS_ARG_NULL (if *nElements* or *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
Note	For 1-bit data buffers, *offset* must be a multiple of 8.	
See Also	CorBufferReadLine	

CorBufferWriteRect

Write a set of elements forming a rectangular area into a buffer resource

Prototype	CORSTATUS **CorBufferWriteRect**(CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *width*, UINT32 *height*, const void **array*, UINT32 *size*);	
Description	Writes the contents of an array to a rectangular region into a buffer resource.	
Input	*hBuffer*	Buffer resource handle
	x	Horizontal offset in destination buffer
	y	Vertical offset in destination buffer
	width	Horizontal length of the rectangle (must be 2 or higher)
	height	Vertical length of the rectangle (must be 2 or higher)
	array	Array which contains the elements to be written (*width*×*height*×CORBUFFER_PRM_DATASIZE) For 1-bit data buffers, the *array* size should be *((width×height) >> 3)* bytes
	size	Size of transfer (*width*×*height*×CORBUFFER_PRM_DATASIZE bytes). For 1-bit data buffers, *size* should be *((width×height) >> 3)* bytes
Output	None	For 1-bit data buffers, *x* and *width* must be multiples of 8.
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE	
See Also	CorBufferReadRect	

Counter Module

The Counter Module is a module which counts events. These events can be external, such as a user supplied signal or can be internal, such as an internal clock. The counter can then be used as a reference by other Modules to control events (i.e., change the state of a general I/O at a specific time), timestamp acquired images, and monitor the progression of an application (by simply reading the counter value).

Capabilities

ID	Capability
0x00	CORCOUNTER_CAP_BASE_UNITS
0x01	CORCOUNTER_CAP_DIRECTION
0x02	CORCOUNTER_CAP_RESOLUTION
0x03	CORCOUNTER_CAP_DETECTION
0x04	CORCOUNTER_CAP_FREQ_MAX
0x08	CORCOUNTER_CAP_EVENT_TYPE

CORCOUNTER_CAP_BASE_UNITS

Description Specifies the counter units.

Type UINT32

Values CORCOUNTER_VAL_BASE_UNITS_TIME
Counter is based on an internal timer.

CORCOUNTER_VAL_BASE_UNITS_EXTERNAL
Counter is based on a supplied external signal.

Note The returned value is the ORed combination of the valid values.

CORCOUNTER_CAP_DETECTION

Description Specifies the event detection for which the counter will increment or decrement.

Type UINT32

Values CORCOUNTER_VAL_RISING_EDGE
Counter increments/decrements on the rising edge of the triggering event.

CORCOUNTER_VAL_FALLING_EDGE
Counter increments/decrements on the falling edge of the triggering event.

Note The returned value is the ORed combination of the valid values.

CORCOUNTER_CAP_DIRECTION

Description Specifies if the counter can increment and/or decrement.

Type	UINT32
Values	CORCOUNTER_VAL_DIRECTION_UP Counter can increment.
	CORCOUNTER_VAL_DIRECTION_DOWN Counter can decrement.
Note	The returned value is the ORed combination of the valid values.

CORCOUNTER_CAP_EVENT_TYPE

Description	Specifies which type of event the counter can signal.
Type	UINT32
Values	CORCOUNTER_VAL_EVENT_TYPE_ELAPSED_TIME Counter can signal an elapsed time event.
Note	The returned value is the ORed combination of the valid values.

CORCOUNTER_CAP_FREQ_MAX

Description	Specifies the maximum frequency (in Hz) the counter can function at.
Type	UINT32

CORCOUNTER_CAP_RESOLUTION

Description	Specifies the different resolutions available from the counter in bits.
Type	UINT32
Values	Numerical value representing the resolution of the counter. If bit 'n' is on, then the counter supports 2**(n+3) bits.

Parameters

ID	Parameters	Attribute
0x00	CORCOUNTER_PRM_LABEL	Read Only
0x01	CORCOUNTER_PRM_DEVICE_ID	Read Only
0x02	CORCOUNTER_PRM_COUNT	Read/Write
0x03	CORCOUNTER_PRM_BASE_UNITS	Read/Write
0x04	CORCOUNTER_PRM_RESOLUTION	Read/Write
0x05	CORCOUNTER_PRM_DIRECTION	Read/Write
0x06	CORCOUNTER_PRM_DETECTION	Read/Write
0x07	CORCOUNTER_PRM_STEP	Read/Write
0x08	CORCOUNTER_PRM_EVENT_TYPE	Read/Write

CORCOUNTER_PRM_BASE_UNITS

Description The counter's basic units.

Type UINT32

Values CORCOUNTER_VAL_BASE_UNITS_TIME,
Counter based on an internal timer.

CORCOUNTER_VAL_BASE_UNITS_EXTERNAL,
Counter based on a supplied external signal.

CORCOUNTER_PRM_COUNT

Description The current count in the units as specified by CORCOUNTER_PRM_BASE_UNITS.

Type CORCOUNT

Note The size of this parameter depends on the current resolution of the counter.

CORCOUNTER_PRM_DETECTION

Description Specifies the detection of the event at which the counter will increment or decrement.

Type UINT32

Values CORCOUNTER_VAL_RISING_EDGE
Counter will increment/decrement on the rising edge of the event triggering the counter.

CORCOUNTER_VAL_FALLING_EDGE
Counter will increment/decrement on the falling edge of the event triggering the counter.

Note The value is the ORed combination of the valid values.

CORCOUNTER_PRM_DEVICE_ID

Description The counter's device ID.

Type UINT32

Note CORCOUNTER_PRM_DEVICE_ID is a read-only parameter.

CORCOUNTER_PRM_DIRECTION

Description Specifies if the counter will increment or decrement.

Type UINT32

Values CORCOUNTER_VAL_DIRECTION_UP, Counter will increment.

CORCOUNTER_VAL_DIRECTION_DOWN, Counter will decrement.

CORCOUNTER_PRM_EVENT_TYPE

Description Specifies the event to signal.

Type UINT32

Values CORCOUNTER_VAL_EVENT_TYPE_ELAPSED_TIME
Signal when the time will be elapsed.

Note The value is the ORed combination of the valid values.

CORCOUNTER_PRM_LABEL

Description The Counter's string ID.

Type CHAR[128]

Values Zero-terminated array of characters with a fixed size of 128 bytes.

Note Read-only parameter.

CORCOUNTER_PRM_RESOLUTION

Description Specifies the resolution of the counter in bits.

Type UINT32

CORCOUNTER_PRM_STEP

Description Specifies the counter step.

Type UINT32

CORCOUNT Structure Definition

```
//  CORCOUNT Structure Definition

typedef union
{
    UINT8   count8;     //8-bit counter value
    UINT16  count16;    //16-bit counter value
    UINT32  count32;    //32-bit counter value
    UINT64  count64;    //64-bit counter value
} CORCOUNT, *PCORCOUNT;
```

Functions

Function	Description
CorCounterGetCap	*Gets the counter capability value*
CorCounterGetCount	*Gets the number of counters on a server*
CorCounterGetHandle	*Gets a handle to a counter device*
CorCounterGetPrm	*Gets a counter parameter value*
CorCounterIncrement	*Increment the counter device*
CorCounterRegisterCallback	*Register callback function for a counter device*
CorCounterRelease	*Releases a handle to a counter device*
CorCounterReset	*Resets a counter device*
CorCounterResetModule	*Resets the resources associated with the server's counter device(s)*
CorCounterSetPrm	*Sets a simple counter parameter*
CorCounterSetPrmEx	*Set a complex counter parameter*

CorCounterStart	*Starts a counter*
CorCounterStop	*Stops a counter*
CorCounterUnregisterCallback	*Unregister callback function for a counter device*

CorCounterGetCap

Get counter capability value

Prototype	CORSTATUS **CorCounterGetCap**(CORCOUNTER *hCounter*, UINT32 *cap*, void **value*);	
Description	Gets counter capability value.	
Input	*hCounter*	Counter resource handle
	cap	Counter device capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_CAP_INVALID	

CorCounterGetCount

Get the number of counter devices on a server

Prototype	CORSTATUS **CorCounterGetCount**(CORSERVER *hServer*, UINT32 **count*);	
Description	Gets the number of counter devices available on a server.	
Input	*hServer*	Server handle
Output	*count*	Number of counter devices
Return Value	CORSTATUS_ARG_NULL (if *count* is NULL) CORSTATUS_INVALID_HANDLE	
Note	The content of *count* is 0 when there is no Counter device available.	

CorCounterGetHandle

Get a handle to a counter device

Prototype	CORSTATUS **CorCounterGetHandle**(CORSERVER *hServer*, UINT32 *deviceId*, CORCOUNTER **hCounter*);	
Description	Gets a handle to a counter device.	
Input	*hServer*	Server handle
	deviceId	Specifies which counter device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorCounterGetCount.
Output	*hCounter*	Counter resource handle
Return Value	CORSTATUS_ARG_NULL (if *hCounter* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_RESOURCE_IN_USE	
See Also	CorCounterGetCount and CorCounterRelease	

CorCounterGetPrm

Get counter parameter value

Prototype	CORSTATUS **CorCounterGetPrm**(CORCOUNTER *hCounter*, UINT32 *prm*, void **value*);	
Description	Gets counter parameter value.	
Input	*hCounter*	Counter resource handle
	prm	Counter parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL(if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID	
See Also	CorCounterSetPrm and CorCounterSetPrmEx	

CorCounterIncrement

Increment the counter device

Prototype	CORSTATUS **CorCounterIncrement**(CORCOUNTER *hCounter*);	
Description	Increments the counter device.	
Input	*hCounter*	Counter resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorCounterRegisterCallback

Register callback function for a counter device

Prototype	CORSTATUS **CorCounterRegisterCallback(** CORCOUNTER *hCounter*, UINT32 *eventType*, PCORCALLBACK *callbackFct*, void **context*);	
Description	Registers callback function for a counter device.	
Input	*hCounter*	Counter resource handle
	eventType	Event to register: CORCOUNTER_VAL_EVENT_TYPE_ELAPSED_TIME, Call callback function when time elapsed
	callbackFct	Callback function to be registered. Callback function must be defined as: *CORSTATUS CCONV* *callback (void *context, UINT32 eventType, UINT32 eventCount);* When called, *context* will have the value specified at callback function registration; *eventType* will contain the event(s) that trigged the call to your callback function; *eventCount* should increment by one at each call, with a starting value of 1. In case the counter resource can not keep up because there is too many events to be signaled, *eventCount* will take non-consecutive values, indicating that events have been lost. See the Data Types section for the PCORCALLBACK definition.
	context	Context pointer to be passed to the callback function when called
Output	None	

Return Value	CORSTATUS_ARG_NULL (if *callbackFct* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_AVAILABLE CORSTATUS_RESOURCE_IN_USE
Note	The values may be ORed if more than one event is desired.
See Also	CorCounterUnregisterCallback

CorCounterRelease

Release handle to a counter device

Prototype	CORSTATUS **CorCounterRelease**(CORCOUNTER *hCounter*);
Description	Releases handle to a counter device.
Input	*hCounter* Counter resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorCounterGetHandle

CorCounterReset

Reset a counter device

Prototype	CORSTATUS **CorCounterReset**(CORCOUNTER *hCounter*);
Description	Resets a counter device. Restore the default values for counter parameters.
Input	*hCounter* Counter resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorCounterResetModule

Reset the resources associated with the server's counter device(s)

Prototype	CORSTATUS **CorCounterResetModule**(CORSERVER *hServer*);
Description	Resets the resources associated with the server's counter device(s). Releases all resources (handle, memory) currently allocated. Make certain that no other application is currently using any counter device resource. This function should be used cautiously.
Input	*hServer* Server handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorCounterSetPrm

Set a simple Counter parameter

Prototype	CORSTATUS **CorCounterSetPrm**(CORCOUNTER *hCounter*, UINT32 *prm*, UINT32 *value*);

Description	Sets a simple counter parameter.	
Input	*hCounter*	Counter resource handle
	prm	Counter parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_OUT_OF_RANGE CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32. If the parameter is complex, use CorCounterSetPrmEx.	
See Also	CorCounterGetPrm and CorCounterSetPrmEx	

CorCounterSetPrmEx

Set a complex counter parameter

Prototype	CORSTATUS **CorCounterSetPrmEx**(CORCOUNTER *hCounter*, UINT32 *prm*, void **value*);	
Description	Sets a complex counter parameter.	
Input	*hCounter*	Counter resource handle
	prm	Counter parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_OUT_OF_RANGECORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, use either CorCounterSetPrm or CorCounterSetPrmEx.	
See Also	CorCounterGetPrm and CorCounterSetPrm	

CorCounterStart

Start a counter device

Prototype	CORSTATUS **CorCounterStart**(CORCOUNTER *hCounter*);
Description	Starts a counter device.
Input	*hCounter* Counter resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorCounterStop

Stop a counter device

Prototype	CORSTATUS **CorCounterStop**(CORCOUNTER *hCounter*);
Description	Stops a counter device.
Input	*hCounter* Counter resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorCounterUnregisterCallback

Unregister callback function for a counter device

Prototype	CORSTATUS **CorCounterUnregisterCallback**(CORCOUNTER *hCounter,* PCORCALLBACK *callbackFct*);	
Description	Stops a counter device.	
Input	*hCounter*	Counter resource handle
	callbackFct	Callback function to unregister. See the Data Types section for the PCORCALLBACK definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorCounterRegisterCallback	

Display Module

The Display Module is used to control the display device.

Capabilities

ID	Capability
0x00	*Reserved*
0x01	CORDISPLAY_CAP_COLOR_SPACE
0x02	CORDISPLAY_CAP_BRIGHTNESS
0x03	CORDISPLAY_CAP_BRIGHTNESS_MIN
0x04	CORDISPLAY_CAP_BRIGHTNESS_MAX
0x05	CORDISPLAY_CAP_BRIGHTNESS_STEP
0x06	CORDISPLAY_CAP_CONTRAST
0x07	CORDISPLAY_CAP_CONTRAST_MIN
0x08	CORDISPLAY_CAP_CONTRAST_MAX
0x09	CORDISPLAY_CAP_CONTRAST_STEP
0x0a	CORDISPLAY_CAP_SATURATION
0x0b	CORDISPLAY_CAP_SATURATION_MIN
0x0c	CORDISPLAY_CAP_SATURATION_MAX
0x0d	CORDISPLAY_CAP_SATURATION_STEP
0x0e	CORDISPLAY_CAP_HUE
0x0f	CORDISPLAY_CAP_HUE_MIN
0x10	CORDISPLAY_CAP_HUE_MAX
0x11	CORDISPLAY_CAP_HUE_STEP
0x12	CORDISPLAY_CAP_ZOOM_HORZ
0x13	CORDISPLAY_CAP_ZOOM_HORZ_METHOD
0x14	CORDISPLAY_CAP_ZOOM_HORZ_MIN
0x15	CORDISPLAY_CAP_ZOOM_HORZ_MAX
0x16	CORDISPLAY_CAP_ZOOM_HORZ_MULT
0x17	CORDISPLAY_CAP_ZOOM_HORZ_MIN_FACTOR
0x18	CORDISPLAY_CAP_ZOOM_HORZ_MAX_FACTOR
0x19	CORDISPLAY_CAP_ZOOM_VERT
0x1a	CORDISPLAY_CAP_ZOOM_VERT_METHOD
0x1b	CORDISPLAY_CAP_ZOOM_VERT_MIN
0x1c	CORDISPLAY_CAP_ZOOM_VERT_MAX

0x1d	CORDISPLAY_CAP_ZOOM_VERT_MULT
0x1e	CORDISPLAY_CAP_ZOOM_VERT_MIN_FACTOR
0x1f	CORDISPLAY_CAP_ZOOM_VERT_MAX_FACTOR
0x20	CORDISPLAY_CAP_STEREO
0x22	CORDISPLAY_CAP_ALIGN_LEFT
0x23	CORDISPLAY_CAP_ALIGN_TOP
0x24	CORDISPLAY_CAP_ALIGN_WIDTH
0x25	CORDISPLAY_CAP_ALIGN_HEIGHT
0x26	CORDISPLAY_CAP_ALIGN_STRIDE
0x27	CORDISPLAY_CAP_WIDTH_MIN
0x28	CORDISPLAY_CAP_WIDTH_MAX
0x29	CORDISPLAY_CAP_HEIGHT_MIN
0x2a	CORDISPLAY_CAP_HEIGHT_MAX

CORDISPLAY_CAP_ALIGN_HEIGHT

Description Sets the basic factor for the height of a display surface, in bits. For example, if the display surface uses 8-bit pixels and CORDISPLAY_CAP_ALIGN_HEIGHT is set to 32, then the factor is 4 pixels. Consequently, any display surface height must be a multiple of 4 pixels.

Type UINT32

Note Only applicable to dedicated displays.

CORDISPLAY_CAP_ALIGN_LEFT

Description Alignment (in bits) of the display region left vertical edge relative to the video memory origin.

Type UINT32

Values The offset of the left vertical edge of the display region relative to the video memory origin must be a multiple of this value.

Note Only applicable to dedicated displays. This value is often the same as the width of the memory bus of the display device.

CORDISPLAY_CAP_ALIGN_STRIDE

Description The alignment of the display stride.

Type UINT32

Note Only applicable to dedicated displays. This value is often the same as the width of the memory bus of the display device.

CORDISPLAY_CAP_ALIGN_TOP

Description Display alignment (in bits) of the top horizontal edge relative to the video memory origin.

Type UINT32

Values	The offset of the top horizontal edge of the display region relative to the video memory origin must be a multiple of this value.
Note	Only applicable to dedicated displays.

CORDISPLAY_CAP_ALIGN_WIDTH

Description	Sets the basic factor for the width of a display surface, in bits. For example, if the display surface uses 8-bit pixels and CORDISPLAY_CAP_ALIGN_WIDTH is set to 32, then the factor is 4 pixels. Consequently, any display surface width must be a multiple of 4 pixels.
Type	UINT32
Note	Only applicable to dedicated displays. This value is often the same as the width of the memory bus of the display device.

CORDISPLAY_CAP_BRIGHTNESS

Description	Specifies whether the display brightness is adjustable.
Type	UINT32
Values	TRUE: The display brightness is adjustable. FALSE: The display brightness is not adjustable.
See Also	CORDISPLAY_CAP_CONTRAST, CORDISPLAY_CAP_HUE, CORDISPLAY_CAP_SATURATION and CORDISPLAY_PRM_BRIGHTNESS

CORDISPLAY_CAP_BRIGHTNESS_MAX

Description	Maximum valid value for brightness.
Type	INT32
See Also	CORDISPLAY_CAP_CONTRAST_MAX, CORDISPLAY_CAP_HUE_MAX, CORDISPLAY_CAP_SATURATION_MAX and CORDISPLAY_PRM_BRIGHTNESS

CORDISPLAY_CAP_BRIGHTNESS_MIN

Description	Minimum valid value for brightness.
Type	INT32
See Also	CORDISPLAY_CAP_CONTRAST_MIN, CORDISPLAY_CAP_HUE_MIN, CORDISPLAY_CAP_SATURATION_MIN and CORDISPLAY_PRM_BRIGHTNESS

CORDISPLAY_CAP_BRIGHTNESS_STEP

Description	Step between two consecutive valid brightness values.
Type	INT32
See Also	CORDISPLAY_CAP_CONTRAST_STEP, CORDISPLAY_CAP_HUE_STEP, CORDISPLAY_CAP_SATURATION_STEP and CORDISPLAY_PRM_BRIGHTNESS

CORDISPLAY_CAP_COLOR_SPACE

Description The display's available color spaces.

Type UINT32

Values CORDISPLAY_VAL_COLOR_SPACE_RGB
Uses the RGB (red, green, blue) primary color components to display images.

CORDISPLAY_VAL_COLOR_SPACE_MONO
Uses shades of gray to display images.

CORDISPLAY_VAL_COLOR_SPACE_YUV
Uses luminance (Y) and two chrominance (U, V) components to display images.

CORDISPLAY_VAL_COLOR_SPACE_CMYK
Uses the CMYK (cyan, magenta, yellow, black) color components to display images.

Note The values may be ORed if a display can use more than one color space.
The above values describe only the primary (visible) display surface.

CORDISPLAY_CAP_CONTRAST

Description Specifies whether the display contrast is adjustable.

Type UINT32

Values TRUE: The display contrast is adjustable.
FALSE: The display contrast is not adjustable.

See Also CORDISPLAY_CAP_BRIGHTNESS, CORDISPLAY_CAP_HUE, CORDISPLAY_CAP_SATURATION and CORDISPLAY_PRM_CONTRAST

CORDISPLAY_CAP_CONTRAST_MAX

Description Maximum valid value for contrast.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MAX, CORDISPLAY_CAP_HUE_MAX, CORDISPLAY_CAP_SATURATION_MAX and CORDISPLAY_PRM_CONTRAST

CORDISPLAY_CAP_CONTRAST_MIN

Description Minimum valid value for contrast.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MIN, CORDISPLAY_CAP_HUE_MIN, CORDISPLAY_CAP_SATURATION_MIN and CORDISPLAY_PRM_CONTRAST

CORDISPLAY_CAP_CONTRAST_STEP

Description Step between two consecutive valid contrast values.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_STEP, CORDISPLAY_CAP_HUE_STEP, CORDISPLAY_CAP_SATURATION_STEP and CORDISPLAY_PRM_CONTRAST

CORDISPLAY_CAP_HEIGHT_MAX

Description Maximum height of the display surface.

Type UINT32

CORDISPLAY_CAP_HEIGHT_MIN

Description Minimum height of the display surface.

Type UINT32

CORDISPLAY_CAP_HUE

Description Specifies whether the display hue is adjustable.

Type INT32

Values TRUE: The display hue is adjustable.
FALSE: The display hue is not adjustable.

See Also CORDISPLAY_CAP_BRIGHTNESS, CORDISPLAY_CAP_CONTRAST, CORDISPLAY_CAP_SATURATION and CORDISPLAY_PRM_HUE

CORDISPLAY_CAP_HUE_MAX

Description Maximum valid value for hue.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MAX, CORDISPLAY_CAP_CONTRAST_MAX, CORDISPLAY_CAP_SATURATION_MAX and CORDISPLAY_PRM_HUE

CORDISPLAY_CAP_HUE_MIN

Description Minimum valid value for hue.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MIN, CORDISPLAY_CAP_CONTRAST_MIN, CORDISPLAY_CAP_SATURATION_MIN and CORDISPLAY_PRM_HUE

CORDISPLAY_CAP_HUE_STEP

Description Step between two consecutive, valid hue values.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_STEP, CORDISPLAY_CAP_CONTRAST_STEP, CORDISPLAY_CAP_SATURATION_STEP and CORDISPLAY_PRM_HUE

CORDISPLAY_CAP_SATURATION

Description Specifies whether the display saturation is adjustable.

Type UINT32

Values TRUE: The display saturation is adjustable.
FALSE: The display saturation is not adjustable.

See Also CORDISPLAY_CAP_BRIGHTNESS, CORDISPLAY_CAP_CONTRAST, CORDISPLAY_CAP_HUE and CORDISPLAY_PRM_SATURATION

CORDISPLAY_CAP_SATURATION_MAX

Description Maximum valid value for saturation.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MAX, CORDISPLAY_CAP_CONTRAST_MAX, CORDISPLAY_CAP_HUE_MAX and CORDISPLAY_PRM_SATURATION

CORDISPLAY_CAP_SATURATION_MIN

Description Minimum valid value for saturation.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_MIN, CORDISPLAY_CAP_CONTRAST_MIN, CORDISPLAY_CAP_HUE_MIN and CORDISPLAY_PRM_SATURATION

CORDISPLAY_CAP_SATURATION_STEP

Description Step between two consecutive valid saturation values.

Type INT32

See Also CORDISPLAY_CAP_BRIGHTNESS_STEP, CORDISPLAY_CAP_CONTRAST_STEP, CORDISPLAY_CAP_HUE_STEP and CORDISPLAY_PRM_SATURATION

CORDISPLAY_CAP_STEREO

Description Specifies whether the display can be used to visualize stereo images.

Type UINT32

Values TRUE
FALSE

CORDISPLAY_CAP_WIDTH_MAX

Description Maximum width of the display surface.

Type UINT32

CORDISPLAY_CAP_WIDTH_MIN

Description Minimum width of the display surface.

Type UINT32

CORDISPLAY_CAP_ZOOM_HORZ

Description	Specifies whether the display supports horizontal zooming.
Type	UINT32
Values	TRUE: The display supports horizontal zooming. FALSE: The display does not support horizontal zooming.
Note	Display zoom applies to the whole display surface and not particular views.
See Also	CORVIEW_CAP_ZOOM_HORZ

CORDISPLAY_CAP_ZOOM_HORZ_MAX

Description	Maximum valid value (in pixels) for the horizontal zoom.
Type	UINT32
See Also	CORVIEW_CAP_ZOOM_HORZ_MAX

CORDISPLAY_CAP_ZOOM_HORZ_MAX_FACTOR

Description	Maximum horizontal zoom factor.
Type	UINT32
Note	An alternative to CORDISPLAY_CAP_ZOOM_HORZ. A factor of 1000 is equivalent to a 1:1 zoom (no zoom). One and/or the other may be specified.
See Also	CORVIEW_CAP_ZOOM_HORZ_MAX_FACTOR

CORDISPLAY_CAP_ZOOM_HORZ_METHOD

Description	Horizontal zooming method implemented by the display.
Type	INT32
Values	CORDISPLAY_VAL_ZOOM_METHOD_SIMPLE Simple zoom using pixel dropping or replication. CORDISPLAY_VAL_ZOOM_METHOD_INTERPOLATION Interpolated zoom. CORDISPLAY_VAL_ZOOM_METHOD_POW2 Zoom by a power of 2 (e.g., 2x, 4x, 8x, etc.).
Note	The values may be ORed if more than one zoom method applies. Display zoom applies to the whole display surface and not particular views.

CORDISPLAY_CAP_ZOOM_HORZ_MIN

Description	Minimum valid value (in pixels) for the horizontal zoom.
Type	UINT32
Note	An alternative to CORDISPLAY_CAP_ZOOM_HORZ_MIN_FACTOR. One and/or the other may be specified.

CORDISPLAY_CAP_ZOOM_HORZ_MIN_FACTOR

Description Minimum horizontal zoom factor.

Type UINT32

Note An alternative to CORDISPLAY_CAP_ZOOM_HORZ_MIN. A factor of 1000 is equivalent to a 1:1 zoom (no zoom).One and/or the other may be specified.

CORDISPLAY_CAP_ZOOM_HORZ_MULT

Description Granularity (in pixels) for the horizontal zoom parameter.

Type UINT32

Note The horizontal zoom dimension must be a multiple of this value.

CORDISPLAY_CAP_ZOOM_VERT

Description Specifies whether the display supports vertical zooming.

Type UINT32

Values TRUE: The display supports vertical zooming.
FALSE: The display does not support vertical zooming.

Note Display zoom applies to the whole display surface and not particular views.

CORDISPLAY_CAP_ZOOM_VERT_MAX

Description Maximum valid value (in pixels) for the vertical zoom.

Type UINT32

Note An alternative to CORDISPLAY_CAP_ZOOM_VERT_MAX_FACTOR.
One and/or the other may be specified.

CORDISPLAY_CAP_ZOOM_VERT_MAX_FACTOR

Description Maximum vertical zoom factor.

Type UINT32

Note An alternative to CORDISPLAY_CAP_ZOOM_VERT_MAX. A factor of 1000 is equivalent to a 1:1 zoom (no zoom). One and/or the other may be specified.

See Also CORVIEW_CAP_ZOOM_VERT_MAX_FACTOR

CORDISPLAY_CAP_ZOOM_VERT_METHOD

Description Vertical zooming method implemented by the display.

Type INT32

Values CORDISPLAY_VAL_ZOOM_METHOD_SIMPLE
Simple zoom drops or replicates pixels.

CORDISPLAY_VAL_ZOOM_METHOD_INTERPOLATION
Interpolated zoom.

CORDISPLAY_VAL_ZOOM_METHOD_POW2
Zoom by a power of 2 (e.g., 2x, 4x, 8x, etc.).

Note The values may be ORed if more than one zoom method applies.
Display zoom applies to the whole display surface and not particular views.

CORDISPLAY_CAP_ZOOM_VERT_MIN

Description Minimum valid value (in pixels) for the vertical zoom.

Type UINT32

Note An alternative to CORDISPLAY_CAP_ZOOM_VERT_MIN_FACTOR.
One and/or the other may be specified.

CORDISPLAY_CAP_ZOOM_VERT_MIN_FACTOR

Description Minimum vertical zoom factor.

Type UINT32

Note An alternative to CORDISPLAY_CAP_ZOOM_VERT_MIN. A factor of 1000 is equivalent to a 1:1 zoom (no zoom). One and/or the other may be specified.

CORDISPLAY_CAP_ZOOM_VERT_MULT

Description Granularity (in pixels) for the vertical zoom parameter.

Type UINT32

Note The vertical zoom dimension must be a multiple of this value.

Parameters

ID	Parameter	Attribute
0x00	CORDISPLAY_PRM_WIDTH	Read/Write
0x01	CORDISPLAY_PRM_HEIGHT	Read/Write
0x02	CORDISPLAY_PRM_REFRESH	Read/Write
0x03	CORDISPLAY_PRM_INTERLACED	Read/Write
0x04	CORDISPLAY_PRM_ZOOM_HORZ	Read/Write
0x05	CORDISPLAY_PRM_ZOOM_VERT	Read/Write
0x06	CORDISPLAY_PRM_BRIGHTNESS	Read/Write
0x07	CORDISPLAY_PRM_CONTRAST	Read/Write
0x08	CORDISPLAY_PRM_SATURATION	Read/Write
0x09	CORDISPLAY_PRM_HUE	Read/Write
0x0a	CORDISPLAY_PRM_PIXEL_DEPTH	Read/Write
0x0b	CORDISPLAY_PRM_LABEL	Read/Write

0x0c	CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN	Read Only
0x0d	CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY	Read Only
0x0e	CORDISPLAY_PRM_TYPE	Read Only
0x0f	CORDISPLAY_PRM_INDEX	Read Only
0x10	CORDISPLAY_PRM_ADDRESS	Read Only
0x11	CORDISPLAY_PRM_PHYS_ADDRESS	Read Only
0x12	CORDISPLAY_PRM_FORMAT	Read Only
0x13	CORDISPLAY_PRM_PITCH	Read Only

CORDISPLAY_PRM_ADDRESS

Description The memory linear address which corresponds to the display.

Type UINT32

Note This parameter is only available when the value of CORDISPLAY_PRM_TYPE is CORDISPLAY_VAL_TYPE_INDEPENDENT; i.e. a VGA display that is independent from the system VGA.

CORDISPLAY_PRM_BRIGHTNESS

Description The display brightness value.

Type UINT32

Values Valid only if CORDISPLAY_PRM_BRIGHTNESS is non-zero.
Cannot be less than CORDISPLAY_CAP_BRIGHTNESS_MIN or greater than CORDISPLAY_CAP_BRIGHTNESS_MAX.

CORDISPLAY_PRM_CONTRAST

Description The display contrast value.

Type UINT32

Values Unsigned integer.
Valid only if CORDISPLAY_PRM_CONTRAST is non-zero. Cannot be less than CORDISPLAY_CAP_CONTRAST_MIN or greater than CORDISPLAY_CAP_CONTRAST_MAX.

CORDISPLAY_PRM_FORMAT

Description The buffer data format corresponding to the current display depth.

Type UINT32

Values For a detailed description of the values, see CorBufferNew.

Note This parameter is only available when the value of CORDISPLAY_PRM_TYPE is CORDISPLAY_VAL_TYPE_INDEPENDENT; i.e. a VGA display that is independent from the system VGA.

CORDISPLAY_PRM_HEIGHT

Description	Width of the display surface.
Type	UINT32
Values	Cannot be less than the value of CORDISPLAY_CAP_HEIGHT_MIN or greater than the value of CORDISPLAY_CAP_HEIGHT_MAX.
Note	CORDISPLAY_PRM_HEIGHT is a read-only parameter on the *SYSTEM* display.

CORDISPLAY_PRM_HUE

Description	The display hue value.
Type	UINT32
Values	Valid only if CORDISPLAY_PRM_HUE is non-zero. Cannot be less than CORDISPLAY_CAP_HUE_MIN or greater than CORDISPLAY_CAP_HUE_MAX.

CORDISPLAY_PRM_INDEX

Description	The display resource index, as specified in the call to CorDisplayGetHandle.
Type	UINT32
See Also	CorDisplayGetHandle

CORDISPLAY_PRM_INTERLACED

Description	Specifies whether or not the display device is interlaced.
Type	UINT32
Values	TRUE FALSE
Note	A read-only parameter on the system display.

CORDISPLAY_PRM_LABEL

Description	The display device's string ID.
Type	CHAR[128]
Values	Zero-terminated array of characters with a fixed size of 128 bytes.
Note	CORDISPLAY_PRM_LABEL is a read-only parameter.

CORDISPLAY_PRM_PHYS_ADDRESS

Description	The physical memory address which corresponds to the display.
Type	UINT32
Note	This parameter is only available when the value of CORDISPLAY_PRM_TYPE is CORDISPLAY_VAL_TYPE_INDEPENDENT; i.e. a VGA display that is independent from the system VGA.

CORDISPLAY_PRM_PITCH

Description The display memory pitch.

Type UINT32

Values Contains the offset (in bytes) between two pixels of the same column on two consecutive lines.

Note This parameter is only available when the value of CORDISPLAY_PRM_TYPE is CORDISPLAY_VAL_TYPE_INDEPENDENT; i.e. a VGA display that is independent from the system VGA.

CORDISPLAY_PRM_PIXEL_DEPTH

Description The size of the display surface pixels (in bits).

Type UINT32

Note A read-only parameter on the system display.

CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN

Description Displays the pixel formats that the display supports for off-screen surfaces.

Type UINT32[32]

Values Zero-terminated array of UINT32 values with a fixed size of 32 dwords.

See Also CorBufferNew and Data Formats

CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY

Description Displays the pixel formats that the display supports for overlay surface.

Type UINT32[32]

Values Zero-terminated array of UINT32 values with a fixed size of 32 dwords.

Note The available pixel formats may not be correctly detected if there is another application using the display adapter's overlay hardware at the same time.

See Also CorBufferNew and Data Formats

CORDISPLAY_PRM_REFRESH

Description Display device's refresh rate.

Type UINT32

Note CORDISPLAY_PRM_REFRESH is a read-only parameter on the system display.

CORDISPLAY_PRM_SATURATION

Description The display saturation value.

Type UINT32

Values Valid only if CORDISPLAY_PRM_ SATURATION is non-zero.
Cannot be less than CORDISPLAY_CAP_SATURATION_MIN or greater than CORDISPLAY_CAP_SATURATION_MAX.

CORDISPLAY_PRM_TYPE

Description The display type.

Type UINT32

Values

CORDISPLAY_VAL_TYPE_SYSTEM
A display under the control of the primary Windows display driver. This is the same display that normally displays the Windows desktop. In Sapera, it belongs to the System server.

CORDISPLAY_VAL_TYPE_DUPLICATE
A secondary display that shows the same contents as the primary Windows VGA display..

CORDISPLAY_VAL_TYPE_EXTENDED
A secondary display that extends the desktop from the primary Windows VGA display.

CORDISPLAY_VAL_TYPE_INDEPENDENT
A secondary display that is completely independent from the primary Windows VGA display.

Note An independent display has limited support in Sapera. Most capabilities and parameters are not available and certain functions may not be implemented.

CORDISPLAY_PRM_WIDTH

Description Width of the display surface.

Type UINT32

Values Cannot be less than CORDISPLAY_CAP_WIDTH_MIN, or greater than CORDISPLAY_CAP_WIDTH_MAX.

Note CORDISPLAY_PRM_WIDTH is a read-only parameter on the system display.

CORDISPLAY_PRM_ZOOM_HORZ

Description Horizontal width in which to zoom the source display surface.

Type UINT32

Values Cannot be less than CORDISPLAY_CAP_WIDTH_MIN or greater than CORDISPLAY_CAP_WIDTH_MAX.

Note Display zoom applies to the whole display surface and not particular views. CORDISPLAY_PRM_ZOOM_HORZ is a read-only parameter on the system display.

CORDISPLAY_PRM_ZOOM_VERT

Description Vertical height in which to zoom the source display surface.

Type UINT32

Values Cannot be less than CORDISPLAY_CAP_ZOOM_VERT_MIN or greater than CORDISPLAY_CAP_ZOOM_VERT_MAX.

Note Display zoom applies to the whole display surface and not particular views. CORDISPLAY_PRM_ZOOM_VERT is a read-only parameter on the system display.

Functions

Function	Description
CorDisplayGetCap	*Gets display capability value from a display device*
CorDisplayGetCount	*Gets the number of display devices on a server*
CorDisplayGetDC	*Gets the Windows device context corresponding to the entire screen*
CorDisplayGetHandle	*Gets an handle to a display device*
CorDisplayGetPrm	*Gets display parameter value from a display device*
CorDisplayRelease	*Releases handle to a display device*
CorDisplayReleaseDC	*Releases the Windows device context corresponding to the entire screen*
CorDisplayRelease	*Resets a display device*
CorDisplayResetModule	*Resets the resources associated with the server's display device(s)*
CorDisplaySetMode	*Programs new display parameters*
CorDisplaySetPrm	*Sets a simple display parameter of a display device*
CorDisplaySetPrmEx	*Sets a complex display parameter of a display device*

CorDisplayGetCap

Get display capability value from a display device

Prototype	CORSTATUS **CorDisplayGetCap**(CORDISPLAY hD*isplay*, UINT32 *cap*, void **value*);	
Description	Gets a display capability value from a display device.	
Input	*hDisplay*	Display resource handle
	cap	Display device capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_CAP_INVALID CORSTATUS_CAP_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE	

CorDisplayGetCount

Get the number of display devices on a server

Prototype	CORSTATUS **CorDisplayGetCount**(CORSERVER hS*erver*, UINT32 **count*);	
Description	Gets the number of display devices on a server.	
Input	*hServer*	Server handle
Output	*count*	Number of display devices
Return Value	CORSTATUS_ARG_NULL (if *count* is NULL) CORSTATUS_INVALID_HANDLE	
Note	The content of *count* is 0 when there is no display device available.	

CorDisplayGetDC

Get Windows device context for a display device

Prototype	CORSTATUS **CorDisplayGetDC**(CORDISPLAY *hDisplay*, void **pDC*);	
Description	Gets the Windows device context corresponding to the entire screen for the specified display device	
Input	*hDisplay*	Display resource handle
Output	*pDC*	The device context
Return Value	CORSTATUS_ARG_NULL (if *pDC* is NULL) CORSTATUS_INVALID_HANDLE	

CorDisplayGetHandle

Get an handle to a display device

Prototype	CORSTATUS **CorDisplayGetHandle**(CORSERVER *hServer*, UINT32 *index*, CORDISPLAY *hD*isplay*);	
Description	Gets an handle to a display device.	
Input	*hServer*	Server handle
	index	Specifies which display device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorDisplayGetCount.
Output	*hDisplay*	Display resource handle
Return Value	CORSTATUS_ARG_NULL (if *hDisplay* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_RESOURCE_IN_USE	
See Also	CorDisplayGetCount and CorDisplayRelease	

CorDisplayGetPrm

Get display parameter value from a display device

Prototype	CORSTATUS **CorDisplayGetParam**(CORDISPLAY *hDisplay*, UINT32 *prm*, void **value*);	
Description	Gets parameter value from a display device.	
Input	*hDisplay*	Display resource handle
	prm	Display parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE	
See Also	CorDisplaySetPrm	

CorDisplayRelease

Release handle to a display device

Prototype	CORSTATUS **CorDisplayRelease**(CORDISPLAY *hDisplay*);
Description	Releases handle to display device
Input	*hDisplay* Display resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorDisplayGetHandle

CorDisplayReleaseDC

Release Windows device context for a display device

Prototype	CORSTATUS **CorDisplayReleaseDC**(CORDISPLAY *hDisplay*);
Description	Releases the Windows device context corresponding to the entire screen for the specified display device
Input	*hDisplay* Display resource handle
Return Value	CORSTATUS_INVALID_HANDLE

CorDisplayReset

Reset a display device

Prototype	CORSTATUS **CorDisplayReset**(CORDISPLAY *hDisplay*);
Description	Resets a display device. Restores the default values of display parameters of the specified display device.
Input	*hDisplay* Display resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorDisplayRelease

CorDisplayResetModule

Reset the resources associated with the server's display device(s)

Prototype	CORSTATUS **CorDisplayResetModule**(CORSERVER *hServer*);
Description	Resets the resources associated with the server's display device(s). Releases all resources (handle, memory) currently allocated. Make certain no other application is currently using any display device resource. This function should be used with caution.
Input	*hServer* Server handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorDisplaySetMode

Set the new display mode

Prototype	CORSTATUS **CorDisplaySetMode**(CORSERVER *hServer*, UINT32 *index*, UINT32 *width*, UINT32 *height*, UINT32 *format*, UINT32 *refresh*);	
Description	Program new display parameters: width, height, pixel depth, and refresh rate	
Input	*hServer*	Server handle
	index	Specifies which display device to program. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorDisplayGetCount.
	width	New width in pixels
	height	New height in lines
	format	Sapera format corresponding to the new pixel depth. See CorBufferNew for a list of valid values.
	refresh	New refresh rate in Hz
Return Value	CORSTATUS_NOT_AVAILABLE CORSTATUS_RESOURCE_IN_USE	
Notes	Only use this function before calling CorDisplayGetHandle.	
	The system display cannot be reprogrammed (which normally shows the Windows desktop).	
	This function also fails if the specified mode is not supported by the display hardware.	
	Depending on the desired pixel depth, use one of the following formats: for 8 bits, use CORBUFFER_VAL_FORMAT_MONO8 for 16 bits, use CORBUFFER_VAL_FORMAT_RGB565 for 32 bits, use CORBUFFER_VAL_FORMAT_RGB8888	
See Also	CorDisplayGetCount and CorDisplayGetHandle	

CorDisplaySetPrm

Set a simple display parameter of a display device

Prototype	CORSTATUS **CorDisplaySetPrm**(CORDISPLAY *hDisplay*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple display parameter of a display device	
Input	*hDisplay*	Display resource handle
	prm	Display parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_CAP_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32.	

Prototype	CORSTATUS **CorDisplaySetPrm**(CORDISPLAY *hDisplay*, UINT32 *prm*, UINT32 *value*); If the parameter is complex, use CorDisplaySetPrmEx.
See Also	CorDisplayGetPrm and CorDisplaySetPrmEx

CorDisplaySetPrmEx

Set a complex display parameter of a display device

Prototype	CORSTATUS **CorDisplaySetPrmEx**(CORDISPLAY *hDisplay*, UINT32 *prm*, const void **value*);	
Description	Sets a complex display parameter of a display device	
Input	*hDisplay*	Display resource handle
	prm	Display parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_CAP_NOT_AVAILABLE CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, use either CorDisplaySetPrm or CorDisplaySetPrmEx.	
See Also	CorDisplayGetPrm and CorDisplaySetPrm	

File Module

The File Module functions create, read, write, load, and save files of a given size and format.

Parameters

ID	Parameter	Attribute
0x00	CORFILE_PRM_FORMAT	Read Only
0x01	CORFILE_PRM_DATAFORMAT	Read Only
0x02	CORFILE_PRM_DATASIZE	Read Only
0x03	CORFILE_PRM_DATADEPTH	Read Only
0x04	CORFILE_PRM_XMIN	Read Only
0x05	CORFILE_PRM_YMIN	Read Only
0x06	CORFILE_PRM_WIDTH	Read Only
0x07	CORFILE_PRM_HEIGHT	Read Only
0x08	CORFILE_PRM_MEM_WIDTH	Read Only
0x09	CORFILE_PRM_MEM_HEIGHT	Read Only
0x0a	CORFILE_PRM_SIGNED	Read Only
0x0b	CORFILE_PRM_COMPRESSION	Read Only
0x0c	CORFILE_PRM_LUT	Read/Write
0x0d	CORFILE_PRM_SIZE	Read/Write
0xoe	CORFILE_PRM_NUM_FRAMES	Read Only
0x0f	CORFILE_PRM_FRAMERATE	Read Only
0x010	CORFILE_PRM_NAME	Read Only
0x011	CORFILE_PRM_ACCESS	Read Only

CORFILE_PRM_ACCESS

Description File access. Determines the type of reading or writing operations permitted.

Type UINT32

Values
CORFILE__VAL_ACCESS_RDONLY
CORFILE__VAL_ACCESS_RDWR
CORFILE__VAL_ACCESS_WRONLY

CORFILE_PRM_COMPRESSION

Description File compression type

Type UINT32

Values
CORFILE_VAL_COMPRESSION_NONE
CORFILE_VAL_COMPRESSION_RLE

CORFILE_VAL_COMPRESSION_LZW
CORFILE_VAL_COMPRESSION_JPEG
CORFILE_VAL_COMPRESSION_JPEG_2000
CORFILE_VAL_COMPRESSION_I263 (Intel H.263)
CORFILE_VAL_COMPRESSION_CVID (Radius Cinepack codec)
CORFILE_VAL_COMPRESSION_IV32 (Intel Indeo 3.2)
CORFILE_VAL_COMPRESSION_MSVC (Microsoft Video 1)
CORFILE_VAL_COMPRESSION_IV50 (Intel Indeo 5.0)
CORFILE_VAL_COMPRESSION_UNKNOWN

CORFILE_PRM_DATADEPTH

Description Number of bits per File element.

Type UINT32

CORFILE_PRM_DATAFORMAT

Description File data format. Determines the buffer format corresponding to a single file element's type.

Type UINT32

Values For a detailed description of the values, see Data Formats

CORFILE_PRM_DATASIZE

Description Size of one File element (in bytes).

Type UINT32

Values This value will depend on the File data format.

CORFILE_PRM_FORMAT

Description File format.

Type UINT32

Values

Windows bitmap files :	CORFILE_VAL_FORMAT_BMP
TIFF files:	CORFILE_VAL_FORMAT_TIF CORFILE_VAL_FORMAT_TIFF
DALSA Coreco raw data files:	CORFILE_VAL_FORMAT_CRC
Raw data files:	CORFILE_VAL_FORMAT_RAW
JPEG files:	CORFILE_VAL_FORMAT_JPG CORFILE_VAL_FORMAT_JPEG
JPEG 2000 files:	CORFILE_VAL_FORMAT_JPEG_2000
AVI files:	CORFILE_VAL_FORMAT_AVI
Others:	CORFILE_VAL_FORMAT_UNKNOWN

CORFILE_PRM_FRAMERATE

Description Frame rate (number of images per second) of a specified image sequence.

Type FLOAT

CORFILE_PRM_HEIGHT

Description Height of the File's region of interest (ROI).

Type UINT32

Values Range within [0...MEM_HEIGHT].

CORFILE_PRM_MEM_HEIGHT

Description File's height. Determines the number of rows in the File.

Type UINT32

CORFILE_PRM_LUT

Description File Lookup table if any

Type PUINT8

Values Array of 256, 3x8 bit, RGB values (CORDATA_FORMAT_RGB888)

CORFILE_PRM_MEM_WIDTH

Description File's width. Determines the number of columns in the File.

Type UINT32

CORFILE_PRM_NAME

Description The File name.

Type CHAR[128]

Values Null-terminated array of characters with a fixed size of 128 bytes.

CORFILE_PRM_NUM_FRAMES

Description Determines the number of image buffers in the File.

Type UINT32

CORFILE_PRM_SIGNED

Description Sign of the File elements.

Type UINT32

Values CORDATA_FORMAT_UNSIGNED
CORDATA_FORMAT_SIGNED

CORFILE_PRM_SIZE

Description File's size. Determines the size in bytes of the specified File.

Type UINT32

CORFILE_PRM_WIDTH

Description Width of the File's region of interest (ROI).

Type UINT32

Values Range within [0...MEM_WIDTH].

CORFILE_PRM_XMIN

Description Origin of the File's region of interest (ROI) along the X axis.

Type UINT32

Values Range within [0...MEM_WIDTH-1].

CORFILE_PRM_YMIN

Description Origin of the File's region of interest (ROI) along the Y axis.

Type UINT32

Values Range within [0...MEM_HEIGHT-1].

Functions

Function	Description
CorFileAddSequence	*Adds a sequence of image buffers to a File*
CorFileCopy	*Copies the content of a File resource into another File resource*
CorFileFree	*Frees handle to a File resource*
CorFileGetPrm	*Gets File parameter value from a File resource*
CorFileLoad	*Loads an image from a File into a Buffer resource*
CorFileLoadSequence	*Loads a sequence of images from a file into separate buffers*
CorFileNew	*Creates in a specified server's memory a new File resource*
CorFileRead	*Reads a series of elements from a File resource*
CorFileReadEx	*Reads a series of elements from a File resource*
CorFileSave	*Saves to a File the contents of a Buffer resource*
CorFileSaveSequence	*Saves a sequence of image buffers to a File*
CorFileSeek	*Moves file pointer to a specified location*
CorFileSetPrm	*Sets a simple File parameter of a File resource*
CorFileSetPrmEx	*Sets a complex File parameter of a File resource*
CorFileWrite	*Writes a series of elements into a File resource*

CorFileAddSequence

Add a sequence of image buffers to a File

Prototype CORSTATUS **CorFileAddSequence**(CORFILE *hFile*, CORHANDLE **hSrc*, UINT32 *nFrames*, FLOAT *frameRate*, PCSTR *options*);

Description Add to a File the content of a sequence of image Buffers.

Input

hFile	File resource handle
hSrc	Array of Buffer resource handles
nFrames	Number of image buffers within the sequence
frameRate	Playback rate in number of images per second for the sequence.
options	Argument-parsing string (may be set to NULL for automatic detection of the file format). The case insensitive switches available are listed below and may be used to form the expression: **-format** [value] or **-f** [value] The file format value supported is: **avi** or **corfile_format_avi**. example: -f CORFILE_VAL_FORMAT_AVI **-compression** [value] or **-c** [value] The codec supported for compression is: none or corfile_val_compression_none (No compression)

Output None

Return Value	CORSTATUS_FILE_OPTIONS_ERROR CORSTATUS_FILE_WRITE_ERROR CORSTATUS_FILE_OPEN_ERROR CORSTATUS_FILE_CREATE_ERROR CORSTATUS_FILE_READ_ERROR CORSTATUS_INCOMPATIBLE_FORMAT CORSTATUS_FILE_FORMAT_UNKNOWN CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY
Note	The only format currently supported is AVI (Audio Video Interleave). The buffers within the sequence must be the same data format as the images stored in the AVI file. The File should be opened in read/write mode. For the list of input buffer formats supported by CorFileAddSequence see section Buffer file formats.
See Also	CorFileLoadSequence, CorFileSaveSequence and CorFileGetPrm

CorFileCopy

Copies contents of a File resource into another File resource

Prototype	CORSTATUS **CorFileCopy**(CORFILE *hSrc*, CORFILE *hDst*);	
Description	Copies the contents of the source file into the destination buffer.	
Input	*hSrc*	File resource handle (source)
	hDst	File resource handle (destination)
Output	None	
Return Value	CORSTATUS_ARG_INVALID CORSTATUS_ARG_OUT_OF_RANGE CORSTATUS_INVALID_HANDLE CORSTATUS_INVALID_HANDLE CORSTATUS_INVALID_HANDLE CORSTATUS_INVALID_HANDLE	
Note	The source and destination files may be located on different servers	

CorFileFree

Frees handle to a File resource

Prototype	CORSTATUS **CorFileFree**(CORFILE *hFile*);	
Description	Frees handle to a File resource.	
Input	*hFile*	File resource handle
Output	None	
Return Value	CORSTATUS_FILE_CLOSE_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_AVAILABLE	

CorFileGetPrm

Gets File parameter value from a File resource

Prototype	CORSTATUS **CorFileGetPrm**(CORFILE *hFile*, UINT32 *prm*, void **value*);	
Description	Gets File parameter value from a File resource.	
Input	*hFile*	File resource handle
	prm	File parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_FILE_FORMAT_UNKNOWN CORSTATUS_FILE_WRITE_ONLY CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE	
See Also	CorFileSetPrm and CorFileSetPrmEx	

CorFileLoad

Loads an image from a file into a Buffer resource

Prototype	CORSTATUS **CorFileLoad**(CORFILE *hFile*, CORHANDLE *hDst*, PCSTR options);	
Description	Load an image from a File into a Buffer resource. Currently, the following file formats are supported: TIFF Tag Image File Format. BMP Microsoft Windows Bitmap Format. CRC DALSA raw data file format. RAW Raw data file format. JPEG Jpeg file format. For a detailed description of some of these file formats (BMP, CRC, and RAW), refer to the section “Buffer File Formats” of the *Appendix B: File Formats*.	
Input	*hFile*	File resource handle
	hDst	Buffer resource handle

options Argument-parsing string (may be set to NULL for automatic detection of the file format). The case insensitive switches available are listed below and may be used to form the argument string:

-format [value] or **-f** [value] - for file format.
The file format value supported is one of the following:
raw or **corfile_format_raw -** for raw data files.
Bmp or **corfile_format_bmp -** for windows bitmap files.
tif, tiff, corfile_format_tif or **corfile_format_tiff -** for TIF files.
jpg, jpeg, corfile_format_jpg or **corfile_format_jpeg -** for JPEG files.
jp2, jpeg_2000, or **corfile_format_jpg_2000** – for JPEG 2000 files.
crc or **corfile_format_crc -** for DALSA raw data files.
auto or **corfile_format_unknown** - for automatic detection of image format.
example: -format AUTO

The following switches are required only for raw data files:
-width [value] or **-w** [value], example: -width 512

-height [value] or **-h** [value], example: -h 512

-offset [value] or **-o** [value], example: -offset 64

The following switch is used for JPEG 2000 files only:
-component [value] or **–cmp** [value]
That switch determines which component (red, green, or blue) of a color JPEG 2000 file will be loaded when *hDst* is monochrome. Value is 0 for red, 1 for green, and 2 for blue.

Output None

Return Value
CORSTATUS_ARG_INVALID_VALUE
CORSTATUS_FILE_OPEN_ERROR
CORSTATUS_FILE_OPTIONS_ERROR
CORSTATUS_FILE_FORMAT_UNKNOWN
CORSTATUS_FILE_READ_ERROR
CORSTATUS_FILE_WRITE_ONLY
CORSTATUS_INCOMPATIBLE_FORMAT
CORSTATUS_INVALID_HANDLE

Note If the Buffer isn't large enough, the image is clipped to Buffer's size. The Buffer must be the same format as the image stored in the file (use CorFileGetPrm to get the file data format).

See Also CorFileSave and CorFileGetPrm

CorFileLoadSequence

Loads a sequence of images from a File into separate Buffers

Prototype CORSTATUS **CorFileLoadSequence**(CORFILE *hFile*, CORHANDLE **hDst*, UINT32 *nFrames*, UINT32 *startFrame*, PCSTR *options*);

Description Loads a sequence of images from a File into separate Buffers.

Input *hFile* File resource handle

hDst Array of Buffer resource handles

	nFrames	Number of Buffers within the array
	startFrame	Index of the first image of the sequence to load
	options	See CorFileAddSequence
Output	None	
Return Value	CORSTATUS_FILE_OPTIONS_ERROR CORSTATUS_FILE_READ_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_INCOMPATIBLE_FORMAT CORSTATUS_FILE_FORMAT_UNKNOWN	
Note	If a specified Buffer is not large enough, the corresponding image is clipped to the Buffer's size. The Buffers must be the same format as the images stored in the file (use CorFileGetPrm to get the file data format).	
See Also	CorFileAddSequence and CorFileSaveSequence	

CorFileNew

Create in a specified server's memory a new File resource

Prototype	CORSTATUS **CorFileNew**(CORSERVER *hServer*, const char *filename*[], const char *mode*[], CORFILE **hFile*);	
Description	Creates in a specified server's memory a new File resource.	
Input	*hServer*	Server handle
	filename	File name
	mode	Specified open mode, as follows: "r", "rb" for read only files (the file must exist). "w", "wb" for write only files (if the file exists, its content is destroyed). "r+" for both reading and writing (the file must exist).
Output	*hFile*	File resource handle
Return Value	CORSTATUS_ARG_NULL (if *hFile* or *filename* or *mode* is NULL) CORSTATUS_FILE_CREATE_ERROR CORSTATUS_FILE_OPEN_ERROR CORSTATUS_FILE_OPEN_MODE_INVALID CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY	
See Also	CorFileFree	

CorFileRead

Read a series of elements from a File resource

Prototype	CORSTATUS **CorFileRead**(CORFILE *hFILE*, void **array*, UINT32 *nItem*, UINT32 *itemSize*);	
Description	Reads a consecutive series of elements from the specified File resource and copies them into a one-dimensional destination array.	
Input	*hFile*	File resource handle

	nItem	Number of elements to read
	itemSize	Size of an element (bytes)
Output	*array*	Array to accommodate the requested number of elements (*nItem*×*itemSize*)
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_FILE_READ_ERROR CORSTATUS_FILE_WRITE_ONLY CORSTATUS_INVALID_HANDLE	
See Also	CorFileReadEx and CorFileWrite	

CorFileReadEx

Reads a series of elements from a File resource

Prototype	CORSTATUS **CorFileReadEx**(CORFILE *hFILE*, void **array*, UINT32 *nItem*, UINT32 *itemSize*, UINT32 **nRead*);	
Description	Reads a consecutive series of elements from the specified File resource and copies them into a one-dimensional destination array.	
Input	*hFile*	File resource handle
	nItem	Number of elements to read
	itemSize	Size of an element (bytes)
Output	*array*	Array to accommodate the requested number of elements (*nItem*×*itemSize*)
	nRead	Number of bytes read
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_FILE_READ_ERROR CORSTATUS_FILE_WRITE_ONLY CORSTATUS_INVALID_HANDLE	
See Also	CorFileRead and CorFileWrite	

CorFileSave

Save to a File the content of a Buffer resource

Prototype	CORSTATUS **CorFileSave**(CORFILE *hFile*, CORHANDLE *hSrc*, PCSTR *options*);	
Description	Saves to a File the content of a Buffer resource. Current supported file formats are BMP, TIFF, JPEG, raw DALSA file and RAW data files. For the list of input buffer formats supported by CorFileSave see section Buffer file formats.	
Input	*hFile*	File resource handle
	hSrc	Buffer resource handle
	options	Argument-parsing string. The case insensitive switches available are listed below and may be used to form the expression: **-format** [value] or **-f** [value] The file format value supported is one of the following: **raw** or **corfile_format_raw**.

bmp or **corfile_format_bmp**.
tif, tiff, corfile_format_tif or **corfile_format_tiff**.
crc or **corfile_format_crc**.
jpg, jpeg, corfile_format_jpg or **corfile_format_jpeg**.
jp2, jpeg_2000 or **corfile_format_jpg**.
example: -f tif

-quality [value] or **-q** [value]
This option is used to set the jpeg compression level. The quality value is an integer within [1, 100] interval. example: -q 90. When saving a JPEG 2000 file, if the quality value is 100, a lossless compression method is used. When quality is not specified, default value is 0 to minimize file size.

-compression [value] or **–c** [value]
This options is used to set the TIFF compression algorithm.
The following values are supported:
none or **corfile_val_compression_none**
rle or **corfile_val_compression_rle**
lzw or **corfile_val_compression_lzw**
jpg, **jpeg** or **corfile_val_compression_jpg**

It should be noted that the Lempel-Ziv-Welch algorithm (used when value **lzw** is set) has been enabled in Sapera only once the related patents have expired.

Output None

Return Value
CORSTATUS_ARG_NULL (if *options* is NULL)
CORSTATUS_FILE_OPTIONS_ERROR
CORSTATUS_FILE_WRITE_ERROR
CORSTATUS_INVALID_HANDLE
CORSTATUS_NO_MEMORY
CORSTATUS_INCOMPATIBLE_FORMAT
CORSTATUS_FILE_FORMAT_UNKNOWN

See Also CorFileLoad

CorFileSaveSequence

Saves a sequence of image buffers to a File

Prototype CORSTATUS **CorFileSaveSequence**(CORFILE *hFile*, CORHANDLE **hSrc*, UINT32 *nFrames*, FLOAT *frameRate*, PCSTR *options*);

Description Saves a sequence of image buffers to a File.

Input

hFile	File resource handle
hSrc	Array of Buffer resource handles
nFrames	Number of Buffers within the array
options	See **CorFileAddSequence**

Output None

Return Value
CORSTATUS_ARG_NULL (if *options* is NULL)
CORSTATUS_FILE_OPTIONS_ERROR

	CORSTATUS_FILE_WRITE_ERROR CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_INCOMPATIBLE_FORMAT CORSTATUS_FILE_FORMAT_UNKNOWNCORSTATUS_FORMAT_UNKNOWN
Note	For the list of input buffer formats supported by CorFileSaveSequence see section Buffer Data Formats Supported as Input by FileSave Functions.
See Also	CorFileAddSequence and CorFileLoadSequence

CorFileSeek

Move file pointer to a specified location

Prototype	CORSTATUS **CorFileSeek**(CORFILE *hFile*, INT32 *offset*, INT32 *origin*);	
Description	Move file pointer to a specified location	
Input	*hFile*	File resource handle
	offset	Number of bytes from origin
	origin	Initial position Beginning of file: CORFILE_BEGIN Current position: CORFILE_CURRENT End of file: CORFILE_END
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_FILE_SEEK_ERROR CORSTATUS_INVALID_HANDLE	

CorFileSetPrm

Set a simple File parameter of a File resource

Prototype	CORSTATUS **CorFileSetPrm**(CORFILE *hFile*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple File parameter of a File resource	
Input	*hFile*	File resource handle
	prm	File parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_FILE_READ_ONLY CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32. If the parameter is complex, use CorFileSetPrmEx.	
See Also	CorFileGetPrm and CorFileSetPrmEx	

CorFileSetPrmEx

Set a complex File parameter of a File resource

Prototype	CORSTATUS **CorFileSetPrmEx**(CORFILE *hFile*, UINT32 *prm*, const void **value*);	
Description	Sets a complex File parameter of a File resource.	
Input	*hFile*	File resource handle
	prm	File parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_FILE_FORMAT_UNKNOWN CORSTATUS_FILE_READ_ONLY CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, use either CorFileSetPrm or CorFileSetPrmEx. Only the LUT parameter (CORFILE_PRM_LUT) can be set in a File resource, the other parameters are Read Only.	
See Also	CorFileGetPrm and CorFileSetPrmEx	

CorFileWrite

Write a series of elements to a File resource

Prototype	CORSTATUS **CorFileWrite**(CORFILE *hFile*, const void **array*, UINT32 *nItem*, UINT32 *itemSize*);	
Description	Writes a series of elements from an one-dimensional source array to a File resource.	
Input	*hFile*	File resource handle
	array	Array which contains the elements to be written
	nItem	Number of elements to write
	itemSize	Size of an element in bytes
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL) CORSTATUS_FILE_READ_ONLY CORSTATUS_FILE_WRITE_ERROR CORSTATUS_INVALID_HANDLE	
See Also	CorFileRead	

Graphic Module

This module performs graphic operations on buffer resources.

Capabilities

ID	Capability
0x00	CORGRAPHIC_CAP_FILL
0x01	CORGRAPHIC_CAP_TEXT

CORGRAPHIC_CAP_FILL

Description Specifies whether the graphic device supports area filling.

Type UINT32

Values TRUE, The graphic device supports area filling.
FALSE, The graphic device does not support area filling.

CORGRAPHIC_CAP_TEXT

Description Specifies whether the graphic device supports text drawing.

Type UINT32

Values TRUE, The graphic device supports text drawing.
FALSE, The graphic device does not support text drawing.

Parameters

ID	Parameter	Attribute
0x00	CORGRAPHIC_PRM_OPM	Read/Write
0x01	*Not defined*	
0x02	CORGRAPHIC_PRM_BKCOLOR	Read/Write
0x03	CORGRAPHIC_PRM_COLOR	Read/Write
0x04	CORGRAPHIC_PRM_FONTSCALE	Read/Write
0x05	CORGRAPHIC_PRM_FONTNAME	Read/Write
0x06	CORGRAPHIC_PRM_LABEL	Read Only
0x07	CORGRAPHIC_PRM_TEXTALIGN	Read/Write
0x08	CORGRAPHIC_PRM_CLIP_ENABLE	Read/Write

CORGRAPHIC_PRM_OPM

Description Operation mode

Type UINT32

Values	CORGRAPHIC_VAL_OPM_REP, destination= CORGRAPHIC_PRM_COLOR CORGRAPHIC_VAL_OPM_XOR, destination= source ^ CORGRAPHIC_PRM_COLOR CORGRAPHIC_VAL_OPM_AND, destination= source & CORGRAPHIC_PRM_COLOR CORGRAPHIC_VAL_OPM_OR, destination= source | CORGRAPHIC_PRM_COLOR CORGRAPHIC_VAL_OPM_T, When enabled, a pixel operation that yields a 0 value is considered transparent, and will not overwrite the current value of the destination pixel.
Note	CORGRAPHIC_VAL_OPM_T can be ORed with one of the other operation mode.

CORGRAPHIC_PRM_BKCOLOR

Description	Background color (monochrome or color value)
Type	*CORDATA*
Note	See Data Types for *CORDATA* definition.

CORGRAPHIC_PRM_CLIP_ENABLE

Description	Enables or disables clipping.
Type	*CORDATA*
Values	TRUE, Enable FALSE, Disable
Note	Default value is FALSE

CORGRAPHIC_PRM_COLOR

Description	Foreground color (monochrome or color value)
Type	*CORDATA*
Note	See Data Types for *CORDATA* definition.

CORGRAPHIC_PRM_FONTSCALE

Description	Font scaling factor
Type	UINT32
Note	Text can only be up-scaled using this parameter.

CORGRAPHIC_PRM_FONTNAME

Description	Graphic font file name
Type	CHAR[128]
Values	Zero-terminated array of characters specifying the path and filename of the font file to be used when drawing text.
Note	See DALSA Font Generator utility program in the *Sapera LT User's Manual*.
See Also	**Graphic Font File Format**.

CORGRAPHIC_PRM_LABEL

Description	The graphic device's string ID.
Type	CHAR[128]
Values	Zero-terminated array of characters width a fixed size of 128 bytes.
Note	CORGRAPHIC_PRM_LABEL is a read-only parameter.

CORGRAPHIC_PRM_TEXTALIGN

Description	Horizontal alignment of text
Type	UINT32
Values	CORGRAPHIC_VAL_TEXTALIGN_L, Align text with the left margin CORGRAPHIC_VAL_TEXTALIGN_C, Center text CORGRAPHIC_VAL_TEXTALIGN_R, Align text with the right margin
Note	The default text alignment mode is CORGRAPHIC_VAL_TEXTALIGN_L

Functions

Function	Description
CorGraphicArc	*Draws an arc in a buffer resource using a graphic device*
CorGraphicCircle	*Draws a circle in a buffer resource using a graphic device*
CorGraphicClear	*Clears an area of a buffer resource using a graphic device*
CorGraphicDot	*Draws a dot in a buffer resource using a graphic device*
CorGraphicDots	*Draws a series of dots in a buffer resource using a graphic device*
CorGraphicDrawVector	*Draws a vector in a buffer resource using a graphic device*
CorGraphicEllipse	*Draws an ellipse in a buffer resource using a graphic device*
CorGraphicFill	*Fills an enclosed area in a buffer resource using a graphic device*
CorGraphicGetCap	*Gets capability value from a graphic device*
CorGraphicGetCount	*Gets the number of graphic devices on a server*
CorGraphicGetHandle	*Gets a handle to a graphic device*
CorGraphicGetPrm	*Gets parameter value from a graphic device*
CorGraphicGrid	*Draws a grid in a buffer resource using a graphic device*
CorGraphicLine	*Draws a line in a buffer resource using a graphic device*
CorGraphicRect	*Draws a rectangle in a buffer resource using a graphic device*
CorGraphicRelease	*Releases handle to a graphic device*
CorGraphicReset	*Resets a graphic device*
CorGraphicResetModule	*Reset resources associated with the server graphic device*
CorGraphicSetFont	*Sets the font to be used by a graphic device*
CorGraphicSetPrm	*Sets a simple graphic parameter of a graphic device*

CorGraphicSetPrmEx	*Sets a complex graphic parameter of a graphic device*
CorGraphicTarget	*Draws a crosshair in a buffer resource using a graphic device*
CorGraphicText	*Draws text in a buffer resource using a graphic device*
CorGraphicTextEx	*Draws text in a buffer resource at any angle using a graphic device*

CorGraphicArc

Draw an arc in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicArc**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *xRadius*, UINT32 *yRadius*, UINT32 *startAngle*, UINT32 *endAngle*, BOOLEAN *fill*);	
Description	Draws an arc in a specified buffer resource using a graphic device. The arc is actually a segment of an ellipse described by the parameters *xRadius* and *yRadius*.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	X-coordinate of ellipse's origin
	y	Y-coordinate of ellipse's origin
	xRadius	Horizontal radius of ellipse
	yRadius	Vertical radius of ellipse
	startAngle	Angle of ellipse that will define the arc's starting point
	endAngle	Angle of ellipse that will define the arc's ending point
	fill	Arc is filled if *fill* has a value of TRUE
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicCircle

Draw a circle in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicCircle**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *radius*, BOOLEAN *fill*);	
Description	Draws a circle in a specified buffer resource.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	X-coordinate of circle's origin
	y	Y-coordinate of circle's origin
	radius	Circle radius
	fill	Circle is filled if *fill* has a value of TRUE
Output	None	

Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_ARG_INVALID and CORSTATUS_ARG_OUT_OF_RANGE

CorGraphicClear

Clear an area of a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicClear**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*);	
Description	Clears a rectangular area in a specified buffer resource using a graphic device. The rectangular area is defined by giving the coordinates for a starting corner and an ending corner.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x1	X-coordinate of the starting corner
	y1	Y-coordinate of the starting corner
	x2	X-coordinate of the ending corner
	y2	Y-coordinate of the ending corner
Output	None	
Return Value	CORSTATUS_ARG_INCOMPATIBLE, CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicDot

Draw a dot in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicDot**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*);	
Description	Draws a dot in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	Dot's x-coordinate
	y	Dot's y-coordinate
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicDots

Draw a series of dots in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicDots**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, CORBUFFER *hDots*, UINT32 *nPixels*);	
Description	Draws a series of dots in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle

	x	First dot's X-coordinate
	y	First dot's Y-coordinate
	hDots	Series of bytes defining the path to follow. Each byte represents the direction to the next adjacent pixel to draw, as indicated in the table below:

Value	1	2	3	4	5	6	7	8
Direction	E	NE	N	NW	W	SW	S	SE

A value of zero or any value greater than 8 is interpreted as not moving, causing the previous element to be drawn again.

	nPixels	Amount of dots to plot in the given direction
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicDrawVector

Draw a vector in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicDrawVector**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, CORBUFFER *vector*, INT32 *min*, INT32 *max*, UINT32 *n*);	
Description	The CorGraphicDrawVector operator accepts vectors of integer numbers, floating point numbers, or vectors of points. Point input data is plotted using the (*x*,*y*) coordinate of the data. Scalar data is plotted using the vector index (starting from 0) as the X coordinate and the element value as the Y coordinate.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	vector	Vector to draw
	min	Minimum Y value to be plotted
	max	Maximum Y value to be plotted
	n	Number of values to be plotted
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicEllipse

Draw an ellipse in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicEllipse**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *xRadius*, UINT32 *yRadius*, BOOLEAN *fill*);	
Description	Draws an ellipse in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	X-coordinate of ellipse's origin

	y	Y-coordinate of ellipse's origin
	xRadius	Horizontal radius of ellipse
	yRadius	Vertical radius of ellipse
	fill	The ellipse is filled if *fill* has a value of TRUE
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicFill

Fill an enclosed area in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicFill**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *xSeed*, UINT32 *ySeed*);	
Description	Fills an enclosed area in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	xSeed	X-coordinate of seed point
	ySeed	Y-coordinate of seed point
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE, CORSTATUS_NO_MEMORY	

CorGraphicGetCap

Gets graphic capability value from a graphic device

Prototype	CORSTATUS **CorGraphicGetCap**(CORGRAPHIC *hGraphic*, UINT32 *cap*, void **value*);	
Description	Gets a graphic capability value from a graphic device.	
Input	*hGraphic*	Graphic resource handle
	cap	Graphic device capability requested
	value	Value of the capability
Output	None	
	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_CAP_INVALID, CORSTATUS_CAP_NOT_AVAILABLE and CORSTATUS_INVALID_HANDLE	

CorGraphicGetCount

Get the number of graphic devices on a server

Prototype	CORSTATUS **CorGraphicGetCount**(CORSERVER *hServer*, UINT32 **count*);	
Description	Gets the number of graphic devices on a server.	
Input	*hServer*	Server handle

Output	*count*	Number of graphic devices
Note	The content of count is 0 when there is no graphic device available.	
Return Value	CORSTATUS_ARG_NULL (if *count* is NULL), CORSTATUS_INVALID_HANDLE	

CorGraphicGetHandle

Get a handle to a graphic device

Prototype	CORSTATUS **CorGraphicGetHandle**(CORSERVER *hServer*, UINT32 *index*, CORGRAPHIC **hGraphic*);	
Description	Gets an handle to a graphic device.	
Input	*hServer*	Server handle
	index	Specifies which graphic device to select. Valid values are in the range [0...count-1], where *count* is the value returned by CorGraphicGetCount.
Output	*hGraphic*	Graphic resource handle
Return Value	CORSTATUS_ARG_NULL (if *hGraphic* is NULL), CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
See Also	CorGraphicGetCount and CorGraphicRelease	

CorGraphicGetPrm

Get graphic parameter value from a graphic device

Prototype	CORSTATUS **CorGraphicGetPrm**(CORGRAPHIC *hGraphic*, UINT32 *prm*, void **value*);	
Description	Gets graphic parameter value from a graphic device.	
Input	*hGraphic*	Graphic resource handle
	prm	Graphic parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID	
See Also	CorGraphicSetPrm and CorGraphicSetPrmEx	

CorGraphicGrid

Draws a grid in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicGrid**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*, UINT32 *nx*, UINT32 *ny*);	
Description	Draws a grid in a buffer resource by defining a rectangular area's start and end corners, and the horizontal and vertical grid spacing.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x1	X-coordinate of the starting corner

	y1	Y-coordinate of the starting corner
	x2	X-coordinate of the ending corner
	y2	Y-coordinate of the ending corner
	nx	Horizontal grid spacing
	ny	Vertical grid spacing
Output	None	
Return Value	CORSTATUS_ARG_INCOMPATIBLE, CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicLine

Draws a line in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicLine**(CORGRAPHIC *hGraphic*, CORBUFFER *buffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*);	
Description	Draws a line in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x1	X-coordinate of the line's starting point
	y1	Y-coordinate of the line's starting point
	x2	X-coordinate of the line's ending point
	y2	Y-coordinate of the line's ending point
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicRect

Draws a rectangle in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicRect**(CORGRAPHIC *hGraphic*, CORBUFFER *buffer*, UINT32 *x1*, UINT32 *y1*, UINT32 *x2*, UINT32 *y2*, BOOLEAN *fill*);	
Description	Draws a rectangular area in a specified buffer resource using a graphic device. The rectangle is defined by giving the coordinates for a starting corner and an ending corner.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x1	X-coordinate of the starting corner
	y1	Y-coordinate of the starting corner
	x2	X-coordinate of the ending corner
	y2	Y-coordinate of the ending corner
	fill	Rectangle is filled if *fill* has a value of TRUE
Output	None	

Return Value	CORSTATUS_ARG_INCOMPATIBLE, CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE

CorGraphicRelease

Release handle to graphic device

Prototype	CORSTATUS **CorGraphicRelease**(CORGRAPHIC *hGraphic*);
Description	Releases handle to graphic device.
Input	*hGraphic* Graphic resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorGraphicGetHandle

CorGraphicReset

Reset a graphic device

Prototype	CORSTATUS **CorGraphicReset**(CORGRAPHIC *hGraphic*);
Description	Resets a graphic device. Restores the default graphic parameters of the specified device.
Input	*hGraphic* Graphic resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorGraphicResetModule

Reset the resources associated with the server's graphic devise(s)

Prototype	CORSTATUS **CorGraphicResetModule** (CORSERVER *hServer*);
Description	Resets the resources associated with the server's graphic device(s). Releases all resources (handle, memory) currently allocated. When using this function, make certain that no other application is currently using any graphic resource. This function should be used with caution.
Input	*hServer* Server handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorGraphicSetFont

Set the font to be used by a graphic device

Prototype	CORSTATUS **CorGraphicSetFont**(CORGRAPHIC *hGraphic*, const char **fontName*, const void **fontData*, UINT32 *fontDataSize*);	
Description	Sets the font to be used by a graphic device.	
Input	*hGraphic*	Graphic resource handle
	fontName	String specifying the path and filename of a binary font file to be used or the name to be attribute to the font in *fontData* array.

	fontData	Array which contains the font to be used (*fontDataSize*). Must be *NULL* when *fontName* contains the path and filename of a binary font file.
	fontDataSize	Size of *fontData* array (in bytes). Should be 0 when *fontName* contains the path and filename of a binary font file.
Output	None	
Return Value	CORSTATUS_FILE_OPEN_ERROR, CORSTATUS_FILE_READ_ERROR, CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
Note	If *fontData* is NULL, *fontName* must contains the path and filename of a binary font file.	
See Also	**Graphic Font File Format**.	

CorGraphicSetPrm

Set a simple graphic parameter of a graphic device

Prototype	CORSTATUS **CorGraphicSetPrm**(CORGRAPHIC *hGraphic*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple graphic parameter of a graphic device.	
Input	*hGraphic*	Graphic resource handle
	prm	Graphic parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID	
Note	A simple parameter fits inside an UINT32. For complex parameters use CorGraphicSetPrmEx.	
See Also	CorGraphicGetPrm	

CorGraphicSetPrmEx

Set a complex graphic parameter of a graphic device

Prototype	CORSTATUS **CorGraphicSetPrmEx**(CORGRAPHIC *hGraphic*, UINT32 *prm*, const void **value*);	
Description	Sets the value of a specified graphic parameter.	
Input	*hGraphic*	Graphic resource handle
	prm	Graphic parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, use either CorGraphicSetPrm or CorGraphicSetPrmEx.	
See Also	CorGraphicGetPrm	

CorGraphicTarget

Draws a crosshair in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicTarget**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*);	
Description	Draw a crosshair in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	Target center X coordinate
	y	Target center Y coordinate
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorGraphicText

Draw text in a buffer resource using a graphic device

Prototype	CORSTATUS **CorGraphicText**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, const char *text[]*);	
Description	Draws text in a specified buffer resource using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	X-coordinate for text origin
	y	Y-coordinate for text origin
	text	Text string to draw
Output	None	
	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_NULL (if *text* is NULL), CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID_VALUE	
See Also	CorGraphicTextEx and **Graphic Font File Format**	

CorGraphicTextEx

Draw text in a buffer at any angle using a graphic device

Prototype	CORSTATUS **CorGraphicTextEx**(CORGRAPHIC *hGraphic*, CORBUFFER *hBuffer*, UINT32 *x*, UINT32 *y*, UINT32 *angle*, const char *text[]*);	
Description	Draws text in a specified buffer at a specified angle using a graphic device.	
Input	*hGraphic*	Graphic resource handle
	hBuffer	Buffer resource handle
	x	X-coordinate for text origin
	y	Y-coordinate for text origin

	angle	Angle at which to rotate text
	text	Text string to draw
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_NULL (if *text* is NULL),CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID_VALUE	
See Also	CorGraphicText and **Graphic Font File Format**	

I/O Module

The General I/O Module is used to control a block of general inputs and outputs (I/O signal pins). A block of general inputs and outputs is a group of I/Os which can be read and/or written all at once. For a TTL level type I/O, its state is considered **on** or **active** if the measured voltage on the I/O is 5V (typical).

Definitions

Below defines the methods related to the I/O module.

CORGIO_VAL_INPUT_CONTROL_METHOD_1

Numerical Value	0x00000001 (Hardware Latch)
Description	This method latches the value of the input I/O pins when there is a signal pulse on the input control pin.

CORGIO_VAL_OUTPUT_CONTROL_METHOD_1

Numerical Value	0x00000001 (Auto latch)
Description	This method generates a pulse on the output control pin each time a new value is written to it by the user.

CORGIO_VAL_OUTPUT_CONTROL_METHOD_2

Numerical Value	0x00000001 (Manual latch)
Description	This method allows the function CorGioSetOutputControlState to control the state of the output control pin.

Capabilities

ID	Capability
0x00	CORGIO_CAP_IO_COUNT
0x01	CORGIO_CAP_DIR_INPUT
0x02	CORGIO_CAP_DIR_OUTPUT

0x03	CORGIO_CAP_DIR_TRISTATE
0x04	CORGIO_CAP_INPUT_CONTROL_METHOD
0x05	CORGIO_CAP_INPUT_CONTROL_POLARITY
0X06	CORGIO_CAP_OUTPUT_CONTROL_METHOD
0X07	CORGIO_CAP_OUTPUT_CONTROL_POLARITY
0X08	CORGIO_CAP_OUTPUT_TYPE
0X09	CORGIO_CAP_INPUT_LEVEL
0x0a	CORGIO_CAP_CONNECTOR
0x0b	CORGIO_CAP_EVENT_TYPE
0x0c	CORGIO_PRM_FAULT_DETECT
0x0d	CORGIO_CAP_POWER_GOOD

CORGIO_CAP_CONNECTOR

Description Specifies which connectors can be use for this I/O device

Type UINT32

Values CORGIO_VAL_CONNECTOR_1 (0x00000001)
CORGIO_VAL_CONNECTOR_2 (0x00000002)
If bit 'n' is 1, then the associated connector can be used.

CORGIO_CAP_DIR_INPUT

Description Specifies which I/O can be set as inputs.

Type UINT32

Values If bit 'n' is 1, then the associated I/O can be an input.

CORGIO_CAP_DIR_OUTPUT

Description Specifies which I/O can be set as outputs.

Type UINT32

Values If bit 'n' is 1, then the associated I/O can be an output.

CORGIO_CAP_DIR_TRISTATE

Description Specifies which I/O can be tri-stated.

Type UINT32

Values If bit 'n' is 1, then the associated I/O can be tri-stated.

CORGIO_CAP_EVENT_TYPE

Description Specifies the event type(s) that can be registered.

Type UINT32

Values	CORGIO_VAL_EVENT_TYPE_RISING_EDGE (0x000000001) CORGIO_VAL_EVENT_TYPE_FALLING_EDGE (0x00000002) CORGIO_VAL_EVENT_TYPE_FAULT (0x00000004)
Note	The returned value is the ORed combination of the valid values.

CORGIO_CAP_FAULT_DETECT

Description	Specifies if the I/O device has a fault detection.
Type	BOOL
Values	If TRUE, the device has a fault detection
Note	See CORGIO_PRM_FAULT_DETECT.

CORGIO_CAP_INPUT_CONTROL_METHOD

Description	Specifies which input control methods are available for this I/O device.
Type	UINT32
Values	CORGIO_VAL_INPUT_CONTROL_METHOD_OFF CORGIO_VAL_INPUT_CONTROL_METHOD_1 CORGIO_VAL_INPUT_CONTROL_METHOD_2 CORGIO_VAL_INPUT_CONTROL_METHOD_3 CORGIO_VAL_INPUT_CONTROL_METHOD_4 CORGIO_VAL_INPUT_CONTROL_METHOD_5

CORGIO_CAP_INPUT_CONTROL_POLARITY

Description	Specifies which signal polarities are available for the input control pin on this I/O device
Type	UINT32
Values	CORGIO_VAL_POLARITY_ACTIVE_LOW CORGIO_VAL_POLARITY_ACTIVE_HIGH CORGIO_VAL_POLARITY_RISING_EDGE CORGIO_VAL_POLARITY_FALLING_EDGE CORGIO_VAL_POLARITY_BOTH_EDGE CORGIO_VAL_POLARITY_DOUBLE_PULSE_RISING_EDGE CORGIO_VAL_POLARITY_DOUBLE_PULSE_FALLING_EDGE

CORGIO_CAP_INPUT_LEVEL

Description	Specifies the input levels for this I/O device.
Type	UINT32
Values	CORGIO_VAL_INPUT_LEVEL_TTL CORGIO_VAL_INPUT_LEVEL_422 CORGIO_VAL_INPUT_LEVEL_LVDS CORGIO_VAL_INPUT_LEVEL_24VOLTS
Note	The returned value is the ORed combination of the valid values.

CORGIO_CAP_IO_COUNT

Description Specifies the number of individual I/Os in the block.

Type UINT32

CORGIO_CAP_OUTPUT_CONTROL_METHOD

Description Specifies which output control methods are available for this I/O device.

Type UINT32

Values
CORGIO_VAL_OUTPUT_CONTROL_METHOD_OFF
CORGIO_VAL_OUTPUT_CONTROL_METHOD_1
CORGIO_VAL_OUTPUT_CONTROL_METHOD_2
CORGIO_VAL_OUTPUT_CONTROL_METHOD_3
CORGIO_VAL_OUTPUT_CONTROL_METHOD_4
CORGIO_VAL_OUTPUT_CONTROL_METHOD_5

CORGIO_CAP_OUTPUT_CONTROL_POLARITY

Description Specifies which signal polarities are available for the output control pin on this I/O device.

Type UINT32

Values
CORGIO_VAL_POLARITY_ACTIVE_LOW
CORGIO_VAL_POLARITY_ACTIVE_HIGH
CORGIO_VAL_POLARITY_RISING_EDGE
CORGIO_VAL_POLARITY_FALLING_EDGE
CORGIO_VAL_POLARITY_BOTH_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_RISING_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_FALLING_EDGE

CORGIO_CAP_OUTPUT_TYPE

Description Specifies the output types for this I/O device.

Type UINT32

Values
CORGIO_VAL_OUTPUT_TYPE_PNP
CORGIO_VAL_OUTPUT_TYPE_NPN

Note The returned value is the ORed combination of the valid values.

CORGIO_CAP_POWER_GOOD

Description Specifies if the I/O device has a power good output.

Type BOOL

Values
If TRUE, the device has a power good output.
This output is an open collector output.
See CORGIO_PRM_POWER_GOOD.

Parameters

ID	Parameters	Attribute
0x00	CORGIO_PRM_LABEL	Read Only
0x01	CORGIO_PRM_DEVICE_ID	Read Only
0x02	*Reserved*	
0x03	CORGIO_PRM_DIR_OUTPUT	Read/Write
0x04	CORGIO_PRM_DIR_TRISTATE	Read/Write
0x05	CORGIO_PRM_INPUT_CONTROL_METHOD	Read/Write
0x06	CORGIO_PRM_INPUT_CONTROL_POLARITY	Read/Write
0x07	CORGIO_PRM_OUTPUT_CONTROL_METHOD	Read/Write
0x08	CORGIO_PRM_OUTPUT_CONTROL_POLARITY	Read/Write
0x09	CORGIO_PRM_OUTPUT_TYPE	Read/Write
0x0a	CORGIO_PRM_INPUT_LEVEL	Read/Write
0x0b	CORGIO_PRM_CONNECTOR	Read/Write
0x0c	CORGIO_PRM_FAULT_DETECT	Read Only
0x0d	CORGIO_PRM_POWER_GOOD	Read/Write

CORGIO_PRM_CONNECTOR

Description Selects which connector is used for this I/O device

Type UINT32

Values CORGIO_VAL_CONNECTOR_1
CORGIO_VAL_CONNECTOR_2

Note This value is board specific.

CORGIO_PRM_DEVICE_ID

Description The General I/O's device ID.

Type UINT32

Note CORGIO_PRM_DEVICE_ID is a read-only parameter.

CORGIO_PRM_DIR_OUTPUT

Description Specifies which I/O are set as outputs.

Type UINT32

Values If bit 'n' is 1, then the associated I/O is an output; otherwise it is an input.

CORGIO_PRM_DIR_TRISTATE

Description Specifies which I/O are to be tri-stated.

Type UINT32

Values If bit 'n' is 1, then the associated I/O is tri-stated; otherwise it is not.

CORGIO_PRM_FAULT_DETECT

Description Use to get the fault detect status for this I/O device.

Type BOOL

Values If TRUE, a fault has been detected.

Note If TRUE, the fault needs to be corrected and the device needs to be reset.

CORGIO_PRM_INPUT_CONTROL_METHOD

Description Selects which input control method is activated for this I/O device.

Type UINT32

Values
CORGIO_VAL_INPUT_CONTROL_METHOD_OFF
CORGIO_VAL_INPUT_CONTROL_METHOD_1
CORGIO_VAL_INPUT_CONTROL_METHOD_2
CORGIO_VAL_INPUT_CONTROL_METHOD_3
CORGIO_VAL_INPUT_CONTROL_METHOD_4
CORGIO_VAL_INPUT_CONTROL_METHOD_5

CORGIO_PRM_INPUT_CONTROL_POLARITY

Description Specifies which signal polarity to use for the input control pin on this I/O device

Type UINT32

Values
CORGIO_VAL_POLARITY_ACTIVE_LOW
CORGIO_VAL_POLARITY_ACTIVE_HIGH
CORGIO_VAL_POLARITY_RISING_EDGE
CORGIO_VAL_POLARITY_FALLING_EDGE
CORGIO_VAL_POLARITY_BOTH_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_RISING_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_FALLING_EDGE

CORGIO_PRM_INPUT_LEVEL

Description Selects which input level is used for this I/O device.

Type UINT32

Values
CORGIO_VAL_INPUT_LEVEL_TTL
CORGIO_VAL_INPUT_LEVEL_422
CORGIO_VAL_INPUT_LEVEL_LVDS
CORGIO_VAL_INPUT_LEVEL_24VOLTS

CORGIO_PRM_LABEL

Description The General I/O's string ID.

Type CHAR[128]

Values Zero-terminated array of characters with a fixed size of 128 bytes.

Note CORGIO_PRM_LABEL is a read-only parameter.

CORGIO_PRM_OUTPUT_CONTROL_METHOD

Description Selects which output control method is activated for this I/O device

Type UINT32

Values CORGIO_VAL_OUTPUT_CONTROL_METHOD_OFF
CORGIO_VAL_OUTPUT_CONTROL_METHOD_1
CORGIO_VAL_OUTPUT_CONTROL_METHOD_2
CORGIO_VAL_OUTPUT_CONTROL_METHOD_3
CORGIO_VAL_OUTPUT_CONTROL_METHOD_4
CORGIO_VAL_OUTPUT_CONTROL_METHOD_5

CORGIO_PRM_OUTPUT_CONTROL_POLARITY

Description Specifies which signal polarity to use for the output control pin on this I/O device

Type UINT32

Values CORGIO_VAL_POLARITY_ACTIVE_LOW
CORGIO_VAL_POLARITY_ACTIVE_HIGH
CORGIO_VAL_POLARITY_RISING_EDGE
CORGIO_VAL_POLARITY_FALLING_EDGE
CORGIO_VAL_POLARITY_BOTH_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_RISING_EDGE
CORGIO_VAL_POLARITY_DOUBLE_PULSE_FALLING_EDGE

CORGIO_PRM_OUTPUT_TYPE

Description Selects which output type is activated for this I/O device.

Type UINT32

Values CORGIO_VAL_OUTPUT_TYPE_PNP
CORGIO_VAL_OUTPUT_TYPE_NPN

CORGIO_PRM_POWER_GOOD

Description Sets the power good output.

Type BOOL

Values TRUE or FALSE
See CORGIO_CAP_POWER_GOOD.

Functions

Function	Description
CorGioAutoTrigger	*Triggers an I/O at a specific time*
CorGioGetCap	*Gets a general I/O capability value*
CorGioGetCount	*Gets the number of general I/O devices on a server*
CorGioGetHandle	*Gets a handle to a general I/O device*
CorGioGetPrm	*Gets a general I/O parameter value*
CorGioGetState	*Gets the current I/O states*
CorGioRegisterCallback	*Registers a function that will be called when an input I/O generates an interrupt*
CorGioRelease	*Releases a handle to a general I/O device*
CorGioReset	*Resets a general I/O device*
CorGioResetModule	*Resets the resources associated with the server's general I/O device(s)*
CorGioSetPrm	*Sets a simple general I/O parameter*
CorGioSetPrmEx	*Sets a complex general I/O parameter*
CorGioSetOutputControlState	*Set the state of the IO ouput.*
CorGioSetState	*Sets the state of the I/Os*
CorGioUnregisterCallback	*Unregisters a callback function*

CorGioAutoTrigger

Automatically trigger the state of an I/O at a specific time

Prototype	CORSTATUS **CorGioAutoTrigger**(CORGIO *hGio*, CORCOUNTER *hCounter*, UINT32 *io*, CORCOUNT *startCount*, CORCOUNT *stopCount*, UINT32 *state);*	
Description	Automatically trigger the state of an I/O	
Input	*hGio*	General I/O resource handle
	hCounter	Counter resource handle
	io	I/O mask to automatically trigger the state. If bit 'n' is 1, then the associated I/O will be triggered.
	startCount	Count at which the I/O will be triggered.
	stopCount	Count at which the I/O will go back to its original state.
	state	State of the I/O at *startCount.*
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_NOT_IMPLEMENTED	
Notes	Upon calling this function, the specified I/O(s) will be set to the state opposite of *state.*	

CorGioGetCap

Get general I/O capability value

Prototype	CORSTATUS **CorGioGetCap**(CORGIO *hGio*, UINT32 *cap*, void **value*);	
Description	Gets general I/O capability value.	
Input	*hGio*	General I/O resource handle
	cap	General I/O device capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_CAP_INVALID and CORSTATUS_INVALID_HANDLE	

CorGioGetCount

Get the number of general I/O devices on a server

Prototype	CORSTATUS **CorGioGetCount**(CORSERVER *hServer*, UINT32 **count*);	
Description	Gets the number of general I/O devices available on a server.	
Input	*hServer*	Server handle
Output	*count*	Number of general I/O devices
Return Value	CORSTATUS_ARG_NULL (if *count* is NULL), CORSTATUS_INVALID_HANDLE	
Note	The content of *count* is 0 when there is no general I/O device available.	

CorGioGetHandle

Get a handle to a general I/O device

Prototype	CORSTATUS **CorGioGetHandle**(CORSERVER *hServer*, UINT32 *deviceId*, CORGIO **hGio*);	
Description	Gets a handle to a general I/O device.	
Input	*hServer*	Server handle
	deviceId	Specifies which general I/O device to select. Valid values are in the range [0...*count*-1], where *count* is the value returned by CORGIOGetCount.
Output	*hGio*	General I/O resource handle
Return Value	CORSTATUS_ARG_NULL (if *hGio* is NULL), CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_INVALID_HANDLE, CORSTATUS_NO_MEMORY and CORSTATUS_RESOURCE_IN_USE	
See Also	CorGioGetCount and CorGioRelease	

CorGioGetPrm

Get general I/O parameter value

Prototype	CORSTATUS **CorGioGetPrm**(CORGIO *hGio*, UINT32 *prm*, void **value*);
Description	Gets general I/O parameter value.

Input	*hGio*	General I/O resource handle
	prm	General I/O parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID	
See Also	CorGioRelease and CorGioSetPrmEx	

CorGioGetState

Get the state of the I/Os

Prototype	CORSTATUS **CorGioGetState**(CORGIO *hGio*, UINT32 **value*);	
Description	Gets the state of the I/Os.	
Input	*hGio*	General I/O resource handle
Output	*value*	Current I/O values. If a bit is '1', then the corresponding I/O is **high**; otherwise, it is **low**.
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_SERVICE_NOT_AVAILABLE	
See Also	CorGioSetPrm and CorGioSetPrmEx	

CorGioRegisterCallback

Register a function that will be called when an input I/O generates an interrupt.

Prototype	CORSTATUS **CorGioRegisterCallback**(CORGIO *hGio*, UINT32 *eventType*, UINT32 *io*, void* *callbackFct* , void **context*)	
Description	Registers a function that will be called when an input I/O generates an interrupt.	
Input	*hGio*	General I/O resource handle
	eventType	Type of event to register: CORGIO_VAL_EVENT_TYPE_RISING_EDGE CORGIO_VAL_EVENT_TYPE_FALLING_EDGE
	io	If bit 'n' is 1, then the corresponding I/O will cause *callbackFct* to be called.
	callbackFct	Callback function to be registered. Define your callback function as follows: *CORSTATUS CCONV* *callback (void *context, UINT32 eventType, UINT32 eventCount);* When called, *context* will have the value you have specified at callback function registration; *eventType* will contain the event(s) that triggered the call to your callback function; *eventCount* should increment by one at each call, with a starting value of 1. In case the counter resource cannot keep up because there is too many events to be signaled, *eventCount* will take nonconsecutive values, indicating that events have been lost. See the Data Types section for the PCORCALLBACK definition.
	context	Context pointer to be passed to the callback function when called.
Output	None	

Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_RESOURCE_IN_USE (if attempting to associate a callback function to an I/O that already has a callback) CORSTATUS_SERVICE_NOT_AVAILABLE
Note	Before returning CORSTATUS_RESOURCE_IN_USE, the function CorGioRegisterCallback will be called with *index* and *callbackFct* as parameters.
See Also	CorGioUnregisterCallback

CorGioRelease

Release handle to a general I/O device

Prototype	CORSTATUS **CorGioRelease**(CORGIO *hGio*);
Description	Releases handle to a general I/O device.
Input	*hGio* General I/O resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorGioGetHandle

CorGioReset

Reset a general I/O device

Prototype	CORSTATUS **CorGioReset**(CORGIO *hGio*);
Description	Resets a general I/O device. Restores the default values for general I/O parameters.
Input	*hGio* General I/O resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_SERVICE_NOT_AVAILABLE

CorGioResetModule

Reset the resources associated with the server's general I/O device(s)

Prototype	CORSTATUS **CorGioResetModule**(CORSERVER *hServer*);
Description	Resets the resources associated with the server's general I/O device(s). Releases all resources (handle, memory) currently allocated. Make certain that no other application is currently using any general I/O device resource. This function should be use with caution.
Input	*hServer* Server handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE

CorGioSetPrm

Set a simple general I/O parameter

Prototype	CORSTATUS **CorGioSetPrm**(CORGIO *hGio*, UINT32 *prm*, UINT32 *value*);
Description	Sets a simple general I/O parameter.

Input	*hGio*	General I/O resource handle
	prm	General I/O parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_OUT_OF_RANGE CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32. If the parameter is complex, use CorGioSetPrmEx.	
See Also	CorGioGetPrm and CorGioSetPrmEx	

CorGioSetPrmEx

Set a complex general I/O parameter

Prototype	CORSTATUS **CorGioSetPrmEx**(CORGIO *hGio*, UINT32 *prm*, void **value*);	
Description	Sets a complex general I/O parameter.	
Input	*hGio*	General I/O resource handle
	prm	General I/O parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY CORSTATUS_PRM_INVALID CORSTATUS_PRM_INVALID_VALUE CORSTATUS_PRM_MUTUALLY_EXCLUSIVE CORSTATUS_PRM_NOT_AVAILABLE CORSTATUS_PRM_OUT_OF_RANGE CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, used either CorGioSetPrm or CorGioSetPrmEx.	
See Also	CorGioGetPrm and CorGioSetPrm	

CorGioSetOutputControlState

Set the state of the IO ouput. This is a board specific function. See the board user's manual.

Prototype	CORSTATUS **CorGioSetOutputControlState**(CORGIO *hGio*,,UINT32 *ioMask*, UINT32 *value*);	
Description	Sets the state of the I/O output.	
Input	*hGio*	General I/O resource handle
	ioMask	If a bit is '1', then the corresponding Output will be affected.
	value	
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE CORSTATUS_SERVICE_NOT_AVAILABLE	
See Also	CorGioGetPrm and CorGioSetPrmEx	

CorGioSetState

Set the state of the I/Os

Prototype	CORSTATUS **CorGioSetState**(CORGIO *hGio*, UINT32 *ioMask*, UINT32 *value*);	
Description	Sets the state of the general I/O pins, if available on the hardware.	
Input	*hGio*	General I/O resource handle
	ioMask	Mask specifying the I/Os to modify. If bit 'n' is 1, then the I/O will be written with the corresponding bit in *value.*
	value	New I/O values. If a bit is '1', the corresponding I/O will be set to **high**; otherwise, it will be set to **low**. Value represents a bit-field.
Output	None	
Return Value	CORSTATUS_ARG_INCOMPATIBLE (if trying to set the state of an input pin) CORSTATUS_INVALID_HANDLE and CORSTATUS_SERVICE_NOT_AVAILABLE	
See Also	CorGioGetPrm and CorGioSetPrmEx	

CorGioUnregisterCallback

Unregister a callback function.

Prototype	CORSTATUS **CorGioUnregisterCallback**(CORGIO *hGio*, void* *callbackFct*);	
Description	Unregisters a callback function.	
Input	*hGio*	General I/O resource handle
	callbackFct	Pointer to a callback function that has been previously registered.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorGioRegisterCallback	

LUT Module

The LUT (Lookup Table) Module defines LUT data structures that are used by the Acquisition, Display, and Processing modules. Refer to the functions CorAcqSetLut and CorViewSetLut for a description on how to apply the LUTs to the Acquisition and Display hardware, respectively.

Parameters

ID	Parameter	Attribute
0x00	CORLUT_PRM_ADDRESS	Read Only
0x01	CORLUT_PRM_DATASIZE	Read Only
0x02	CORLUT_PRM_FORMAT	Read Only
0x03	CORLUT_PRM_NPAGES	Read Only
0x04	CORLUT_PRM_NENTRIES	Read Only
0x05	CORLUT_PRM_SIZE	Read Only
0x06	CORLUT_PRM_PHYSADDRESS	Read Only
0x07	CORLUT_PRM_SIGNED	Read Only

CORLUT_PRM_ADDRESS

Description Address of the buffer containing the lookup table.

Type UINT32

Values 32-bit address of the buffer.

CORLUT_PRM_DATASIZE

Description Size of one lookup table element (in bytes).

Type UINT32

Values Will have a value of 1 or 2.

CORLUT_PRM_FORMAT

Description Lookup table data format. Determines a single LUT element's type.

Type UINT32

Values For a detailed description of the values, see CorLutNew.

CORLUT_PRM_NENTRIES

Description The number of elements in a lookup table; also, the number of different pixel values which can be transformed by the LUT.

Type UINT32

Values Usually ranges from 256 to 65536.

CORLUT_PRM_NPAGES

Description The number of pages in the lookup table. Usually understood as the number of color planes represented in the LUT.

Type UINT32

Values 1: Monochrome LUT
3: Color LUT

CORLUT_PRM_PHYSADDRESS

Description Physical address of the buffer containing the lookup table.

Type UINT32

Values 32-bit address of the buffer.

CORLUT_PRM_SIGNED

Description Sign of the lookup table elements.

Type UINT32

Values CORLUT_VAL_FORMAT_UNSIGNED
CORLUT_VAL_FORMAT_SIGNED

CORLUT_PRM_SIZE

Description Size of the lookup table data buffer (in bytes).

Type UINT32

Values The value is a product of the number of entries, the element size, and the number of color components (1 for monochrome, 3 for color).

Functions

Function	Description
CorLutAdd	*Performs addition operation on a LUT resource*
CorLutAnd	*Performs logical AND operation on a LUT resource*
CorLutASub	*Performs absolute subtraction operation on a LUT resource*
CorLutBit	*Sets a binary pattern in a LUT resource*
CorLutClip	*Clips the lookup table values of a LUT resource*
CorLutCopy	*Copies a source LUT resource to a destination LUT resource*

Function	Description
CorLutFree	*Releases handle to a LUT resource*
CorLutGamma	*Computes by means of a gamma law the lookup table values of a LUT resource*
CorLutGetPrm	*Gets LUT parameter value from a LUT resource*
CorLutLoad	*Loads a LUT resource from a file*
CorLutMax	*Performs maximum operation on a LUT resource*
CorLutMin	*Performs minimum operation on a LUT resource*
CorLutNew	*Creates in a specified server's memory a new LUT resource*
CorLutNewFromFile	*Creates from a file and in a specified server's memory a new LUT resource*
CorLutNormal	*Sets a normal lookup table for a LUT resource*
CorLutOr	*Performs logical OR operation on a LUT resource*
CorLutRead	*Reads a series of elements from a LUT resource*
CorLutReadEx	*Reads an element from a LUT resource*
CorLutReverse	*Sets a reverse lookup table for a LUT resource*
CorLutRoll	*Shifts the lookup table values of a LUT resource*
CorLutSave	*Saves to a file the content of a LUT resource*
CorLutScale	*Performs scaling operation on a LUT resource*
CorLutSetPrm	*Sets a simple LUT parameter of a LUT resource*
CorLutSetPrmEx	*Sets a complex LUT parameter of a LUT resource*
CorLutShift	*Shifts the lookup table values of a LUT resource*
CorLutSlope	*Slopes the lookup table values of a LUT resource*
CorLutSub	*Performs subtraction operation on a LUT resource*
CorLutThreshold1	*Sets a single-threshold lookup table for a LUT resource*
CorLutThreshold2	*Sets a double-threshold for the lookup table values of a LUT resource*
CorLutWrite	*Writes a series of elements into a LUT resource*
CorLutWriteEx	*Writes an element into a LUT resource*
CorLutXor	*Performs logical XOR operation on a LUT resource*

CorLutAdd

Perform addition operation on a LUT resource

Prototype	CORSTATUS **CorLutAdd**(CORLUT hL*ut*, CORDATA *k*);	
Description	Modifies the LUT values through the addition operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[I]+k*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutASubCorLutSub	

CorLutAnd

Perform logical AND operation on a LUT resource

Prototype	CORSTATUS **CorLutAnd**(CORLUT hL*ut*, CORDATA *k*);	
Description	Modifies the LUT valuesthrough the logical AND operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[i]* AND *k*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutOr and CorLutXor	

CorLutASub

Perform absolute difference operation on a LUT resource

Prototype	CORSTATUS **CorLutASub**(CORLUT hL*ut*, CORDATA *k*);	
Description	Modifies the LUT values through the absolute difference operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i]* = abs(*lut[i]-k*)	
Input	*hLut*	LUT to be transformed
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutAdd and CorLutSub	

CorLutBit

Set a binary pattern in a LUT resource

Prototype	CORSTATUS **CorLutBit**(CORLUT hL*ut*, UINT32 *bit*, CORDATA *k*);
Description	Sets a binary pattern in a LUT resource. Only entries that the bit, as specified by the *bit* argument, is set to 1, are set to the color *k*. Each entry is calculated as follows: *lut[i] = (i & (1L << bit)*) ? *k* : *lut[i]*

Input	*hLut*	LUT resource handle
	bit	Selected bit
	k	Color assigned when bit is on. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorLutClip

Clip the lookup table values of a LUT resource

Prototype	CORSTATUS **CorLutClip**(CORLUT hL*ut*, INT32 *imin*, INT32 *imax*, CORDATA *omin*, CORDATA *omax*);	
Description	Computes the values of a LUT resource so that the gray levels clip to the specified values. This will keep only the pixels in the range [*imin*...*imax*] and map them to the range [*omin*...*omax*].	
Input	*hLut*	LUT resource handle
	imin	Minimum index of clipping region
	imax	Maximum index of clipping region
	omin	Minimum color to which the minimum index is mapped. See Data Types for *CORDATA* definition.
	omax	Maximum color to which the maximum index is mapped. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_ARG_INCOMPATIBLE, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorLutCopy

Copy a source LUT resource to a destination LUT resource

Prototype	CORSTATUS **CorLutCopy**(CORLUT *hSrc*, CORLUT *hDst*);	
Description	Copies the values of one LUT to another.	
Input	*hSrc*	LUT resource handle (source)
	hDst	LUT resource handle (destination)
Output	None	
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_INCOMPATIBLE_LUT and CORSTATUS_INVALID_HANDLE	
Note	When the source LUT size is larger than the destination LUT size, only the section of the source that fits the destination is copied.	

CorLutFree

Release handle to a LUT resource

Prototype	CORSTATUS **CorLutFree**(CORLUT *hLut*);	
Description	Releases handle to a LUT resource.	
Input	*hLut*	LUT resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutNew and CorLutNewFromFile	

CorLutGamma

Compute by means of a gamma law the lookup table values of a LUT resource

Prototype	CORSTATUS **CorLutGamma**(CORLUT *hLut*, FLOAT *factor*);	
Description	Computes the values of a LUT resource using an inverse gamma law with *factor*. Used to correct the camera's light response, which is often set to be a power function (referred to as the gamma function). A gamma factor of 1 means no correction will be applied, and a normal LUT is computed.	
Input	*hLut*	LUT resource handle
	factor	Gamma law factor; must be positive
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_INCOMPATIBLE_LUT and CORSTATUS_INVALID_HANDLE	

CorLutGetPrm

Get LUT parameter value from a LUT resource

Prototype	CORSTATUS **CorLutGetPrm**(CORLUT *hLut*, UINT32 *prm*, void **value*);	
Description	Gets LUT parameter value from a LUT resource	
Input	*hLut*	LUT resource handle
	prm	LUT parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE and CORSTATUS_PRM_INVALID	
See Also	CorLutSetPrm and CorLutSetPrmEx	

CorLutLoad

Load a LUT resource from a file

Prototype	CORSTATUS **CorLutLoad**(CORLUT hL*ut*, const char *f*ilename*);
Description	Loads a LUT resource from a file.

Input	*hLut*	LUT resource handle
	filename	String specifying the path and filename
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_OPEN_ERROR, CORSTATUS_FILE_READ_ERROR, CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
Note	If the LUT buffer is not large enough, data read from file is clipped to the LUT's size.	
See Also	CorLutSave and **LUT File Format**	

CorLutMax

Perform maximum operation on a LUT resource

Prototype	CORSTATUS **CorLutMax**(CORLUT hL*ut*, CORDATA *k*);	
Description	Replaces each element of the existing LUT by the maximum of either its value or the specified gray level *k*. Each entry is calculated as follows: *lut[i]* = max(*k*,*lut[i]*)	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutMin	

CorLutMin

Perform minimum operation on a LUT resource

Prototype	CORSTATUS **CorLutMin**(CORLUT hL*ut*, CORDATA *k*);	
Description	Replaces each element of the existing LUT by the minimum of either its value or the specified gray level *k*. Each entry is calculated as follows: *lut[i]* = min(*k*,*lut[i]*)	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutMax	

CorLutNew

Create in a specified server's memory a new LUT resource

Prototype	CORSTATUS **CorLutNew**(CORSERVER *hServer*, UINT32 *nEntries*, UNIT32 *format*, CORLUT **hLut*);	
Description	Allocates and initializes a LUT of the specified size and format.	
Input	*hServer*	Server handle
	nEntries	Number of entries in the LUT

format The format of the LUT entries is determined either by the ORing the format and bit width values, or using one of the combined values. The sign value can optionally be ORed to the result.

Format values:

Monochrome: Each element stores monochrome data.
The LUT is a single vector of such elements.

CORLUT_VAL_FORMAT_MONO8
CORLUT_VAL_FORMAT_MONO9
CORLUT_VAL_FORMAT_MONO10
CORLUT_VAL_FORMAT_MONO11
CORLUT_VAL_FORMAT_MONO12
CORLUT_VAL_FORMAT_MONO13
CORLUT_VAL_FORMAT_MONO14
CORLUT_VAL_FORMAT_MONO15
CORLUT_VAL_FORMAT_MONO16

Unsigned integer (same as monochrome): Each element stores monochrome data. The LUT is a single vector of such elements.

CORLUT_VAL_FORMAT_UINT8
CORLUT_VAL_FORMAT_UINT9
CORLUT_VAL_FORMAT_UINT10
CORLUT_VAL_FORMAT_UINT11
CORLUT_VAL_FORMAT_UINT12
CORLUT_VAL_FORMAT_UINT13
CORLUT_VAL_FORMAT_UINT14
CORLUT_VAL_FORMAT_UINT15
CORLUT_VAL_FORMAT_UINT16

Signed integer (monochrome with a sign)

Each element stores monochrome data. The LUT is a single vector of such elements

CORLUT_VAL_FORMAT_INT8
CORLUT_VAL_FORMAT_INT9
CORLUT_VAL_FORMAT_INT10
CORLUT_VAL_FORMAT_INT11
CORLUT_VAL_FORMAT_INT12
CORLUT_VAL_FORMAT_INT13
CORLUT_VAL_FORMAT_INT14
CORLUT_VAL_FORMAT_INT15
CORLUT_VAL_FORMAT_INT16

Color non-interlaced: One element stores data for one color component. Each color component is represented as a separate vector of single-component elements.

CORLUT_VAL_FORMAT_COLORNI8
CORLUT_VAL_FORMAT_COLORNI9
CORLUT_VAL_FORMAT_COLORNI10
CORLUT_VAL_FORMAT_COLORNI11
CORLUT_VAL_FORMAT_COLORNI12
CORLUT_VAL_FORMAT_COLORNI13
CORLUT_VAL_FORMAT_COLORNI14

CORLUT_VAL_FORMAT_COLORNI15
CORLUT_VAL_FORMAT_COLORNI16

Color interlaced: One element stores data for the three color components. The LUT is a single vector of such elements.

CORLUT_VAL_FORMAT_COLORI8
CORLUT_VAL_FORMAT_COLORI9
CORLUT_VAL_FORMAT_COLORI10
CORLUT_VAL_FORMAT_COLORI11
CORLUT_VAL_FORMAT_COLORI12
CORLUT_VAL_FORMAT_COLORI13
CORLUT_VAL_FORMAT_COLORI14
CORLUT_VAL_FORMAT_COLORI15
CORLUT_VAL_FORMAT_COLORI16

Output	*hLut*	LUT resource handle
Return Value	CORSTATUS_ARG_INVALID_VALUE, CORSTATUS_ARG_NULL (if *hLut* is NULL), CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
See Also	CorLutFree and CorLutNewFromFile	

CorLutNewFromFile

Create from a file and in specified server's memory a new LUT resource

Prototype	CORSTATUS **CorLutNewFromFile**(CORSERVER *hServer*, const *char *filename*, CORLUT **hLut*);	
Description	Allocates and initializes a LUT of the same format as the designated file's LUT.	
Input	*hServer*	Server handle
	filename	String specifying the path and name
Output	*hLut*	LUT resource handle
Return Value	CORSTATUS_ARG_NULL (if *filename* is NULL) CORSTATUS_FILE_OPEN_ERROR, CORSTATUS_FILE_READ_ERROR, CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
Note	Same as calling CorLutNew then CorLutLoad.	
See Also	CorLutFree and CorLutNew	

CorLutNormal

Set a normal lookup table for a LUT resource

Prototype	CORSTATUS **CorLutNormal**(CORLUT *hLut*);	
Description	Defines a normal (linear) LUT. Each entry is assigned the following value: *lut[i] = i * (2 ^{n} / CORLUT_PRM_NENTRIES)* *where n = number of bits per entry (See CORLUT_PRM_FORMAT)*	
Input	*hLut*	LUT resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorLutOr

Perform logical OR operation on a LUT resource

Prototype	CORSTATUS **CorLutOr**(CORLUT *hLut*, CORDATA *k*);	
Description	Modifies the values of a LUT through the logical OR operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[i]* OR *k*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for CORDATA definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutXor	

CorLutRead

Read a series of elements from a LUT resource

Prototype	CORSTATUS **CorLutRead**(CORLUT hL*ut*, UINT32 *offset*, void **array, UINT32 size*);	
Description	Reads a consecutive series of elements from the specified LUT and copies them into an one-dimensional destination array.	
Input	*hLut*	LUT resource handle
	offset	Offset to seek within the LUT prior to copy
	size	Size of transfer (*nElements* × *elementSize* bytes)
Output	*array*	Array which can accommodate the requested number of elements (*nElements* × *elementSize*)
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL), CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	
See Also	CorLutWrite	

CorLutReadEx

Read an element from a LUT resource

Prototype	CORSTATUS **CorLutReadEx**(CORLUT hL*ut*, UINT32 *offset*, CORDATA *element);	
Description	Reads an element from the specified LUT.	
Input	*hLut*	LUT resource handle
	offset	Offset to seek within the LUT prior to read
Output	*element*	Current value of the element. See Data Types for *CORDATA* definition.
Return Value	CORSTATUS_ARG_NULL (if *element* is NULL), CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	
See Also	CorLutWriteEx	

CorLutReverse

Sets a reverse lookup table for a LUT resource

Prototype	CORSTATUS **CorLutReverse**(CORLUT *hLut*);	
Description	Sets a reverse LUT: *lut[i] = max-i*, where *max* is the highest pixel value. For instance, for an unsigned 8 bit/pixel image, *max* is set to 255; an unsigned 16 bit/pixel has *max* set to 65535.	
Input	*hLut*	LUT resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorLutRoll

Shift the lookup table values of a LUT resource

Prototype	CORSTATUS **CorLutRoll**(CORLUT *hLut*, INT32 *rol*);	
Description	Shifts a LUT, wrapping the values at each end. The direction of shift is determined by the sign of the argument *rol*. A positive *rol* shifts the LUT from the low indexes to the higher ones, while a negative *rol* shifts the LUT from high indexes to lower ones.	
Input	*hLut*	LUT resource handle
	rol	Number of shifts of the LUT indexes
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY	
See Also	CorLutShift	

CorLutSave

Saves to a file the content of a LUT resource

Prototype	CORSTATUS **CorLutSave**(CORLUT *hLut*, const char **filename*);	
Description	Saves to a file the content of a LUT resource.	
Input	*hLut*	LUT resource handle
	filename	String specifying the path and filename
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *filename* is NULL), CORSTATUS_FILE_CREATE_ERROR, CORSTATUS_FILE_WRITE_ERROR and CORSTATUS_INVALID_HANDLE	
See Also	CorLutLoad and **LUT File Format**	

CorLutScale

Perform scaling operation on a LUT resource

Prototype	CORSTATUS **CorLutScale**(CORLUT *hLut*, CORDATA *k*);	
Description	Modifies the values of a LUT through a scaling operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[i]*k/maxcolor*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorLutSetPrm

Set a simple LUT parameter of a LUT resource

Prototype	CORSTATUS **CorLutSetPrm**(CORLUT *hLut*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple LUT parameter of a LUT resource.	
Input	*hLut*	LUT resource handle
	prm	LUT parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID and CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32. If the parameter is complex, use CorLutSetPrmEx. All LUT parameters are read-only parameters. Therefore, you can get their values with CorLutGetPrm but cannot change these values with CorLutSetPrm.	
See Also	CorLutGetPrm and CorLutSetPrmEx	

CorLutSetPrmEx

Set a complex LUT parameter of a LUT resource

Prototype	CORSTATUS **CorLutSetPrmEx**(CORLUT *hLut*, UINT32 *prm*, const void **value*);	
Description	Sets a complex LUT parameter of a LUT resource.	
Input	*hLut*	LUT resource handle
	prm	LUT parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE CORSTATUS_PRM_INVALID and CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is one whose size is greater than an UINT32. If the parameter size is UINT32, use either CorLutSetPrm or CorLutSetPrmEx.	

See Also CorLutGetPrm and CorLutSetPrm

CorLutShift

Shift the lookup table values of a LUT resource

Prototype	CORSTATUS **CorLutShift**(CORLUT *hLut*, INT32 *nShift*);	
Description	Shifts the values of a LUT by *nShift*. If *nShift* is positive, values are shifted left; if *nShift* is negative, values are shifted right.	
Input	*hLut*	LUT resource handle
	nShift	Number of shift bits
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutRoll	

CorLutSlope

Slope the lookup table values of a LUT resource

Prototype	CORSTATUS **CorLutSlope**(CORLUT *hLut*, INT32 *i1*, CORDATA *c1*, INT32 *i2*, CORDATA *c2*);	
Description	Modifies the values of the LUT so that the pixels in the range [*i1...i2*] are mapped to the range [*c1...c2*]. Pixels outside of the range are unchanged.	
Input	*hLut*	LUT resource handle
	i1	Minimum index of slope region.
	c1	Minimum color to which the minimum index is mapped. See Data Types for *CORDATA* definition.
	i2	Maximum index of slope region
	c2	Maximum color to which the maximum index is mapped. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_ARG_INCOMPATIBLE, CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	

CorLutSub

Perform subtraction operation on a LUT resource

Prototype	CORSTATUS **CorLutSub**(CORLUT *hLut*, CORDATA *k*);	
Description	Modifies the values of a LUT through the subtraction operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[i]-k*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutAdd and CorLutASub	

CorLutThreshold1

Sets a single-threshold lookup table for a LUT resource

Prototype	CORSTATUS **CorLutThreshold1**(CORLUT *hLut*, CORDATA *thrs*);	
Description	Sets a threshold LUT. Pixels under *thrs* are set to the lowest color value, the others are set to the highest color value.	
Input	*hLut*	LUT resource handle
	thrs	Threshold gray level. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutThreshold2	

CorLutThreshold2

Sets a double-threshold lookup table for a LUT resource

Prototype	CORSTATUS **CorLutThreshold2**(CORLUT *hLut*, CORDATA *thrs1*, CORDATA *thrs2*);	
Description	Sets a threshold LUT. Pixels in the range [*thrs1...thrs2*] are mapped to the highest color value, the others are set to the lowest color value.	
Input	*hLut*	LUT resource handle
	thrs1	Threshold gray level. See Data Types for *CORDATA* definition.
	thrs2	Threshold gray level. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutThreshold1	

CorLutWrite

Write a series of elements into a LUT resource

Prototype	CORSTATUS **CorLutWrite**(CORLUT *hLut*, UINT32 *offset*, void ***array, UINT32 size*);	
Description	Writes a series of elements from an one-dimensional source array to a LUT resource.	
Input	*hLut*	LUT resource handle
	offset	Offset to seek within the LUT prior to copy
	array	Array which contains the elements to be written (*nElements*×*elementSize*)
	size	Size of transfer (*nElements*×*elementSize*)
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *array* is NULL), CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	
See Also	CorLutRead	

CorLutWriteEx

Write an element into a LUT resource

Prototype	CORSTATUS **CorLutWriteEx**(CORLUT *hLut*, UINT32 *offset*, CORDATA data);	
Description	Writes an element to a LUT resource.	
Input	*hLut*	LUT resource handle
	offset	Offset to seek within the LUT prior to write
	data	New value of the element. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE and CORSTATUS_INVALID_HANDLE	
See Also	CorLutReadEx	

CorLutXor

Perform logical XOR operation on a LUT resource

Prototype	CORSTATUS **CorLutXor**(CORLUT *hLut*, CORDATA *k*);	
Description	Modifies the values of a LUT through the logical XOR operation, using a gray level of *k*. Each entry is calculated as follows: *lut[i] = lut[i]* XOR *k*	
Input	*hLut*	LUT resource handle
	k	Constant. See Data Types for *CORDATA* definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorLutOr	

Manager Module

The Manager Module allows the application to connect to other available resources (boards).

Functions

Function	Description
CorManAllocContigBuffer	*Allocates a contiguous memory block*
CorManClose	*Closes the Sapera standard API*
CorManExecuteCmd	*Executes an application on a specified server*
CorManFreeContigBuffer	*Frees a contiguous memory block*
CorManGetHandleByIndex	*Gets a handle registered on a remote server by index.*
CorManGetHandleByName	*Gets a handle registered on a remote server from its name.*
CorManGetInstallationDirectory	*Gets full path of installation directory.*
CorManGetLocalServer	*Gets handle for local server*
CorManGetPixelDepthMax	*Returns the maximum number of significant bits per component for a data format*
CorManGetPixelDepthMin	*Returns the minimum number of significant bits per component for a data format*
CorManGetRemoteServerByName	*Gets the server handle corresponding to a specific name on a remote server*
CorManGetRemoteServerChild	*Gets the server handle corresponding to one child of a parent remote server.*
CorManGetRemoteServerParent	*Gets the server handle corresponding to the parent of one child remote server.*
CorManGetServerByIndex	*Gets server handle by index*
CorManGetServerByName	*Gets server handle from its name*
CorManGetServerCount	*Gets number of available servers*
CorManGetServerSerialNumber	*Gets the serial number for the specified server*
CorManGetStatusText	*Gets text for ID and info fields in status code*
CorManGetStatusTextEx	*Gets text for all fields in status code*
CorManGetStringFromFormat	*Gets a text description of a Sapera data format*
CorManIsLocalHandle	*Checks for a local handle*
CorManIsSameLocation	*Checks for a local handle (obsolete)*
CorManIsServerAccessible	*Checks if a server is accessible in the server database*
CorManIsSystemHandle	*Checks for a System handle*
CorManIsMambaHandle	*Checks for a MAMBA handle*

CorManLogMessage	*Adds a "printf-like" user string in the Sapera Log Viewer.*
CorManLogStatus	*Adds a predefined string corresponding to a status in the Sapera Log Viewer.*
CorManMapBuffer	*Maps a contiguous memory block in current process address space*
CorManOpen	*Initializes the Sapera standard API*
CorManRegisterCallback	*Registers a callback function to be called when receiving a user command*
CorManRegisterCallbackEx	*Registers a callback function to be called for server related events*
CorManRegisterHandle	*Adds a handle to the local handle database to allow other servers to access it*
CorManReleaseHandle	*Releases a handle obtained from CorManGetHandleByName or CorManGetHandleByIndex*
CorManReleaseServer	*Releases a server handle*
CorManResetServer	*Resets a server (hardware reset)*
CorManSetLocalServerName	*Sets name of local server*
CorManSetTimeout	*Sets communication time out*
CorManSoftResetServer	*Resets a server (software reset)*
CorManUnmapBuffer	*Unmaps a previously mapped contiguous memory block in current process address space*
CorManUnregisterCallback	*Unregisters the callback function to be called when receiving a user command*
CorManUnregisterCallbackEx	*Unregisters the callback function to be called for server related events*
CorManUnregisterHandle	*Removes a handle from the local handle database*
CorManUserCmd	*Sends a user command to a server*
CorManWaitForServerReady	*Waits until a given server is ready*

CorManAllocContigBuffer

Allocate a contiguous memory block

Prototype	CORSTATUS **CorManAllocContigBuffer**(UINT32 *nBytes*, UINT32 **physAddr*, void ***addr*);	
Description	Allocates a block in contiguous memory.	
Input	*nBytes*	Number of bytes requested
Output	*physAddr*	Physical address
	addr	Virtual address
Return Value	CORSTATUS_ARG_NULL (if *physAddr* or *addr* is NULL) and CORSTATUS_NO_MEMORY	
See Also	CorManFreeContigBuffer and Contiguous Memory	

CorManClose

Closes the Sapera standard API

Prototype	CORSTATUS **CorManClose**(void);
Description	Terminates all access to the standard C library. This must be the last Sapera call in an application program.
Return Value	CORSTATUS_INSUFFICIENT_RESOURCES
Notes	This function must not be called from the DllMain function of a Windows DLL
See Also	CorManOpen

CorManExecuteCmd

Executes an application on a specified server

Prototype	CORSTATUS **CorManExecuteCmd**(CORSERVER hServer, const char *szCmdLine*[])	
Description	Executes a command line application on a remote server. For example, this function can run a user application on a processing board such as the Mamba.	
Input	*hServer*	Board server handle
	szCmdLine	String containing the application name to execute (including any application arguments)
Return Value	CORSTATUS_NO_MEMORY, CORSTATUS_ARG_INVALID, CORSTATUS_INVALID_HANDLE, CORSTATUS_SERVER_NOT_FOUND, CORSTATUS_TIMEOUT and CORSTATUS_BOARD_NOT_READY	
Notes	The executed application uses the environment variables (e.g., PATH) defined in the remote server's system.	

CorManFreeContigBuffer

Free a contiguous memory buffer

Prototype	CORSTATUS **CorManFreeContigBuffer**(void **addr*);	
Description	Frees a contiguous memory block.	
Input	*addr*	Virtual address
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *addr* is NULL) and CORSTATUS_ARG_INVALID	
See Also	CorManAllocContigBuffer and Contiguous Memory	

CorManGetHandleByIndex

Gets a handle registered on a remote server by index.

Prototype	CORSTATUS **CorManGetHandleByIndex**(CORSERVER *hServer*, UINT32 *index*, PCORHANDLE *pHandle*);
Description	Gets a handle from a remote server's handle database by index. The handle must have been previously registered using CorManRegisterHandle on the remote server.

Input	*hServer*	Handle to the remote server where the handle is registered.
	index	Index to the remote server's handle database.
Output	*pHandle*	Pointer to a handle.
Return Value	CORSTATUS_ARG_NULL (if *pHandle* is NULL) and CORSTATUS_NOT_ACCESSIBLE	
See Also	CorManRegisterHandle, CorManGetHandleByName and CorManReleaseHandle	

CorManGetHandleByName

Gets a handle registered on a remote server from its name.

Prototype	CORSTATUS **CorManGetHandleByName**(CORSERVER *hServer*, PSTR *name*, PCORHANDLE *pHandle*);	
Description	Gets a handle from a remote server's handle database from its name. The handle must have been previously registered using CorManRegisterHandle on the remote server.	
Input	*hServer*	Handle to the remote server on which to get the handle.
	name	Name of the handle in the remote server's handle database.
Output	*pHandle*	Pointer to a handle.
Return Value	CORSTATUS_ARG_NULL (if *name* or *pHandle* is NULL) CORSTATUS_NOT_ACCESSIBLE	
See Also	CorManRegisterHandle, CorManGetHandleByIndex and CorManReleaseHandle	

CorManGetInstallationDirectory

Gets full path of installation directory

Prototype	CORSTATUS **CorManGetInstallationDirectory**(CORSERVER *hServer*, PSTR *pInstallDir*, UINT32 strSize);	
Description	Gets the directory where Sapera LT or a Sapera driver is installed.	
Input	*hServer*	Handle of Sapera LT system server or Sapera board server.
	strSize	Size of buffer for storing the installation directory.
Output	*pInstallDir*	Product installation directory (e.g., "c:\DALSA Coreco\Sapera").
Return Value	CORSTATUS_ARG_NULL	
See Also	CorManGetLocalServer, CorManGetServerByIndex and CorManGetServerByName	

CorManGetLocalServer

Get handle for local server

Prototype	CORSERVER **CorManGetLocalServer**();
Description	Gets server handle corresponding to current process.
Input	None
Output	None
Return Value	Local server handle
Notes	Use this function instead of the now obsolete CorManGetServer. Under Win32, the returned server handle may not be the same as the one you get when you use CorManGetServerByName for the local system.
See Also	CorManGetServerByName

CorManGetPixelDepthMax

Prototype	UINT32 **CorManGetPixelDepthMax**(UINT32 *format*);
Description	Returns the maximum number of significant bits per component for a data format. This value is documented for each data format in the Data Formats section.
Input	*format* Data format (Refer to Data Formats section for a detailed list).
Output	*None*
Return Value	The maximum number of bits per component.
See Also	CORBUFFER_PRM_PIXEL_DEPTH

CorManGetPixelDepthMin

Prototype	UINT32 **CorManGetPixelDepthMin**(UINT32 *format*);
Description	Returns the minimum number of significant bits per component for a data format. This value is documented for each data format in the Data Formats section.
Input	*format* Data format (Refer to Data Formats section for a detailed list).
Output	*None*
Return Value	The minimum number of bits per component.
See Also	CORBUFFER_PRM_PIXEL_DEPTH

CorManGetRemoteServerByName

Get the server handle from a specific remote server name

Prototype	CORSTATUS **CorManGetServerByName**(CORSERVER *hRemoteServer*, const char **name*, CORSERVER **hServer*);
Description	Gets the server handle corresponding to a specific name on a remote server Use this function to get from another Win32 system a handle to a server corresponding to an

individual Win32 process (not listed in the Sapera Server database). You first need to create an alias for this server from its own process on the remote system.

Input	*hRemoteServer*	Remote server handle
	name	Server name
Output	*hServer*	Server handle
Return Value	CORSTATUS_ARG_INVALID and CORSTATUS_SERVER_NOT_FOUND	
See Also	CorManSetLocalServerName	

CorManGetRemoteServerChild

Get the child server handle from a remote parent server

Prototype	CORSTATUS **CorManGetRemoteServerChild**(CORSERVER *hServer*, UINT8 *nChild*, PCORSERVER *handle*)
Description	This function obtains the child server handle via the parent server handle. Useful for boards with multiple processors. For example the Python board contains one parent server (Python_1) and four child processors (Python_1_C60.. Python_4_C60). This function allows you to obtain any child server handle without specifying the name of that specific child server.
Input	*hServer* Parent server handle
	nChild Child server index (1..N) where N is the number of child servers on the parent board.
Output	*handle* Child server handle
Return Value	CORSTATUS_ARG_INVALID and CORSTATUS_INVALID_HANDLE
See Also	CorManGetRemoteServerParent

CorManGetRemoteServerParent

Get the parent server handle from a remote child server

Prototype	CORSTATUS **CorManGetRemoteServerParent**(CORSERVER *hServer*, PCORSERVER *handle*)
Description	This function obtains the parent server handle via the handle of one of its child servers. Useful on boards with multiple processors. For example the Python board contains one parent server (Python_1) and four child processors (Python_1_C60.. Python_4_C60). This function allows you to obtain the parent server handle without specifying the name of the parent.
Input	*hServer* Child server handle
Output	*handle* Parent server handle
Return Value	CORSTATUS_ARG_INVALID and CORSTATUS_INVALID_HANDLE
See Also	CorManGetRemoteServerChild

CorManGetServerByIndex

Get server handle by index

Prototype	CORSTATUS **CorManGetServerByIndex**(UINT32 *index*, char **name*, CORSERVER **hServer*);	
Description	Use this function to retrieve handles for servers listed in the Sapera Server database.	
Input	*index*	Specifies which server to get. Valid values are in the range [0...*count*-1], where *count* is the value returned by CorManGetServerCount.
Output	*name*	Server name Server names are limited to 30 characters. If this argument is NULL, the server name won't be returned.
	hServer	Server handle. If this argument is NULL, the handle won't be returned, which is useful if only the server name is needed.
Return Value	CORSTATUS_SERVER_NOT_FOUND	
Notes	Use the Sapera configuration program to obtain a list of all available servers in your system and their names.	
See Also	CorManGetServerCount, CorManGetServerByName and CorManReleaseServer	

CorManGetServerByName

Get server handle from its name

Prototype	CORSTATUS **CorManGetServerByName**(const char **name*, CORSERVER **server*);	
Description	Gets server handle from its name.	
Input	*name*	Server name
Output	*server*	Server handle
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_SERVER_NOT_FOUND	
Notes	Use the Sapera configuration program to obtain a list of all available servers in your system and their names. Use this function to retrieve handles both for servers listed in the Sapera Server database and for those corresponding to individual processes. For the latter, use an alias you created previously for this server from its own process.	
See Also	CorManGetServerByIndex CorManSetLocalServerName CorManReleaseServer	

CorManGetServerCount

Get the number of available server

Prototype	CORSTATUS **CorManGetServerCount**(UINT32 **count*);	
Description	Gets the number of available servers.	
Input	None	
Output	*count*	Number of available servers
Return Value	CORSTATUS_OK	

Notes	Use the Sapera configuration program to obtain a list of all available servers in your system. This function returns the number of servers currently listed in the Sapera Server database, it does not include those associated to individual Win32 processes.
See Also	CorManGetServerByIndex

CorManGetServerSerialNumber

Gets the serial number for the specified server.

Prototype	CORSTATUS **CorManGetServerSerialNumber**(CORSERVER *hServer*, PSTR *serial*);	
Description	Gets the serial number for the specified Sapera Server. This number is assigned by DALSA and programmed into the EEPROM of the board associated with the server.	
Input	*hServer*	Handle to the server.
Output	*serial*	Character string that receives the resulting information in the format seen within the board's Viewer window. The first letter is either an "S" or an "H" followed by seven numbers, for example, 'S1234567'. Make certain that the string is long enough for the serial number plus the terminating NULL character.
Return Value	CORSTATUS_ARG_NULL (if *serial* is NULL) CORSTATUS_ARG_INVALID	
Notes	There is no serial number associated with the System server	
See Also	CorManGetServerByIndex and CorManGetServerByName	

CorManGetStatusText

Get text strings for "ID" and "info" fields of a given status value

Prototype	CORSTATUS **CorManGetStatusText**(CORSTATUS *status*, char **idBuf*, UINT32 *idBufSize*, char **infoBuf*, UINT32 *infoBufSize*);	
Description	Gets text strings for "ID" and "info" fields of a given status value.	
Input	*Status*	Status value
	idBufSize	ID buffer size
	infoBufSize	Information buffer size
Output	*idBuf*	ID string
	infoBuf	Information string
Return Value	CORSTATUS_ARG_OUT_OF_RANGE	

CorManGetStatusTextEx

Get text strings for all the fields of a given status value

Prototype	CORSTATUS **CorManGetStatusTextEx**(CORSTATUS *status*, char **idBuf*, UINT32 *idBufSize*, char **infoBuf*, UINT32 *infoBufSize*, char **levelBuf*, UINT32 *levelBufSize*, char **moduleBuf*, UINT32 *moduleBufSize*);	
Description	Gets text strings for all the fields (ID, Info, Level, and Module) of a given status value.	
Input	*status*	Status value
	idBufSize	ID buffer size
	infoBufSize	Information buffer size
	levelBufSize	Level buffer size
	moduleBufSize	Module buffer size
Output	*idBuf*	ID string
	infoBuf	Information string
	levelBuf	Level string
	moduleBuf	Module string
Return Value	CORSTATUS_ARG_OUT_OF_RANGE	

CorManGetStringFromFormat

Get a text description of a Sapera data format

Prototype	BOOL **CorManGetSringFromFormat**(UINT32 *format*, char **szFormat*);	
Description	Get an identification string for the Sapera data format, e.g., 'RGB888'	
Input	*format*	Sapera data format
Output	*szFormat*	String description, set to '(unknown)' if the format is unrecognized
Return Value	TRUE if a non-NULL string argument is specified , FALSE otherwise	

CorManIsLocalHandle

Check for a local handle

Prototype	BOOLEAN **CorManIsLocalHandle**(CORHANDLE *handle*);	
Description	Checks for a handle belonging to the local server.	
Input	*handle*	Sapera handle
Output	None	
Return Value	TRUE if handle is a local handle, FALSE otherwise	

CorManIsSameLocation

Check for a local handle (obsolete)

Prototype	BOOLEAN **CorManIsSameLocation**(CORHANDLE *handle*);

Description	Checks for a handle belonging to the local server.	
Input	*handle*	Sapera handle
Output	None	
Return Value	TRUE if handle is a local handle, FALSE otherwise	
Note	Obsolete. Identical to the preferred function CorManIsLocalHandle	

CorManIsServerAccessible

Checks if a server is accessible in the server database

Prototype	BOOLEAN **CorManIsServerAccessible**(UINT32 *index*);
Description	Checks if the resources belonging to a server are currently accessible. Although existing handles for these resources are still valid when their server becomes unaccessible, they must be left alone or released (e.g., CorAcqDeviceRelease). When a Sapera application starts, all detected servers are automatically accessible. However, some Sapera devices like GigE cameras 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. Accessibility of servers can also be determined by registering callbacks for server related events using CorManRegisterCallbackEx.
Input	*index* Server index
Output	None
Return Value	TRUE if server is accessible, FALSE otherwise
See Also	CorManRegisterCallbackEx

CorManIsSystemHandle

Check for a system handle

Prototype	BOOLEAN **CorManIsSystemHandle**(CORHANDLE *handle*);
Description	Checks for a handle belonging to the 'System' server.
Input	*handle* SAPERA handle
Output	None
Return Value	TRUE if handle is a System handle, FALSE otherwise

CorManIsMambaHandle

Check for a MAMBA handle

Prototype	BOOLEAN **CorManIsMambaHandle**(CORHANDLE *handle*);
Description	Checks for a handle belonging to a Mamba server.
Input	*handle* SAPERA handle
Output	None
Return Value	TRUE if handle is a Mamba handle, FALSE otherwise

CorManLogMessage

Appends a “printf-like” user string to the Sapera Log Viewer output

Prototype CORSTATUS **CorManLogMessage**(UINT32 *logtype*, PCSTR *msg*, PCSTR *file*, UINT32 *line*)

Description This function allows the appended user string to implement formatting as available with the *printf* function so as to display any data type.

Input

logtype Error level:
CORLOG_TYPEID_ERR (Normal error)
CORLOG_TYPEID_FAT (Fatal error)
CORLOG_TYPEID_WRN (Warning)
CORLOG_TYPEID_INF (Information)

msg A “printf-like” string containing the message to display.

file String containing the file name in which the error occurred.
The macro *__FILE__* can be used to specify the current file.

line Line number where the error occurred.
The macro *__LINE__* can be used to specify the current line.

Output None

Return Value CORSTATUS_OK and CORSTATUS_ARG_NULL

CorManLogStatus

Appends a predefined status string to the Sapera Log Viewer output

Prototype CORSTATUS **CorManLogStatus**(CORSTATUS *status*, PCSTR *file*, UINT32 *line*)

Description The appended string is composed with three different status fields. This function does not allow custom formatting. A custom message can be appended by extracting the required status fields with *CorManGetStatusTextEx*, and then formatted using *CorManLogMessage.*

Input

status Error status returned by any Sapera function.

file String containing the file name in which the error occurred.
The macro *__FILE__* can be used to specify the current file.

line Line number where the error occurred.
The macro *__LINE__* can be used to specify the current line.

Output None

Return Value CORSTATUS_OK and CORSTATUS_ARG_NULL

See Also CorManGetStatusTextEx

CorManMapBuffer

Map a contiguous memory block in current process address space

Prototype CORSTATUS **CorManMapBuffer**(UINT32 *physAddr*, UINT32 *size*, void ***virtualAddr*);

Description Maps a contiguous memory block in current process address space

Input

physAddr Physical address of a contiguous memory block

size Contiguous memory block size in bytes

Output	*virtualAddr* Virtual address of the contiguous memory block
Return Value	CORSTATUS_NO_MEMORY
Note	To unmap a previously mapped contiguous memory block use CorManUnmapBuffer.
See Also	CorManUnmapBuffer

CorManOpen

Initializes the Sapera standard API

Prototype	CORSTATUS **CorManOpen**(void);
Description	Initiates access to the standard C library. This must be the first Sapera call in an application program.
Return Value	CORSTATUS_INSUFFICIENT_RESOURCES
Notes	This function must not be called from the DllMain function of a Windows DLL
See Also	CorManClose

CorManRegisterCallback

Register a callback function to be called when receiving a user command

Prototype	CORSTATUS **CorManRegisterCallback**(CORSERVER *hServer*, PCORMANCALLBACK *callback*);
Description	Registers a callback function to be called when receiving a user command
Input	*hServer* Server handle.
	callback Callback function to call
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
Note	The callback function must be defined as: CORSTATUS CCONV callbackFct(UINT32 cmd, void *inData, UINT32 inDataSize, void *outData, UINT32 outDataSize);
See Also	CorManUnregisterCallback and CorManUserCmd

CorManRegisterCallbackEx

Register a callback function for server related events

Prototype	CORSTATUS **CorManRegisterCallbackEx**(UINT32 *eventType*, PCOREVENTINFOCALLBACK *callback*, void **context*);
Description	Registers a callback function for server related events. The callback function provides information on the corresponding event (in the *COREVENTINFO* handle). Refer to the *EventInfo* module for more detail on the available information. The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context. Note that server related events are only available when dealing with Sapera devices like GigE cameras, that can be connected and disconnected while a Sapera application is running.

Input	*eventType*	Type of event to register. The callback function will be called when the specified event(s) occur. The values may be ORed if more than one event is desired. The CORMAN_VAL_EVENT_TYPE_SERVER_NEW event occurs when a new device is connected while a Sapera application is already running. The CORMAN_VAL_EVENT_TYPE_SERVER_NOT_ACCESSIBLE event occurs when the device corresponding to an existing server is disconnected. The CORMAN_VAL_EVENT_TYPE_SERVER_ACCESSIBLE event occurs when the device corresponding to an existing, unaccessible server is reconnected. The CORMAN_VAL_EVENT_TYPE_SERVER_DATABASE_FULL event occurs when there is no room left in the Sapera server database for a new device that has just been connected.

	callback	Address of a user callback function of the following form:
		
		`CORSTATUS CCONV MyCallback(void *context, COREVENTINFO hEventInfo)`
		`{`
		`}`

	context	Pointer to user storage (i.e., variable, structure, buffer, etc). Can be NULL.
Output	None	In the callback function, obtain the event type that triggered the callback by reading COREVENTINFO_PRM_EVENT_TYPE. For all events except the last, you can obtain a handle to the server by calling *CorManGetServerByIndex* using the server index specified by COREVENTINFO_PRM_SERVER_INDEX.
Return Value	CORSTATUS_ARG_NULL (if *callback* is NULL), CORSTATUS_NOT_AVAILABLE, CORSTATUS_RESOURCE_IN_USE, CORSTATUS_TIMEOUT	
Notes		
See Also	CorManIsServerAccessible, CorManRegisterCallbackEx	

CorManRegisterHandle

Adds a handle to the local handle database to allow other servers to access it.

Prototype	CORSTATUS **CorManRegisterHandle**(CORHANDLE *handle*, PSTR *name*, PUINT32 *pIndex*);	
Description	By registering a handle using this function you allow all other servers to get this handle through CorManGetHandleByIndex or CorManGetHandleByName.	
Input	*handle*	handle to register.
	name	Name to give the handle in the database. If NULL, a default name “Handle_X” is given, where “X” corresponds to the index returned by *pIndex* (e.g., Handle_0, Handle_1, …).
Output	*pIndex*	Index in the handle database where the handle is added. Can be NULL.
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorManUnregisterHandle, CorManGetHandleByIndex and CorManGetHandleByName	

CorManReleaseHandle

Releases a handle obtained from *CorManGetHandleByName* or *CorManGetHandleByIndex.*

Prototype	CORSTATUS **CorManReleaseHandle**(CORHANDLE *handle*);
Description	Releases a handle obtained from *CorManGetHandleByName* or *CorManGetHandleByIndex*.
Input	*handle* Handle to release.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorManGetHandleByIndex and CorManGetHandleByName

CorManReleaseServer

Release server handle

Prototype	CORSTATUS **CorManReleaseServer**(CORSERVER *hServer*);
Description	Releases server handle.
Input	*hServer* Server handle.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorManGetLocalServer, CorManGetRemoteServerByName, CorManGetServerByIndex and CorManGetServerByName

CorManResetServer

Resets server (hardware reset)

Prototype	CORSTATUS **CorManResetServer**(CORSERVER *hServer*);
Description	Performs a hardware reset on a server.
Input	*hServer* Server handle.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
Notes	After calling this function, all resources from the server cannot be used anymore; they must be released first. Calling this function for a Mamba handle will bypass the Windows NT shutdown procedure on the Mamba.

CorManSetLocalServerName

Set local server name

Prototype	CORSTATUS **CorManSetLocalServerName**(const char **serverName*);
Description	Sets a new name for the server corresponding to current process
Input	*serverName* New name for local server
Output	None
Return Value	CORSTATUS_ARG_NULL *(if serverName is NULL)*

	CORSTATUS_RESOURCE_LOCKED
Note	Defines an alias for the server corresponding to the current process, so that its handle can be retrieved from a remote server.
See Also	CorManGetLocalServer and CorManGetRemoteServerByName

CorManSoftResetServer

Resets server (software reset)

Prototype	CORSTATUS **CorManSoftResetServer**(CORSERVER *hServer*);
Description	Performs a software reset on a server.
Input	*hServer* Server handle.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
Notes	After calling this function, all resources from the server cannot be used anymore; they must be released first. Calling this function for a Mamba handle will perform the Windows NT shutdown procedure on the Mamba.

CorManSetTimeout

Set communication timeout

Prototype	void **CorManSetTimeout**(UINT32 *timeOut*);
Description	Sets communication timeout.
Input	*timeOut* Communication timeout in milliseconds.
Output	None
Return Value	(none; function has void return type)

CorManUnmapBuffer

Unmap a contiguous memory block in current process address space

Prototype	CORSTATUS **CorManUnmapBuffer(** void **virtualAddr*);
Description	Unmaps a contiguous memory block in current process address space
Input	*virtualAddr* Previously mapped virtual address to be unmapped.
Output	None
Return Value	CORSTATUS_ARG_NULL *(if virtualAddr is NULL)*
See Also	CorManMapBuffer

CorManUnregisterCallback

Unregister the callback function to be called when receiving a user command

Prototype	CORSTATUS **CorManUnregisterCallback**(CORSERVER *hServer*);
Description	Unregisters the callback function to be called when receiving a user command
Input	*hServer* Server handle.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorManRegisterCallback and CorManUserCmd

CorManUnregisterCallbackEx

Unregister the callback function for server related events

Prototype	CORSTATUS **CorManUnregisterCallbackEx**(void);
Description	Unregisters the callback function for server related events.
Input	None
Output	None
Return Value	CORSTATUS_TIMEOUT
See Also	CorManRegisterCallbackEx

CorManUnregisterHandle

Removes a handle from the local handle database

Prototype	CORSTATUS **CorManUnregisterHandle**(CORHANDLE *handle*);
Description	Removes a handle from the local handle database. This function must be used to remove a handle previously added by *CorManRegisterHandle*.
Input	*handle* Handle to unregister.
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorManRegisterHandle

CorManUserCmd

Send a user command to a remote server

Prototype	CORSTATUS **CorManUserCmd**(CORSERVER *hServer*, UINT32 *cmd*, void **inData*, UINT32 *inDataSize*, void **outData*, UINT32 *outDataSize*);	
Description	Sends a user command to a server. To receive a user command, use CorManRegisterCallback to register a callback function to be called when receiving a user command from a server. To unregister the callback function use CorManUnregisterCallback.	
Input	*hServer*	Server handle.
	cmd	User command number (0 .. 65536).
	inData	Input data.
	inDataSize	Input data size in bytes.
	outData	Output data.
	outDataSize	Output data size in bytes.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
Note	*inData* allows data to be sent along with the user command to the registered callback function; *outData* allows data to be received resulting from the execution of the user command. Both *inData* and *outData* can be specified as *NULL*.	
See Also	CorManRegisterCallback CorManUnregisterCallback	

CorManWaitForServerReady

Wait until a given server is ready

Prototype	CORSTATUS **CorManWaitForServerReady**(CORSERVER *hServer*, UINT32 *timeOut*);	
Description	Waits until a given server is ready. The function returns CORSTATUS_OK as soon as the server is ready, or with CORSTATUS_TIMEOUT if the *timeOut* seconds have elapsed.	
Input	*hServer*	Server handle
	timeOut	Maximum time (in seconds) to wait
Output	None	
Return Value	CORSTATUS_TIMEOUT	
See Also	CorManGetServerByIndex and CorManGetServerByName	

Transfer Module

The Transfer Module is responsible for moving data between various sources and destinations.

Capabilities

ID	Capability
0x00	*Reserved*
0x01	CORXFER_CAP_EVENT_TYPE
0x02	CORXFER_CAP_CROP_HORZ
0x03	CORXFER_CAP_CROP_LEFT_MIN
0x04	CORXFER_CAP_CROP_LEFT_MAX
0x05	CORXFER_CAP_CROP_LEFT_MULT
0x06	CORXFER_CAP_CROP_VERT
0x07	CORXFER_CAP_CROP_TOP_MIN
0x08	CORXFER_CAP_CROP_TOP_MAX
0x09	CORXFER_CAP_CROP_TOP_MULT
0x0a	CORXFER_CAP_CROP_WIDTH_MIN
0x0b	CORXFER_CAP_CROP_WIDTH_MAX
0x0c	CORXFER_CAP_CROP_WIDTH_MULT
0x0d	CORXFER_CAP_CROP_HEIGHT_MIN
0x0e	CORXFER_CAP_CROP_HEIGHT_MAX
0x0f	CORXFER_CAP_CROP_HEIGHT_MULT
0x10	CORXFER_CAP_SCALE_HORZ_METHOD
0x11	CORXFER_CAP_SCALE_HORZ_MIN
0x12	CORXFER_CAP_SCALE_HORZ_MAX
0x13	CORXFER_CAP_SCALE_HORZ_MULT
0x14	CORXFER_CAP_SCALE_HORZ_MIN_FACTOR
0x15	CORXFER_CAP_SCALE_HORZ_MAX_FACTOR
0x16	CORXFER_CAP_SCALE_VERT_METHOD
0x17	CORXFER_CAP_SCALE_VERT_MIN
0x18	CORXFER_CAP_SCALE_VERT_MAX
0x19	CORXFER_CAP_SCALE_VERT_MULT
0x1a	CORXFER_CAP_SCALE_VERT_MIN_FACTOR
0x1b	CORXFER_CAP_SCALE_VERT_MAX_FACTOR
0x1c	CORXFER_CAP_COUNTER_STAMP_EVENT_TYPE

0x1d	CORXFER_CAP_MAX_XFER_SIZE
0x1e	CORXFER_CAP_SCALE_HORZ
0x1f	CORXFER_CAP_SCALE_VERT
0x20	CORXFER_CAP_FLIP
0x21	CORXFER_CAP_NB_INT_BUFFERS
0x22	CORXFER_CAP_EVENT_COUNT_SOURCE
0x23	CORXFER_CAP_MAX_FRAME_COUNT
	Reserved
0x25	CORXFER_CAP_COUNTER_STAMP_AVAILABLE
0x26	CORXFER_CAP_COUNTER_STAMP_TIME_BASE
0x27	CORXFER_CAP_COUNTER_STAMP_MAX
0x28	CORXFER_CAP_CYCLE_MODE
0x29	CORXFER_CAP_FLATFIELD
0x2c	CORXFER_CAP_FLATFIELD_CYCLE_MODE
0x2d	CORXFER_CAP_PROCESSING_MODE

CORXFER_CAP_COUNTER_STAMP_AVAILABLE

Description	Specifies if the transfer resource supports a counter stamp
Type	UINT32
Values	TRUE, Counter stamp is available. FALSE, Counter stamp is not available.

CORXFER_CAP_COUNTER_STAMP_MAX

Description	Specifies the maximum value for the counter stamp
Type	UINT32
Values	
Note	Only valid if CORXFER_CAP_COUNTER_STAMP_AVAILABLE is TRUE

CORXFER_CAP_COUNTER_STAMP_TIME_BASE

Description	Specifies the counter stamp time base values available
Type	UINT32
Values	CORXFER_VAL_TIME_BASE_US the time base is in micro-seconds. CORXFER_VAL_TIME_BASE_MS the time base is in mili-seconds. CORXFER_VAL_TIME_BASE_LINE_TRIGGER the time base is in line trigger. CORXFER_VAL_TIME_BASE_LINE the time base is in line valid.
Note	The returned value is the ORed combination of the valid values. Only valid if CORXFER_CAP_COUNTER_STAMP_AVAILABLE is TRUE.

CORXFER_CAP_COUNTER_STAMP_EVENT_TYPE

Description Specifies the event type(s) that will perform a counter stamp of the transfer destination.

Type UINT32

Values
CORXFER_VAL_EVENT_TYPE_START_OF_FIELD
Counter stamped with the count at which the start of the field occurred.
CORXFER_VAL_EVENT_TYPE_START_OF_ODD
Counter stamped with the count at which the start of the odd field occurred
CORXFER_VAL_EVENT_TYPE_START_OF_EVEN
Counter stamped with the count at which the start of the even field occurred
CORXFER_VAL_EVENT_TYPE_END_OF_FIELD
Counter stamped with the count at which the end of the field occurred
CORXFER_VAL_EVENT_TYPE_END_OF_ODD
Counter stamped with the count at which the end of the odd field occurred
CORXFER_VAL_EVENT_TYPE_END_OF_EVEN
Counter stamped with the count at which the end of the even field occurred
CORXFER_VAL_EVENT_TYPE_START_OF_FRAME
Counter stamped with the count at which the start of the frame occurred.
CORXFER_VAL_EVENT_TYPE_END_OF_FRAME
Counter stamped with the count at which the end of the frame occurred.
CORXFER_VAL_EVENT_TYPE_END_OF_LINE
Counter stamped with the count at which the end of the line occurred
CORXFER_VAL_EVENT_TYPE_END_OF_NLINES
Counter stamped with the count at which the end of *n* lines occurred.

Note The returned value is the ORed combination of the valid values.

CORXFER_CAP_CROP_HEIGHT_MAX

Description Specifies the maximum supported cropping height value (in lines) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_HEIGHT_MIN

Description Specifies the minimum supported cropping height value (in lines) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_HEIGHT_MULT

Description Specifies the supported cropping height granularity (in lines) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_HORZ

Description Specifies if the transfer device supports horizontal cropping of the transferred data.

Type UINT32

Values TRUE, Horizontal cropping is supported.
FALSE, Horizontal cropping is not supported.

CORXFER_CAP_CROP_LEFT_MAX

Description Specifies the maximum supported left side cropping value (in pixels) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_LEFT_MIN

Description Specifies the minimum supported left side cropping value (in pixels) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_LEFT_MULT

Description Specifies the supported left side cropping granularity (in pixels) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_TOP_MAX

Description Specifies the maximum supported cropping value (in lines) for the top of the transferred data.

Type UINT32

CORXFER_CAP_CROP_TOP_MIN

Description Specifies the minimum supported cropping value (in lines) for the top of the transferred data

Type UINT32

CORXFER_CAP_CROP_TOP_MULT

Description Specifies the supported cropping granularity (in lines) for the top of the transferred data.

Type UINT32

CORXFER_CAP_CROP_VERT

Description Specifies if the transfer device supports vertical cropping of the transferred data.

Type UINT32

Values TRUE, Vertical cropping is supported.
FALSE, Vertical cropping is not supported.

CORXFER_CAP_CROP_WIDTH_MAX

Description Specifies the maximum supported cropping width value (in pixels) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_WIDTH_MIN

Description Specifies the minimum supported width cropping value (in pixels) of the transferred data.

Type UINT32

CORXFER_CAP_CROP_WIDTH_MULT

Description	Specifies the supported cropping granularity (in pixels) for the width of the transferred data.
Type	UINT32

CORXFER_CAP_CYCLE_MODE

Description	Specifies the different cycle modes supported by the transfer module for the current transfer level.
Type	UINT32
Values	See CORXFER_PRM_CYCLE_MODE.
Note	Use the macro CORXFER_IS_CYCLE_MODE_SUPPORTED to test for the valid cycle mode.

Example Code	#define CORXFER_CAP_CYCLE_MODE CORXFER_CAP(40, 4)
	
	// use this macro to check if a cycle mode is supported
	// cap is the capability of the transfer level
	// cycleMode is a CORXFER_VAL_CYCLE_MODE_XXXX value
	#define CORXFER_IS_CYCLE_MODE_SUPPORTED(cap,cycleMode) (((cap)&(1 << ((cycleMode) & 31))) != 0)

CORXFER_CAP_EVENT_COUNT_SOURCE

Description	Specifies the possible handle types that can increase the event count for each call to the transfer callback function.
Type	UINT32
Values	See CORXFER_PRM_EVENT_TYPECOUNT_SOURCE.
Note	The returned value is the ORed combination of the valid values.

CORXFER_CAP_EVENT_TYPE

Description	Specifies the event type(s) that can be registered.
Type	UINT32
Values	See CORXFER_PRM_EVENT_TYPE.
Note	The returned value is the ORed combination of the valid values.

CORXFER_CAP_FLATFIELD

Description	Specifies the different flatfield modes supported by the transfer module.
Type	BOOL
Values	CORXFER_VAL_FLATFIELD_NOT_SUPPORTED (0x00000000) CORXFER_VAL_FLATFIELD_SUPPORTED (0x00000001) See CORXFER_PRM_FLATFIELD_NUMBER and CORXFER_PRM_FLATFIELD_CYCLE_MODE

CORXFER_CAP_FLATFIELD_CYCLE_MODE

Description Specifies the different flatfield cycle modes supported by the transfer module.

Type UINT32

Values CORXFER_VAL_FLATFIELD_CYCLE_MODE_NOT_SUPPORTED (0x00000000) not suported

CORXFER_VAL_FLATFIELD_CYCLE_MODE_OFF (0x00000001) only 1 flatfield is used, see CORXFER_PRM_FLATFIELD_NUMBER

CORXFER_VAL_FLATFIELD_CYCLE_MODE_AUTOMATIC (0x000000002) cycle throught all flatfields created, the prm CORXFER_PRM_FLATFIELD_NUMBER is then read-only.

CORXFER_VAL_FLATFIELD_CYCLE_MODE_MANUAL (0x000000004), use the flatfield number specified with prm CORXFER_PRM_FLATFIELD_NUMBER

See CORXFER_PRM_FLATFIELD_CYCLE_MODE.

Note The returned value is the ORed combination of the valid values.

CORXFER_CAP_FLIP

Description Specifies the different flipping modes supported by the transfer module.

Type UINT32

Values See CORXFER_PRM_FLIP

Note The returned value is the ORed combination of the valid values.

CORXFER_CAP_MAX_FRAME_COUNT

Description Specifies the maximum number of frames that can be acquired in a sequential grab, that is, when calling the CorXferStart function with a count argument not equal to CORXFER_CONTINUOUS

Type UINT32

CORXFER_CAP_MAX_XFER_SIZE

Description Specifies the maximum number of bytes the transfer device can transfer.

Type UINT32

CORXFER_CAP_NB_INT_BUFFERS

Description Gets the internal buffer capability of the board

Type UINT32

Values CORXFER_VAL_NB_INT_BUFFERS_NONE, non available.
CORXFER_VAL_NB_INT_BUFFERS_MANUAL, created by the application. The user must create the internal buffer by using a handle to the board, and then append the buffer to the Xfer module.
CORXFER_VAL_NB_INT_BUFFERS_AUTO, automaticaly created. The internal buffers are not accessible by the user.

CORXFER_CAP_PROCESSING_MODE

Description	Gets the processing capability of the board.
Type	UINT32
Values	CORXFER_VAL_PROCESSING_MODE_NOT_SUPPORTED , not available. CORXFER_VAL_PROCESSING_MODE_1 or CORXFER_VAL_PROCESSING_MODE_2 or CORXFER_VAL_PROCESSING_MODE_3 or CORXFER_VAL_PROCESSING_MODE_4
Note	The processing behavior is specific to the board driver. See the board user's manual for more information about the processing.

CORXFER_CAP_SCALE_HORZ

Description	Specifies if the transfer device supports horizontal scaling.
Type	UINT32
Values	TRUE, Horizontal scaling is available. FALSE, Horizontal scaling is not available.

CORXFER_CAP_SCALE_HORZ_MAX

Description	Specifies the maximum number of pixels that can be output by the transfer resource.
Type	UINT32

CORXFER_CAP_SCALE_HORZ_MAX_FACTOR

Description	Specifies the maximum horizontal upscaling ratio supported by the transfer resource.
Type	UINT32
Notes	The ratio is equal to CORXFER_CAP_SCALE_HORZ_MAX_FACTOR/ CORXFER_VAL_SCALE_FACTOR

CORXFER_CAP_SCALE_HORZ_METHOD

Description	Specifies the different horizontal scaling methods supported by the transfer resource.
Type	UINT32
Values	CORXFER_VAL_SCALE_DISABLE, The horizontal scaling can be disabled. CORXFER_VAL_SCALE_SIMPLE, Horizontal scaling drops pixels. CORXFER_VAL_SCALE_INTERPOLATION, Horizontal scaling interpolates pixels. CORXFER_VAL_SCALE_POW2, Horizontal scaling factor must be a power of 2.
Note	The returned value is the ORed combination of the valid values.

CORXFER_CAP_SCALE_HORZ_MIN

Description	Specifies the minimum number of pixels that can be output by the transfer resource.
Type	UINT32

CORXFER_CAP_SCALE_HORZ_MIN_FACTOR

Description Specifies the minimum horizontal downscaling ratio supported by the transfer resource.

Type UINT32

Note The ratio is equal to 1/(COXFER_CAP_SCALE_HORZ_MIN_FACTOR/ CORXFER_VAL_SCALE_FACTOR).

CORXFER_CAP_SCALE_HORZ_MULT

Description Specifies the granularity (in pixels) that can be output by the transfer resource.

Type UINT32

CORXFER_CAP_SCALE_VERT

Description Specifies if the transfer resource supports vertical scaling.

Type UINT32

Values TRUE, Vertical scaling is available.
FALSE, Vertical scaling is not available.

CORXFER_CAP_SCALE_VERT_MAX

Description Specifies the maximum number of lines that can be output by the transfer resource.

Type UINT32

CORXFER_CAP_SCALE_VERT_MAX_FACTOR

Description Specifies the maximum vertical upscaling ratio supported by the transfer resource.

Type UINT32

Note The ratio is equal to:
CORXFER_CAP_SCALE_VERT_MAX_FACTOR/ CORXFER_VAL_SCALE_FACTOR.

CORXFER_CAP_SCALE_VERT_METHOD

Description Specifies the different vertical scaling methods supported by the transfer resource.

Type UINT32

Values CORXFER_VAL_SCALE_DISABLE, The vertical scaling can be disabled.
CORXFER_VAL_SCALE_SIMPLE, Vertical scaling drops lines.
CORXFER_VAL_SCALE_INTERPOLATION, Vertical scaling interpolates lines.
CORXFER_VAL_SCALE_POW2, Vertical scaling factor must be a power of 2.

Note The returned value is the ORed combination of the valid values.

CORXFER_CAP_SCALE_VERT_MIN

Description Specifies the minimum number of lines that can be output by the transfer resource.

Type UINT32

CORXFER_CAP_SCALE_VERT_MIN_FACTOR

Description Specifies the minimum vertical downscaling ratio supported by the transfer resource.

Type UINT32

Note The ratio is equal to:
1/(CORXFERCAP_SCALE_VERT_MIN_FACTOR/ CORXFER_VAL_SCALE_FACTOR).

CORXFER_CAP_SCALE_VERT_MULT

Description Specifies the vertical granularity (in lines) that can be output by the transfer resource.

Type UINT32

Parameters

ID	Parameter	Attribute
0x00	*Reserved*	----------
0x01	*Reserved*	----------
0x02	CORXFER_PRM_CROP_LEFT	Read/Write
0x03	CORXFER_PRM_CROP_TOP	Read/Write
0x04	CORXFER_PRM_CROP_WIDTH	Read/Write
0x05	CORXFER_PRM_CROP_HEIGHT	Read/Write
0x06	CORXFER_PRM_SCALE_HORZ	Read/Write
0x07	CORXFER_PRM_SCALE_VERT	Read/Write
0x08	CORXFER_PRM_SCALE_HORZ_METHOD	Read/Write
0x09	CORXFER_PRM_SCALE_VERT_METHOD	Read/Write
0x0a	CORXFER_PRM_EVENT_TYPE	Read Only
0x0b	CORXFER_PRM_EVENT_COUNT	Read Only
0x0c	CORXFER_PRM_START_MODE	Read/Write
0x0d	CORXFER_PRM_TIMEOUT	Read/Write
0x0e	CORXFER_PRM_CYCLE_MODE	Read/Write
0x0f	CORXFER_PRM_EVENT_SERVER	Read Only
0x10	CORXFER_PRM_EVENT_CALLBACK	Read Only
0x11	CORXFER_PRM_EVENT_CONTEXT	Read Only
0x12	CORXFER_PRM_FLIP	Read/Write
0x13	CORXFER_PRM_NB_INT_BUFFERS	Read/Write
0x14	CORXFER_PRM_EVENT_COUNT_SOURCE	Read/Write
0x15	*Reserved*	
0x16	CORXFER_PRM_COUNTER_STAMP_BASE	Read/Write
0x17	CORXFER_PRM_FLATFIELD_NUMBER	Read/Write
0x1a	CORXFER_PRM_FLATFIELD_CYCLE_MODE	Read/Write

0x1b	CORXFER_PRM_PROCESSING_MODE	Read/Write

CORXFER_PRM_COUNTER_STAMP_BASE

Description Sets the counter stamp time base.

Type UINT32

Values CORXFER_VAL_TIME_BASE_US the time base is in micro-second

CORXFER_VAL_TIME_BASE_MS the time base is in mili-second
CORXFER_VAL_TIME_BASE_LINE_TRIGGER the time base is in line trigger
CORXFER_VAL_TIME_BASE_LINE the time base is in line valid

Note Only valid if CORXFER_CAP_COUNTER_STAMP_AVAILABLE is TRUE

See CORXFER_CAP_COUNTER_STAMP_TIME_BASE

CORXFER_PRM_CROP_HEIGHT

Description Cropped height of the transferred data (in lines).

Type UINT32

CORXFER_PRM_CROP_LEFT

Description Number of pixels to crop from the left side of the transferred data.

Type UINT32

CORXFER_PRM_CROP_TOP

Description Number of lines to crop from the top of the transferred data.

Type UINT32

CORXFER_PRM_CROP_WIDTH

Description Cropped width of the transferred data (in pixels).

Type UINT32

CORXFER_PRM_CYCLE_MODE

Description Sets the mode used by the transfer device to specify which buffer gets the next data transfer.

Type UINT32

Values The available modes differ by the way in which they specify which buffer gets the next data transfer.

The empty state refers to the case where buffer data has been completely processed and may be overwritten. It is set by application code as soon as it has finished processing buffer data.

The full state refers to the case where buffer data has not been processed since its latest data transfer. It is set by the transfer device as soon as a data transfer has completed.

The current buffer is the one in which the latest data transfer occurred.

The next buffer is the one immediately after the current buffer, with wraparound to the first buffer at the end of the list.

The trash buffer is defined as the last buffer in the list for the WITH_TRASH modes only. Its state is always considered to be empty by the transfer device.

CORXFER_VAL_CYCLE_MODE_ASYNCHRONOUS (0x00000000),
Always transfer to the next buffer regardless of its state.

CORXFER_VAL_CYCLE_MODE_SYNCHRONOUS (0x00000001),
If next buffer is empty, then transfer to next buffer, otherwise, transfer to current buffer.

CORXFER_VAL_CYCLE_MODE_SYNCHRONOUS_WITH_TRASH (0x00000002),
If next buffer is empty, then transfer to the next buffer, otherwise, transfer to the trash buffer. Repeat transferring to the trash buffer as long as the next buffer is full.

CORXFER_VAL_CYCLE_MODE_OFF (0x00000003),
Always transfer to the current buffer.

CORXFER_VAL_CYCLE_MODE_NEXT_EMPTY (0x00000004),
If next buffer is empty, then transfer to next buffer, otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to current buffer.

CORXFER_VAL_CYCLE_MODE_SYNCHRONOUS_NEXT_EMPTY_WITH_TRASH (0x00000005),
If next buffer is empty, then transfer to next buffer, otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to trash buffer. Repeat transferring to the trash buffer as long as there is no empty buffer in the list.

See Also CORBUFFER_PRM_STATE

CORXFER_PRM_EVENT_CALLBACK

Description Callback registered using the function CorXferRegisterCallback for the current item selected.

Type PCORCALLBACK

Values Pointer to the callback function registered.

Note This parameter is read-only.

CORXFER_PRM_EVENT_CONTEXT

Description Context pointer registered using the function CorXferRegisterCallback for the current item selected..

Type void *

Values Pointer to the context.

Note This parameter is read-only.

CORXFER_PRM_EVENT_COUNT

Description Number of events that have occurred for the current item selected since a callback function was registered using the CorXferRegisterCallback function.

Type UINT32

Note This parameter is read-only.

CORXFER_PRM_EVENT_COUNT_SOURCE

Description Handle type that increases the event count for each call to the transfer callback function.

Type UINT32

Values CORXFER_VAL_EVENT_COUNT_SOURCE_DST (0x0000001)
The event count is associated with the destination handle, which is usually a buffer. This means that all buffers in a list have their own event count.

CORXFER_VAL_EVENT_COUNT_SOURCE_SRC (0x0000002)
The event count is associated with the source handle which is usually an acquisition device. This means that the count increases at each acquired frame.

CORXFER_PRM_EVENT_SERVER

Description Server to which an event notification through a callback function will be made.

Type CORSERVER

Values Server handle.

Note This parameter is read-only.

CORXFER_PRM_EVENT_TYPE

Description Event to be signaled while a transfer is in progress.

Type UINT32

Values CORXFER_VAL_EVENT_TYPE_START_OF_FIELD (0x00010000)
Call the callback function at the start of an odd or even field.
CORXFER_VAL_EVENT_TYPE_START_OF_ODD (0x0020000)
Call the callback function at the start of an odd field.
CORXFER_VAL_EVENT_TYPE_START_OF_EVEN (0x00040000)
Call the callback function at the start of an even field.
CORXFER_VAL_EVENT_TYPE_START_OF_FRAME (0x00080000)
Call the callback at the start of a frame.
CORXFER_VAL_EVENT_TYPE_END_OF_FIELD (0x00100000)
Call the callback function at the end of an odd or even field.
CORXFER_VAL_EVENT_TYPE_END_OF_ODD (0x00200000)
Call the callback function at the end of an odd field.
CORXFER_VAL_EVENT_TYPE_END_OF_EVEN (0x00400000)
Call the callback function at the end of an even field.
CORXFER_VAL_EVENT_TYPE_END_OF_FRAME (0x00800000)
Call the callback function at the end of a frame.
CORXFER_VAL_EVENT_TYPE_END_OF_LINE (0x01000000)
Call the callback function at end of line *n*.

CORXFER_VAL_EVENT_TYPE_END_OF_NLINES (0x02000000)
Call the callback function at end of *n* lines.
CORXFER_VAL_EVENT_TYPE_END_OF_TRANSFER (0x04000000)
Call the callback function at the end of a transfer.

CORXFER_VAL_EVENT_TYPE_LINE_UNDERRUN (0x08000000)
Call the callback function if during a transfer the number of active pixels per line received from a video source is smaller than requested.

CORXFER_VAL_EVENT_TYPE_FIELD_UNDERRUN (0x10000000)
Call the callback function if during a transfer the number of active lines per field received from a video source is smaller than requested.

Note The values may be ORed if more than one event is desired.

CORXFER_PRM_FLATFIELD_NUMBER

Description The flatfield number used for the current source and destination.

Type UINT32

CORXFER_PRM_FLATFIELD_CYCLE_MODE

Description Sets the flatfield cycle mode used by the transfer device to specify which flatfield buffer is used for the data transfer.

Type UINT32

Values CORXFER_VAL_FLATFIELD_CYCLE_MODE_OFF (0x00000001)
Only one flatfield is used. See CORXFER_PRM_FLATFIELD_NUMBER.

CORXFER_VAL_FLATFIELD_CYCLE_MODE_AUTOMATIC (0x000000002) Cycle through all flatfields created. The PRM CORXFER_PRM_FLATFIELD_NUMBER is then read-only.

CORXFER_VAL_FLATFIELD_CYCLE_MODE_MANUAL (0x000000004) Use the flatfield number specified with PRM CORXFER_PRM_FLATFIELD_NUMBER.

See CORXFER_CAP_FLATFIELD_CYCLE_MODE.

CORXFER_PRM_FLIP

Description The transfer module Flipping Mode control.

Type UINT32

Values CORXFER_VAL_FLIP_OFF (0x00000000), Will not flip incoming lines and frames.

CORXFER_VAL_FLIP_HORZ (0x00000001), Will flip incoming lines for the current destination buffer; i.e. the rightmost pixels become the leftmost pixels.

CORXFER_VAL_FLIP_VERT (0x00000002), Will flip incoming frames for the current destination buffer; i.e. the bottom lines become the top lines.

Limits This value must match one of the supported capabilities of the transfer module given by CORXFER_CAP_FLIP

CORXFER_PRM_NB_INT_BUFFERS

Description Sets the number of internal buffers on the frame grabber that will be used when acquiring images.

Type UINT32

CORXFER_PRM_PROCESSING_MODE

Description Sets the processing mode of the board.

Type UINT32

Values CORXFER_VAL_PROCESSING_MODE_NOT_SUPPORTED = 0, not available.
CORXFER_VAL_PROCESSING_MODE_1 = processing mode 1 (board specific)
CORXFER_VAL_PROCESSING_MODE_2 = processing mode 2 (board specific)
CORXFER_VAL_PROCESSING_MODE_3 = processing mode 3 (board specific)
CORXFER_VAL_PROCESSING_MODE_4 = processing mode 4 (board specific)

Note A processing mode can only be set before calling CorXferConnect

CORXFER_PRM_SCALE_HORZ

Description Number of pixels per line to be output by the transfer.

Type UINT32

CORXFER_PRM_SCALE_HORZ_METHOD

Description Horizontal scaling method.

Type UINT32

Values CORXFER_VAL_SCALE_DISABLE, Disable horizontal scaling.
CORXER_VAL_SCALE_SIMPLE, Horizontal scaling drops pixels.
CORXFER_VAL_SCALE_INTERPOLATION, Horizontal scaling interpolates pixels.

CORXFER_PRM_SCALE_VERT

Description Number of lines per frame to be output by the transfer.

Type UINT32

CORXFER_PRM_SCALE_VERT_METHOD

Description Vertical scaling method.

Type UINT32

Values CORXFER_VAL_SCALE_DISABLE, Disable vertical scaling.
CORXFER_VAL_SCALE_SIMPLE, Vertical scaling drops lines.
CORXFER_VAL_SCALE_INTERPOLATION, Vertical scaling interpolates lines.

CORXFER_PRM_START_MODE

Description Controls the behavior of CorXferStart function when called.

Type UINT32

Values	CORXFER_VAL_START_MODE_ASYNCHRONOUS (0x00000000) When starting a transfer, CorXferStart returns immediately without waiting for the transfer to begin. CORXFER_VAL_START_MODE_SYNCHRONOUS (0x00000001) When starting a single frame transfer, CorXferStart returns only when the transfer has been completed. CORXFER_VAL_START_MODE_HALF_ASYNCHRONOUS (0x00000002) If a transfer is currently in progress when starting a new single frame transfer, CorXferStart will wait for the first transfer to finish and then start the transfer. It then returns immediately. CORXFER_VAL_START_MODE_SEQUENTIAL (0x00000003) If a multi-level transfer is defined (i.e. acquisition to on-board memory to host memory), the transfer process will wait until all frames in the sequence are in the on-board memory before sending them to the host memory.
Note	This parameter has no effect when starting a transfer in continuous mode.

CORXFER_PRM_TIMEOUT

Description	Specifies the maximum number of milliseconds to wait for a transfer to finish.
Type	UINT32

CORXFER_DESC Structure Definition

```
// CORXFER_DESC Structure Definition

typedef struct
{
    UINT32 frame;  //has the following values:
    //CORXFER_VAL_FRAME_INTERLACED
    //CORXFER_VAL_FRAME_NON_INTERLACED

    UINT32 fieldOrder; //has the following values if frame is interlaced:
    //CORXFER_VAL_FIELD_ORDER_ODD_EVEN
    //CORXFER_VAL_FIELD_ORDER_EVEN_ODD
    //CORXFER_VAL_FIELD_ORDER_ANY_ORDER
    UINT32 widthByte;  //line width of the frame in bytes
    UINT32 height; //frame height in lines
    UINT32 incByte;    //the stride between two lines in bytes
    //(even if the frame is interlaced,
    //incByte should usually be equal to widthByte)

} CORXFER_DESC, *PCORXFER_DESC;   //any parameter with a value of 0 is ignored


//the source is then interrogated to retrieve the
//corresponding information when possible)
```

Functions

Function	Description
CorXferAbort	*Aborts transfer asynchronously for a transfer resource*
CorXferAppend	*Appends item to the transfer description list of a transfer resource. Source and destination resource have a single port.*
CorXferAppendEx	*Appends item to the transfer description list of a transfer resource. Source and destination resource can have more than one port.*
CorXferConnect	*Builds the transfer description list and locks resources of a transfer resource*
CorXferDisconnect	*Frees resources used by a transfer resource*
CorXferFree	*Frees handle to a transfer resource*
CorXferGetCap	*Gets transfer capability value from a transfer resource*
CorXferGetPrm	*Gets transfer parameter value from a transfer resource*
CorXferLinkCounterStamp	*Link a counter resource to a transfer resource and specify which event will be stamped.*
CorXferUnlinkCounterStamp	*Unlink a counter resource from the currently selected item.*
CorXferNew	*Creates in the specified server's memory a new transfer resource handle. Source and destination resource have a single port.*
CorXferNewEx	*Creates in the specified server's memory a new transfer resource handle. Source and destination resource can have more than one port.*
CorXferRegisterCallback	*Registers callback function for a transfer resource*
CorXferReset	*Resets a transfer resource*
CorXferResetModule	*Resets the resources associated with the server's transfer device(s)*
CorXferSelect	*Selects an item as the current item of a transfer resource. Source and destination resource have a single port.*
CorXferSelectEx	*Selects an item as the current item of a transfer resource. Source and destination resource can have more than one port.*
CorXferSetPrm	*Sets a simple transfer parameter of a transfer resource*
CorXferSetPrmEx	*Sets a complex transfer parameter of a transfer resource*
CorXferStart	*Starts transfer for a transfer resource*
CorXferStop	*Stops transfer synchronously for a transfer resource*
CorXferUnregisterCallback	*Unregisters callback function for a transfer resource*
CorXferWait	*Waits until end of transfer or until timeout for a transfer resource*

CorXferAbort

Stop transfer asynchronously for a transfer resource

Prototype CORSTATUS **CorXferAbort**(CORXFER *hXfer*);

Description	Stops transfer asynchronously for a transfer resource. On return, transfer is finished but part of the last transferred frame may be corrupted.	
Input	*hXfer*	Transfer resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_XFER_NOT_CONNECTED	

CorXferAppend

Append item to the transfer description list of a transfer resource

Prototype	CORSTATUS **CorXferAppend**(CORXFER *hXfer*, CORHANDLE *hSrc*, CORHANDLE *hDst*, CORXFER_DESC**pDesc*);	
Description	Appends item to the transfer description list of a transfer resource. The new appended item (hSrc, hDst) becomes the current item. If the source and/or the destination resource have more than one port, port number 0 will be used.	
Input	*hXfer*	Transfer resource handle
	hSrc	Resource handle (source)
	hDst	Resource handle (destination)
	pDesc	Transfer description structure. See CORXFER_DESC Structure Definition.
Output	None	
Return Value	CORSTATUS_INCOMPATIBLE_BUFFER, CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY, CORSTATUS_ROUTING_NOT_IMPLEMENTED and CORSTATUS_XFER_CANT_CYCLE	
Note	If *pDesc* is specified as NULL, automatic source format detection will be used to provide all the information needed to specify the transfer.	
	When transferring to a buffer resource, the specified buffer must have been created using the CORBUFFER_VAL_TYPE_CONTIGUOUS or CORBUFFER_VAL_TYPE_SCATTER_GATHER type. Otherwise, CORSTATUS_INCOMPATIBLE_BUFFER is returned.	
	If there is not enough local memory to add the (hSrc, hDst) items to the transfer description list, CORSTATUS_NO_MEMORY is returned.	
	If there is no available transfer path for the newly added (hSrc, hDst) item, CORSTATUS_ROUTING_NOT_IMPLEMENTED is returned.	
	If for the newly added (hSrc, hDst) item in the transfer description list, the hDst resource location has already been added and the transfer resource does not support cycle transfer for this type of resource, CORSTATUS_XFER_CANT_CYCLE is returned.	
See also	CorXferAppendEx	

CorXferAppendEx

Append item to the transfer description list of a transfer resource

Prototype	CORSTATUS **CorXferAppendEx**(CORXFER *hXfer*, CORHANDLE *hSrc*, UINT32 *srcPort*, CORHANDLE *hDst*, UINT32 *dstPort*, CORXFER_DESC**pDesc*);	
Description	Appends item to the transfer description list of a transfer resource. Source and destination resource can have more than one port. The new appended item (hSrc, hDst) becomes the current item.	
Input	*hXfer*	Transfer resource handle
	hSrc	Resource handle (source)
	srcPort	Source port number
	hDst	Resource handle (destination)
	dstPort	Destination port number
	pDesc	Transfer description structure. See CORXFER_DESC Structure Definition.
Output	None	
Return Value	CORSTATUS_INCOMPATIBLE_BUFFER, CORSTATUS_INVALID_HANDLE CORSTATUS_NO_MEMORY, CORSTATUS_ROUTING_NOT_IMPLEMENTED and CORSTATUS_XFER_CANT_CYCLE	
Note	If *pDesc* is specified as NULL, automatic source format detection will be used to provide all the information needed to specify the transfer.	
	When transferring to a buffer resource, the specified buffer must have been created using the CORBUFFER_VAL_TYPE_CONTIGUOUS or CORBUFFER_VAL_TYPE_SCATTER_GATHER type. Otherwise, CORSTATUS_INCOMPATIBLE_BUFFER is returned.	
	If there is not enough local memory to add the (hSrc, hDst) items to the transfer description list, CORSTATUS_NO_MEMORY is returned.	
	If there is no available transfer path for the newly added (hSrc, hDst) item, CORSTATUS_ROUTING_NOT_IMPLEMENTED is returned.	
	If for the newly added (hSrc, hDst) item in the transfer description list, the hDst resource location has already been added and the transfer resource does not support cycle transfer for this type of resource, CORSTATUS_XFER_CANT_CYCLE is returned.	

CorXferConnect

Build the transfer description list and locks resources of a transfer resource

Prototype	CORSTATUS **CorXferConnect**(CORXFER *hXfer*);	
Description	Builds the transfer description list and locks resources for a transfer resource	
Input	*hXfer*	Transfer resource handle
Output	None	

Return Value	CORSTATUS_INCOMPATIBLE_SIZE, CORSTATUS_INVALID_HANDLE, CORSTATUS_NO_MEMORY, CORSTATUS_RESOURCE_IN_USE, CORSTATUS_ROUTING_IN_USE, CORSTATUS_XFER_EMPTY_LIST and CORSTATUS_XFER_MAX_SIZE
Note	If there are resources already in use that are needed to build the transfer description list, CORSTATUS_RESOURCE_IN_USE is returned.
	If for any one of the (hSrc, hDst) items in the transfer description list, the size of the source resource is larger than the size of the destination resource, CORSTATUS_INCOMPATIBLE_SIZE is returned.
	If for any one of the (hSrc, hDst) items in the transfer description list, the size in bytes of the source resource is larger than the CORXFER_CAP_MAX_XFER_SIZE capability, CORSTATUS_XFER_MAX_SIZE is returned.
	If the transfer description list is empty, CORSTATUS_XFER_EMPTY_LIST is returned.
	If there is a routing already in use that is needed to build the transfer description list, CORSTATUS_ROUTING_IN_USE is returned.
	If there is not enough local memory to build the local representation of the transfer description list, CORSTATUS_NO_MEMORY is returned.
See Also	CorXferDisconnect

CorXferDisconnect

Free resources used by a transfer resource

Prototype	CORSTATUS **CorXferDisconnect**(CORXFER *hXfer*);
Description	Frees resources used by a transfer resource.
Input	*hXfer* Transfer resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE
See Also	CorXferConnect

CorXferFree

Free handle to a transfer resource

Prototype	CORSTATUS **CorXferFree**(CORXFER *hXfer*);
Description	Frees handle to a transfer resource
Input	*hXfer* Transfer resource handle
Output	None
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_RESOURCE_IN_USE
See Also	CorXferNew

CorXferGetCap

Get transfer capability value from a transfer resource

Prototype	CORSTATUS **CorXferGetCap**(CORXFER *hXfer*, UINT32 *cap*, void **value*);	
Description	Gets transfer capability value for the current item of a transfer resource.	
Input	*hXfer*	Transfer resource handle
	cap	Transfer device capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_CAP_INVALID and CORSTATUS_INVALID_HANDLE	
Note	To select an item (hSrc, hDst) as the current item of a transfer resource, use CorXferSelect.	
See Also	CorXferSetPrm, CorXferSetPrmEx and CorXferSelect	

CorXferGetPrm

Get transfer parameter value from a transfer resource

Prototype	CORSTATUS **CorXferGetPrm**(CORXFER *hXfer*, UINT32 *prm*, void **value*);	
Description	Gets transfer parameter value from a transfer resource.	
Input	*hXfer*	Transfer resource handle
	prm	Transfer parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID and CORSTATUS_PRM_NOT_AVAILABLE	
Note	To select an item (hSrc, hDst) as the current transfer resource item, use CorXferSelect.	
See Also	CorXferSetPrm, CorXferSetPrmEx and CorXferSelect	

CorXferLinkCounterStamp

Link a counter resource to a transfer resource and specify the events to be stamped

Prototype	CORSTATUS **CorXferLinkCounterStamp**(CORXFER *hXfer*, CORCOUNTER *hCounter*, UNIT32 *eventType*);	
Description	Enables the counter stamp of the specified events into the current destination resource.	
Input	*hXfer*	Transfer resource handle
	hCounter	Counter resource handle
	eventType	Type of event to register. The destination will be counter stamped with the count where the specified event(s) occurred. See CORXFER_PRM_EVENT_TYPE.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

Notes	To select an item (hSrc, hDst) as the current transfer resource item, use CorXferSelect. To retrieve counter stamp information see CORBUFFER_PRM_COUNTER_STAMP.
See also	CorXferUnlinkCounterStamp

CorXferNew

Create in the specified server's memory a new transfer resource

Prototype	CORSTATUS **CorXferNew**(CORSERVER *hServer*, CORHANDLE *hSrc*, CORHANDLE *hDst*, CORXFER_DESC **pDesc*, CORXFER **hXfer*);	
Description	Creates in the specified server's memory a new transfer resource. The new item (hSrc, hDst) becomes the current item. If the source and/or the destination resource have more than one port, port number 0 will be used.	
Input	*hServer*	Server handle
	hSrc	Source of transfer
	hDst	Destination of transfer
	pDesc	Transfer description structure. See CORXFER_DESC Structure Definition
Output	*hXfer*	Transfer resource handle
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_NULL (if *hXfer* is NULL), CORSTATUS_INCOMPATIBLE_BUFFER, CORSTATUS_INVALID_HANDLE, CORSTATUS_NO_MEMORY, CORSTATUS_ROUTING_NOT_IMPLEMENTED and CORSTATUS_XFER_CANT_CYCLE	
Note	If *pDesc* is specified as NULL, automatic source format detection will provide all the information needed to specify the transfer.	
	The specified buffer must be created using CORBUFFER_VAL_TYPE_CONTIGUOUS or CORBUFFER_VAL_TYPE_SCATTER_GATHER when transferring to a buffer resource. Otherwise, CORSTATUS_INCOMPATIBLE_BUFFER is returned.	
	If there is not enough local memory to add the *hSrc* and *hDst* items to the transfer description list, CORSTATUS_NO_MEMORY is returned.	
	If there is no available transfer path for the newly added *hSrc* and *hDst* items, CORSTATUS_ROUTING_NOT_IMPLEMENTED is returned.	
	For the newly added *hSrc* and *hDst* items in the transfer description list, if the *hDst* resource has been previously added and the transfer resource does not support cycle transfers, CORSTATUS_XFER_CANT_CYCLE is returned.	
See Also	CorXferFree, CorXferAppend and CorXferNewEx	

CorXferNewEx

Create in the specified server's memory a new transfer resource

Prototype	CORSTATUS **CorXferNewEx**(CORSERVER *hServer*, CORHANDLE *hSrc*, UINT32 *srcPort*, CORHANDLE *hDst*, UINT32 *dstPort*, CORXFER_DESC **pDesc*, CORXFER **hXfer*);	
Description	Creates in the specified server's memory a new transfer resource. Source and destination resource can have more than one port. The new items *hSrc* and *hDst* are now the current items.	
Input	*hServer*	Server handle
	hSrc	Source of transfer
	srcPort	Source port number
	hDst	Destination of transfer
	dstPort	Destination port number
	pDesc	Transfer description structure. See CORXFER_DESC Structure Definition
Output	*hXfer*	Transfer resource handle
Return Value	CORSTATUS_ARG_INVALID, CORSTATUS_ARG_NULL (if *hXfer* is NULL), CORSTATUS_INCOMPATIBLE_BUFFER, CORSTATUS_INVALID_HANDLE, CORSTATUS_NO_MEMORYCORSTATUS_ROUTING_NOT_IMPLEMENTED and CORSTATUS_XFER_CANT_CYCLE	
Note	If *pDesc* is specified as NULL, automatic source format detection will provide all the information needed to specify the transfer.	
	The specified buffer must have been created using CORBUFFER_VAL_TYPE_CONTIGUOUS or CORBUFFER_VAL_TYPE_SCATTER_GATHER when transferring to a buffer resource. Otherwise, CORSTATUS_INCOMPATIBLE_BUFFER is returned.	
	If there is not enough local memory to add the *hSrc* and *hDst* items to the transfer description list, CORSTATUS_NO_MEMORY is returned.	
	If there is no available transfer path for the newly added *hSrc* and *hDst* items, CORSTATUS_ROUTING_NOT_IMPLEMENTED is returned.	
	For the newly added *hSrc* and *hDst* items in the transfer description list, if the *hDst* resource has been previously added and the transfer resource does not support cycle transfer for this type of resource, CORSTATUS_XFER_CANT_CYCLE is returned.	
See Also	CorXferFree and CorXferAppendEx	

CorXferRegisterCallback

Register callback function for a transfer resource

Prototype	CORSTATUS **CorXferRegisterCallback**(CORXFER *hXfer*, UINT32 *eventType*, PCORCALLBACK *callbackFct*, void **context*);	
Description	Registers callback function for the current item of a transfer resource.	
Input	*hXfer*	Transfer resource handle

	eventType	Type of event to register. The callback function will be called when the specified event(s) occur. See CORXFER_PRM_EVENT_TYPE.
	callbackFct	Callback function must be defined as follow: *CORSTATUS CCONV* *callback (void *context, UINT32 eventType, UINT32 eventCount);* When called, *context* contains the value specified at callback function registration; and *eventType* contains the event(s) that triggered the call to the callback function. The *eventCount* argument starts with a value of 1, and then is incremented either by the source or the destination handle for the transfer, as specified by the CORXFER_PRM_EVENT_COUNT_SOURCE parameter. This counter is reinitialized each time a new transfer is started by calling *CorXferStart*. In case the transfer resource can not keep up because there are too many events to be signaled, *eventCount* will take non-consecutive values, indicating that events have been lost. See the Data Types section for the PCORCALLBACK definition.
	context	Context pointer to be passed to the callback function when called
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *callbackFct* is NULL), CORSTATUS_INVALID_HANDLE CORSTATUS_NOT_AVAILABLE and CORSTATUS_RESOURCE_IN_USE	
Note	The values may be ORed if more than one event is desired. To select an item (hSrc, hDst) as the current item of a transfer resource, use CorXferSelect. When used, CORXFER_VAL_EVENT_TYPE_END_OF_LINE must be ORed with an *unsigned* integer representing the line on which the callback function has to be called, while CORXFER_VAL_EVENT_TYPE_END_OF_NLINES must be ORed with an *unsigned* integer representing the number of lines after which the callback function has to be called.	
See Also	CorXferUnregisterCallback, CorXferSelect and CORXFER_PRM_EVENT_COUNT_SOURCE	

CorXferReset

Reset a transfer resource

Prototype	CORSTATUS **CorXferReset**(CORXFER *hXfer*);	
Description	Deletes the existing transfer routing associated with the specified transfer resource.	
Input	*hXfer*	Transfer resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_NOT_ACCESSIBLE, CORSTATUS_RESOURCE_IN_USE and CORSTATUS_XFER_NOT_CONNECTED	

CorXferResetModule

Reset the resources associated with the server's transfer device(s)

Prototype	CORSTATUS **CorXferResetModule**(CORSERVER *hServer*);	
Description	Resets the resources associated with the server's transfer device(s). It will release all resources (handle, memory) currently allocated. When using this function, make certain that no other application is currently using any transfer device resources. This function should be used with caution.	
Input	*hServer*	Server handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	

CorXferSelect

Select an item as the current item of a transfer resource

Prototype	CORSTATUS **CorXferSelect**(CORXFER *hXfer*, CORHANDLE *hSrc*, CORHANDLE *hDst*, UINT32 *index*);	
Description	Selects an item as the current item of a transfer resource. If the source and/or the destination resource have more than one port, port number 0 will be used.	
Input	*hXfer*	Transfer resource handle
	hSrc	Resource handle (source)
	hDst	Resource handle (destination)
	index	Specifies which item to select. Valid values are in the range [0...n-1], where n is the number of items having the same source and destinations. Usually index is 0.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_RESOURCE_IN_USE	
See Also	CorXferSelectEx, CorXferGetCap, CorXferGetPrm, CorXferSetPrm, CorXferRegisterCallback, CorXferSetPrmEx and CorXferUnregisterCallback	

CorXferSelectEx

Select an item as the current item of a transfer resource

Prototype	CORSTATUS **CorXferSelectEx**(CORXFER *hXfer*, CORHANDLE *hSrc*, UINT32 *srcPort*, CORHANDLE *hDst*, UINT32 *dstPort*, UINT32 *index*);	
Description	Selects an item as the current item of a transfer resource. Source and destination resource can have more than one port.	
Input	*hXfer*	Transfer resource handle
	hSrc	Resource handle (source)
	srcPort	Source port number
	hDst	Resource handle (destination)
	dstPort	Destination port number

	index	Specifies which item to select. Valid values are in the range [0...n-1], where n is the number of items having the same source and destinations. Usually, index is 0.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_RESOURCE_IN_USE	
See Also	CorXferGetCap, CorXferGetPrm, CorXferSetPrm, CorXferRegisterCallback, CorXferSetPrmEx and CorXferUnregisterCallback	

CorXferSetPrm

Set a simple transfer parameter of a transfer resource

Prototype	CORSTATUS **CorXferSetPrm**(CORXFER *hXfer*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple transfer parameter for the current item of a transfer resource.	
Input	*hXfer*	Transfer resource handle
	prm	Transfer parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID_VALUE, CORSTATUS_PRM_NOT_AVAILABLE and CORSTATUS_PRM_READ_ONLY	
Note	A simple parameter fits inside an UINT32. If the parameter is complex, use CorXferSetPrmEx. To select an item (hSrc, hDst) as the current item of a transfer resource, use CorXferSelect.	
See Also	CorXferSetPrm, CorXferSetPrmEx and CorXferSelect	

CorXferSetPrmEx

Set a complex transfer parameter of a transfer resource

Prototype	CORSTATUS **CorXferSetPrmEx**(CORXFER *hXfer*, UINT32 *prm*, const void **value*);	
Description	Sets a complex transfer parameter for the current item of a transfer resource.	
Input	*hXfer*	Transfer resource handle
	prm	Transfer parameter to set
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL) CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID_VALUE, CORSTATUS_PRM_NOT_AVAILABLE and CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is greater than an UINT32. If the parameter size is UINT32, either CorXferSetPrm or CorXferSetPrmEx can be used. To select an item (hSrc, hDst) as the current item of a transfer resource, use CorXferSelect.	
See Also	CorXferGetPrm, CorXferSetPrm and CorXferSelect	

CorXferStart

Start transfer for a transfer resource

Prototype	CORSTATUS **CorXferStart**(CORXFER *hXfer*, UINT32 *count*);	
Description	Starts transfer for a transfer resource	
Input	*hXfer*	Transfer object handle
	count	Numerical value within the range [1...CORXFER_CAP_MAX_FRAME_COUNT] or CORXFER_CONTINUOUS.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_NOT_ACCESSIBLE, CORSTATUS_RESOURCE_IN_USE, CORSTATUS_XFER_NOT_CONNECTED, CORSTATUS_TIMEOUT and CORSTATUS_ARG_INVALID	
Note	Before calling CorXferStart, the transfer resource has to be connected to the hardware by calling CorXferConnect; otherwise, CORSTATUS_XFER_NOT_CONNECTED is returned.	
	When calling CorXferStart, if a transfer is still in progress and the CORXFER_PRM_START_MODE parameter has been set to CORXFER_VAL_START_MODE_ASYNCHRONOUS, then CORSTATUS_RESOURCE_IN_USE is returned.	
	When calling CorXferStart, if a transfer is still in progress and CORXFER_PRM_START_MODE parameter has been set to CORXFER_VAL_START_MODE_SYNCHRONOUS or CORXFER_VAL_START_MODE_HALF_ASYNCHRONOUS, then CORSTATUS_TIMEOUT is returned.	
	For stopping a transfer started in continuous mode, CorXferStop must first be called to flag the end of the transfer, followed by a call to CorXferWait to wait for the actual end of transfer.	
	If for any reason the source of the transfer does not provide enough data for the transfer to finish, CorXferAbort must be used to force an asynchronous end of transfer. In this situation, calling CorXferStop would have no effect.	
See Also	CorXferAbort and CorXferStop	

CorXferStop

Stop transfer synchronously for a transfer resource

Prototype	CORSTATUS **CorXferStop**(CORXFER *hXfer*);	
Description	Stops transfer synchronously for a transfer resource. Transfer will be stopped at the end of the current frame; therefore, last transferred frame is valid.	
Input	*hXfer*	Transfer resource handle
Output	None	
Return Value	CORSTATUS_ARG_INVALID_VALUE, CORSTATUS_INVALID_HANDLE	
Note	Function call returns immediately, therefore current transfers can still be in progress. To detect the actual end of transfer, CorXferWait should be used.	
	If for any reason the source of transfer does not provide enough data for the transfer to finish, CorXferAbort must be used to force an asynchronous end of transfer. In this situation, calling CorXferStop would have no effect.	

See Also	CorXferAbort, CorXferStart and CorXferWait

CorXferUnlinkCounterStamp

Unlink a counter resource from the currently selected item

Prototype	CORSTATUS **CorXferUnlinkCounterStamp**(CORXFER *hXfer*, CORCOUNTER *hCounter*);	
Description	Unlink the specified counter resource from the currently selected item..	
Input	*hXfer*	Transfer resource handle
	hCounter	Counter resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
Notes	To select an item (hSrc, hDst) as the current transfer resource, use CorXferSelect.	
See also	CorXferLinkCounterStamp	

CorXferUnregisterCallback

Unregister callback function for a transfer resource

Prototype	CORSTATUS **CorXferUnregisterCallback**(CORXFER *hXfer*, PCORCALLBACK *callbackFct*);	
Description	Unregisters callback function for the current item of a transfer resource.	
Input	*hXfer*	Transfer resource handle
	callbackFct	Callback function to unregister. See the Data Types section for the PCORCALLBACK definition.
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
Note	To select an item (hSrc, hDst) as the current transfer resource, use CorXferSelect.	
See Also	CorXferRegisterCallback and CorXferSelect	

CorXferWait

Wait until end of transfer or until timeout for a transfer resource

Prototype	CORSTATUS **CorXferWait**(CORXFER *hXfer*, UINT32 *timeout*);	
Description	Waits until end of transfer or until timeout for a transfer resource.	
Input	*hXfer*	Transfer resource handle
	timeout	Maximum time to wait (in ms)
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_XFER_NOT_CONNECTED CORSTATUS_TIMEOUT	
Note	If the transfer has not finished within the specified timeout interval, CORSTATUS_TIMEOUT is returned. The reason could be that the specified timeout interval was too small or that the transfer source didn't provided enough data for the transfer to finish. To asynchronously end the transfer, CorXferAbort function is used.	
See Also	CorXferStart	

View Module

The View Module controls a viewing window within the display device.

Capabilities

ID	Capability
0x00	*Reserved*
0x01	CORVIEW_CAP_LUTS
0x02	CORVIEW_CAP_ZORDER
0x03	CORVIEW_CAP_FLIP
0x04	*Reserved*
0x05	CORVIEW_CAP_COLOR_SPACE_CONVERT
0x06	CORVIEW_CAP_ROTATE
0x07	CORVIEW_CAP_RANGE
0x08	CORVIEW_CAP_MASK
0x09	CORVIEW_CAP_ROI_SRC
0x0a	CORVIEW_CAP_ROI_DST
0x0b	CORVIEW_CAP_ZOOM_HORZ
0x0c	CORVIEW_CAP_ZOOM_HORZ_METHOD
0x0d	CORVIEW_CAP_ZOOM_HORZ_MIN
0x0e	CORVIEW_CAP_ZOOM_HORZ_MAX
0x0f	CORVIEW_CAP_ZOOM_HORZ_MULT
0x10	CORVIEW_CAP_ZOOM_HORZ_MIN_FACTOR
0x11	CORVIEW_CAP_ZOOM_HORZ_MAX_FACTOR
0x12	CORVIEW_CAP_ZOOM_VERT
0x13	CORVIEW_CAP_ZOOM_VERT_METHOD
0x14	CORVIEW_CAP_ZOOM_VERT_MIN
0x15	CORVIEW_CAP_ZOOM_VERT_MAX
0x16	CORVIEW_CAP_ZOOM_VERT_MULT
0x17	CORVIEW_CAP_ZOOM_VERT_MIN_FACTOR
0x18	CORVIEW_CAP_ZOOM_VERT_MAX_FACTOR
0x19	CORVIEW_CAP_NO_TEARING
0x1a	CORVIEW_CAP_LUT_ENABLE
0x1b	CORVIEW_CAP_LUT
0x1c	*Reserved*

0x1d	CORVIEW_CAP_ALPHA_BLEND_MODE
0x1e	CORVIEW_CAP_ALPHA_KEY_MODE
0x1f	CORVIEW_CAP_ALPHA_BLEND_STEP
0x20	CORVIEW_CAP_OVERLAY_MODE
0x21	CORVIEW_CAP_RANGE_MAX

CORVIEW_CAP_ALPHA_BLEND_MODE

Description Specifies the Alpha blending mode types supported by this view module.

Type UINT32

Values CORVIEW_VAL_ALPHA_BLEND_SRC_CONST
The CORVIEW_PRM_ALPHA_BLEND_CONST.srcConst can be used as the weighing factor for source pixels when alpha blending is enabled.

CORVIEW_VAL_ALPHA_BLEND_DST_CONST
The CORVIEW_PRM_ALPHA_BLEND_CONST.dstConst can be used as the weighing factor for destination pixels. If this capability is not present, the weighing factor used is (100% - CORVIEW_PRM_ALPHA_BLEND_CONST.srcConst)

CORVIEW_CAP_ALPHA_BLEND_STEP

Description Specifies the minimum percentage step that must be applied to the alpha blend value in order to get a change in the resulting blended image.

Type UINT32

CORVIEW_CAP_ALPHA_KEY_MODE

Description Alpha keying mode types supported by this view module..

Type UINT32

Values CORVIEW_VAL_ALPHA_KEY_SRC_NOT_EQUAL (0x00000001)
The source pixel is shown at 100% intensity, if the alpha value in the source pixel does not equal CORVIEW_PRM_ALPHA_SRC_VALUE.

CORVIEW_VAL_ALPHA_KEY_DST_NOT_EQUAL (0x00000002)
The destination pixel is shown at 100% intensity, if the alpha value in the destination pixel does not equal CORVIEW_PRM_ALPHA_DST_VALUE.

CORVIEW_CAP_COLOR_SPACE_CONVERT

Description Specifies whether this view can convert video information from one color space to another (the buffer's color space to the display's color space) while it's being displayed.

Type UINT32

Values TRUE, The video information can be converted.
FALSE, The video information cannot be converted.

CORVIEW_CAP_FLIP

Description Specifies whether this view can be flipped during display.

Type	UINT32
Values	CORVIEW_VAL_FLIP_X: Can be flipped vertically. CORVIEW_VAL_FLIP_Y: Can be flipped horizontally.
Note	The values may be ORed when both are supported.
See Also	CORVIEW_PRM_FLIP_X and CORVIEW_PRM_FLIP_Y

CORVIEW_CAP_LUT

Description	Specifies if an output LUT is available.
Type	UINT32
Values	TRUE: At least one LUT is available. FALSE: No LUT is available.
See Also	CORVIEW_PRM_LUT_MAX and CORVIEW_PRM_LUT_ENABLE

CORVIEW_CAP_LUTS

Description	Number of output LUT. A zero value indicates that no LUT is available.
Type	UINT32
Notes	This capability is obsolete and should not be used anymore. Use CORVIEW_CAP_LUT and the parameter CORVIEW_PRM_LUT_MAX.
See Also	CORVIEW_CAP_LUT and CORVIEW_PRM_LUT_MAX

CORVIEW_CAP_LUT_ENABLE

Description	Specifies if an output LUT can be enabled/disabled.
Type	UINT32
Values	TRUE, Output LUT can be enabled/disabled. FALSE, Output LUT cannot be enabled/disabled.

CORVIEW_CAP_MASK

Description	Specifies if certain bits of the pixel format may be masked-out from being shown on the display.
Type	UINT32
Values	TRUE, The pixels can be masked. FALSE, The pixels cannot be masked.
See Also	CORVIEW_PRM_MASK

CORVIEW_CAP_NO_TEARING

Description	Specifies whether the view is capable of refreshing itself in such a way as to avoid tearing on the display.
Type	UINT32
Values	TRUE, The view can avoid tearing. FALSE, The view cannot avoid tearing.

Note Was called CORVIEW_CAP_NOTEARING.
Tearing is visible during continuous acquisition when the field acquisition rate differs from the field display rate, and while active video is being acquired during display refresh periods.

See Also CORVIEW_PRM_STOP_TEARING

CORVIEW_CAP_OVERLAY_MODE

Description Specifies the behavior of a view when the overlay is available.

Type UINT32

Values CORVIEW_VAL_OVERLAY_MODE_ALWAYS_ON_TOP
No color keying scheme is in effect. The associated buffer contents will be displayed directly on the screen using the display adapter's overlay hardware. This is the fastest method, but its drawback is that other windows will not be displayed correctly if they come in front of the Sapera application that has the overlay.

CORVIEW_VAL_OVERLAY_MODE_AUTO_COLOR_KEYING
A destination key color scheme is enabled, i.e., a source buffer pixel will only be displayed if the corresponding pixel on the display surface has the key color. Each time CorViewShow is called, the defined key color is painted on the display surface ROI. Calling CorViewOnPaint will only repaint the key color on the portion of the display surface that has been revealed

CORVIEW_VAL_OVERLAY_MODE_MANUAL_COLOR_KEYING
Similar to auto-keying mode, but the user is responsible for painting the key color on the display surface. This gives the user more flexibility as to where the overlay image should be displayed. An easy way to paint the key color is to use a video memory off-screen buffer that contains the key color. This buffer can be copied very quickly to the display surface by the display adapter's hardware and thus minimizes the CPU load.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_BLENDING
The alpha blending features of the overlay can be enabled.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_KEYING
The alpha keying features of the overlay can be enabled.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_KEYING_BLENDING
This feature implies that the alpha keying and alpha blending features of the overlay hardware are enabled or disabled at the same time. One cannot be enabled without enabling the other.

CORVIEW_CAP_RANGE

Description Specifies whether a range within the pixel bit depth can be designated for viewing.

Type UINT32

Values TRUE, The range can be designated for viewing.
FALSE, The range cannot be designated for viewing.

Note Useful when a buffer must be viewed on a display which is only capable of displaying at a lower bit depth than the buffer pixel depth.

See Also CORVIEW_PRM_RANGE

CORVIEW_CAP_RANGE_MAX

Description	Specifies the maximum value allowed for CORVIEW_PRM_RANGE.
Type	UINT32
Values	The value corresponds to the maximum number of bits of the attached buffer (**CorManGetPixelDepthMax**) minus the view's bit depth.
Note	Useful when a buffer must be viewed on a display that is only capable of displaying at a lower bit depth than the buffer pixel depth.
See Also	CORVIEW_CAP_RANGE and CORVIEW_PRM_RANGE.

CORVIEW_CAP_ROTATE

Description	Specifies whether this view can rotate the video information while it is being displayed.
Type	UINT32
Values	CORVIEW_VAL_ROTATE_ANY, Supports arbitrary integer angle rotation. CORVIEW_VAL_ROTATE_90, Supports rotation angles that are a multiple of 90 degrees.
See Also	CORVIEW_PRM_ROTATE

CORVIEW_CAP_ROI_DST

Description	Specifies whether a destination ROI can be defined on the associated display surface. If so, after the ROI is set up, the video information contained within the source ROI is displayed in the destination ROI. This video information is zoomed when necessary to fill the destination ROI.
Type	UINT32
Values	TRUE, The destination ROI can be defined. FALSE, The destination ROI cannot be defined.
Note	Used to be CORVIEW_CAP_WIN_DST
See Also	CORVIEW_PRM_ROI_DST_HEIGHT, CORVIEW_PRM_ROI_DST_LEFT, CORVIEW_PRM_ROI_DST_TOP and CORVIEW_PRM_ROI_DST_WIDTH

CORVIEW_CAP_ROI_SRC

Description	Specifies whether a source ROI can be defined in the associated buffer. If so, after the ROI is set up, only the region delimited by the ROI is displayed.
Type	UINT32
Values	TRUE, The source ROI can be defined. FALSE, The source ROI cannot be defined.
Note	Used to be CORVIEW_CAP_WIN_SRC
See Also	CORVIEW_PRM_ROI_SRC_HEIGHT, CORVIEW_PRM_ROI_SRC_LEFT, CORVIEW_PRM_ROI_SRC_TOP and CORVIEW_PRM_ROI_SRC_WIDTH

CORVIEW_CAP_ZOOM_HORZ

Description Specifies whether the view supports horizontal zooming.

Type UINT32

Values TRUE, The view supports horizontal zooming.
FALSE, The view does not support horizontal zooming.

Note View zoom applies only to a single view, not to the whole display surface.

See Also CORVIEW_CAP_ZOOM_VERT

CORVIEW_CAP_ZOOM_HORZ_MAX

Description Maximum valid value (in pixels) for the horizontal zoom.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_HORZ_MAX_FACTOR. One and/or the other may be specified.

CORVIEW_CAP_ZOOM_HORZ_MAX_FACTOR

Description Maximum horizontal zoom factor.

Type UINT32

Note A factor of 1000 is equivalent to a 1:1 zoom (no zoom). An alternative to CORVIEW_CAP_ZOOM_HORZ_MAX. One and/or the other may be specified.

CORVIEW_CAP_ZOOM_HORZ_METHOD

Description Horizontal zooming method implemented by the view.

Type INT32

Values CORVIEW_VAL_ZOOM_METHOD_SIMPLE
Simple zoom using pixel dropping or replication.

CORVIEW_VAL_ZOOM_METHOD_INTERPOLATION
Interpolated zoom.

CORVIEW_VAL_ZOOM_METHOD_POW2
Zoom by a power of 2 (e.g., 2x, 4x, 8x, etc.).

CORVIEW_VAL_ZOOM_METHOD_INTEGER
Zoom by an integer factor (e.g., 2x, 3x, 4x, 5x, etc.).

Note The values may be ORed if more than one zoom method applies.
View zoom applies to a single view, not the whole display surface.

CORVIEW_CAP_ZOOM_HORZ_MIN

Description Minimum valid value (in pixels) for the horizontal zoom.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_HORZ_MIN_FACTOR. One and/or the other may be specified.

CORVIEW_CAP_ZOOM_HORZ_MIN_FACTOR

Description Minimum horizontal zoom factor.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_HORZ_MIN. A factor of 1000 is equivalent to a 1:1 zoom (no zoom).

CORVIEW_CAP_ZOOM_HORZ_MULT

Description Granularity (in pixels) for the horizontal zoom parameter.

Type UINT32

Note The horizontal zoom dimension must be a multiple of this value.

CORVIEW_CAP_ZOOM_VERT

Description Specifies whether the view supports vertical zooming.

Type UINT32

Values TRUE, The display supports vertical zooming.
FALSE, The display does not support vertical zooming.

Note View zoom applies only to a single view, not to the whole display surface.

See Also CORVIEW_CAP_ZOOM_HORZ

CORVIEW_CAP_ZOOM_VERT_MAX

Description Maximum valid value (in pixels) for the vertical zoom.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_VERT_MAX_FACTOR.

CORVIEW_CAP_ZOOM_VERT_MAX_FACTOR

Description Maximum vertical zoom factor.

Type UINT32

Note A factor of 1000 is equivalent to a 1:1 zoom (no zoom). An alternative to CORVIEW_CAP_ZOOM_VERT_MAX. One and/or the other may be specified.

CORVIEW_CAP_ZOOM_VERT_METHOD

Description Vertical zooming method implemented by the view.

Type INT32

Values CORVIEW_VAL_ZOOM_METHOD_SIMPLE
Simple zoom using pixel dropping or replication.

CORVIEW_VAL_ZOOM_METHOD_INTERPOLATION
Interpolated zoom.

CORVIEW_VAL_ZOOM_METHOD_POW2
Zoom by a power of 2 (e.g., 2x, 4x, 8x, etc.).

CORVIEW_VAL_ZOOM_METHOD_INTEGER
Zoom by an integer factor (e.g., 2x, 3x, 4x, 5x, etc.)

Note The values may be ORed if more than one zoom method applies.
View zoom applies to a single view, not the whole display surface.

CORVIEW_CAP_ZOOM_VERT_MIN

Description Minimum valid value (in pixels) for the vertical zoom.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_HORZ_MIN_FACTOR.
One and/or the other may be specified.

CORVIEW_CAP_ZOOM_VERT_MIN_FACTOR

Description Minimum vertical zoom factor.

Type UINT32

Note An alternative to CORVIEW_CAP_ZOOM_VERT_MIN. A factor of 1000 is equivalent to a 1:1 zoom (no zoom). One and/or the other may be specified.

CORVIEW_CAP_ZOOM_VERT_MULT

Description Granularity (in pixels) for the vertical zoom parameter.

Type UINT32

Note The vertical zoom dimension must be a multiple of this value.

CORVIEW_CAP_ZORDER

Description Specifies whether this view can be Z-ordered (have its relative position in the Z-plane specified) on the display associated with it.

Type UINT32

Values TRUE, The view can be Z-ordered.
FALSE, The view cannot be Z-ordered.

See Also CORVIEW_PRM_ZORDER

Parameters

ID	Parameter	Attribute
0x00	*Reserved*	
0x01	CORVIEW_PRM_KEYER_COLOR_RED	Read/Write
0x02	CORVIEW_PRM_KEYER_COLOR_GREEN	Read/Write
0x03	CORVIEW_PRM_KEYER_COLOR_BLUE	Read/Write
0x04	CORVIEW_PRM_KEYER_COLOR_PALETTE	Read/Write
0x05	CORVIEW_PRM_KEYER_CHROMA_LO	Read/Write
0x06	CORVIEW_PRM_KEYER_CHROMA_HI	Read/Write
0x07	*Reserved*	
0x08	*Reserved*	
0x09	*Reserved*	
0x0a	*Reserved*	
0x0b	*Reserved*	
0x0c	CORVIEW_PRM_HWND	Read/Write
0x0d	CORVIEW_PRM_LUT_NUMBER	Read/Write
0x0e	CORVIEW_PRM_FLIP_X	Read/Write
0x0f	CORVIEW_PRM_FLIP_Y	Read/Write
0x10	CORVIEW_PRM_ROTATE	Read/Write
0x11	CORVIEW_PRM_ZORDER	Read/Write
0x12	CORVIEW_PRM_RANGE	Read/Write
0x13	CORVIEW_PRM_MASK	Read/Write
0x14	CORVIEW_PRM_ROI_SRC_LEFT	Read/Write
0x15	CORVIEW_PRM_ROI_SRC_TOP	Read/Write
0x16	*Reserved*	
0x17	CORVIEW_PRM_ROI_SRC_HEIGHT	Read/Write
0x18	CORVIEW_PRM_ROI_SRC_WIDTH	Read/Write
0x19	CORVIEW_PRM_ROI_DST_LEFT	Read/Write
0x1a	CORVIEW_PRM_ROI_DST_TOP	Read/Write
0x1b	CORVIEW_PRM_ROI_DST_HEIGHT	Read/Write
0x1c	CORVIEW_PRM_ROI_DST_WIDTH	Read/Write
0x1d	CORVIEW_PRM_STOP_TEARING	Read/Write
0x1e	CORVIEW_PRM_LUT_MAX	Read Only
0x1f	CORVIEW_PRM_LUT_ENABLE	Read/Write
0x20	CORVIEW_PRM_LUT_FORMAT	Read Only
0x21	CORVIEW_PRM_MODE	Read Only

0x22	CORVIEW_PRM_OVERLAY_MODE	Read/Write
0x23	CORVIEW_PRM_HWND_TITLE	Read/Write
0x24	CORVIEW_PRM_ALPHA_BLEND_CONST	Read/Write
0x25	CORVIEW_PRM_ALPHA_BLEND_MODE	Read/Write
0x26	CORVIEW_PRM_ALPHA_KEY_MODE	Read/Write
0x27	CORVIEW_PRM_ALPHA_KEY_VALUE	Read/Write
0x28	CORVIEW_PRM_ZOOM_HORZ_METHOD	Read/Write
0x29	CORVIEW_PRM_ZOOM_VERT_METHOD	Read/Write

CORVIEW_PRM_ALPHA_BLEND_CONST

Description Sets the alpha blend constants used for alpha blending between source and destination.

Type CORVIEW_ALPHA_BLEND_CONSTS

Note This parameter is a structure. The CorViewSetPrmEx function must be used to set the value.

CORVIEW_PRM_ALPHA_BLEND_MODE

Description Sets the alpha blending mode. Only effective when alpha blending is enabled with the CORVIEW_PRM_OVERLAY_MODE parameter.

Type UINT32

Values CORVIEW_VAL_ALPHA_BLEND_SRC_CONST (0x00000001)
The CORVIEW_PRM_ALPHA_BLEND_SRC_CONST will be used as the weighing factor for the source pixels..

CORVIEW_VAL_ALPHA_BLEND_DST_CONST (0x00000002)
The CORVIEW_PRM_ALPHA_BLEND_DST_CONST is used as the weighing factor for the destination pixels. If this feature is not enabled, the weighing factor used for the destination pixel is (100% - CORVIEW_PRM_ALPHA_BLEND_SRC_CONST)

CORVIEW_PRM_ALPHA_KEY_MODE

Description Used to read or set the alpha keying mode. This value only has an effect when alpha keying is enabled.with the CORVIEW_PRM_OVERLAY_MODE parameter.

Type UINT32

Values CORVIEW_VAL_ALPHA_KEY_SRC_NOT_EQUAL (0x00000001)
The source pixel is displayed at full intensity if the alpha value in the source pixel does not equal CORVIEW_PRM_ALPHA_KEY_VALUE.

CORVIEW_VAL_ALPHA_KEY_DST_NOT_EQUAL (0x00000002)
The destination pixel is displayed at full intensity, if the alpha value in the destination pixel does not equal CORVIEW_PRM_ALPHA_KEY_VALUE.

CORVIEW_PRM_ALPHA_KEY_VALUE

Description Sets the value that is compared with the alpha keying bits of the source or destination surface when alpha keying is enabled.

Type UINT32

Limits	Value limits - maximum alpha value supported by the destination or source pixel format. RGB5551 surface: value can be 0 or 1 RGB8888 surface: value can be 0 to 255.
Note	CORVIEW_PRM_ALPHA_KEY_MODE indicates whether this value is compared with the alpha bits of the source buffer or the alpha bits of the destination buffer

CORVIEW_PRM_FLIP_X

Description	Enable/disable X axis vertical flipping of the source image while the view is displayed.
Type	UINT32
Values	TRUE, Enables vertical flipping. FALSE, Disables vertical flipping.
Note	Valid only when CORVIEW_CAP_FLIP includes the CORVIEW_CAP_FLIP_X flag.
See Also	CORVIEW_CAP_FLIP and CORVIEW_PRM_FLIP_Y

CORVIEW_PRM_FLIP_Y

Description	Enable/disable Y axis horizontal flipping of the source image while the view is displayed.
Type	UINT32
Values	TRUE, Enables horizontal flipping. FALSE, Disables horizontal flipping.
Note	Valid only when CORVIEW_CAP_FLIP includes the CORVIEW_CAP_FLIP_Y flag.
See Also	CORVIEW_CAP_FLIP and CORVIEW_PRM_FLIP_X

CORVIEW_PRM_HWND

Description	Window handle to be used as the destination view display surface. When specified, the destination rectangle becomes relative to the window's client area rectangle, instead of the whole display surface.
Type	HWND
Values	Valid window handles including NULL. If set to –1, Sapera will automatically create a window running in a separate thread. In this case, CORVIEW_PRM_HWND can be read later to obtain the HWND of the created window.
Notes	Valid only on a system display.
See also	CORVIEW_PRM_HWND_TITLE

CORVIEW_PRM_HWND_TITLE

Description	Specifies the title of the window created by Sapera if CORVIEW_PRM_HWND is set to –1.
Type	CHAR[128]
Values	Zero-terminated array of characters with a fixed size of 128 bytes.
See Also	CorViewSetPrmEx

CORVIEW_PRM_KEYER_CHROMA_HI

Description The upper keying color range used with CORVIEW_VAL_KEYER_TYPE_CHROMA color keyer.

Type UINT32

Values *Unsigned* integer with format dependent on the color mode of the target display.

Note Colors for all color modes are defined using an UINT32 value, although some of them will not be represented in the full 32 bits.

See Also CORVIEW_PRM_KEYER_CHROMA_LO

CORVIEW_PRM_KEYER_CHROMA_LO

Description The lower keying color range used with a CORVIEW_VAL_KEYER_TYPE_CHROMA color keyer.

Type UINT32

Values *Unsigned* integer with format dependent on the color mode of the target display.

Note Colors for all color modes are defined using an UINT32 value, although some of them will not be represented in the full 32 bits.

See Also CORVIEW_PRM_KEYER_CHROMA_HI

CORVIEW_PRM_KEYER_COLOR_BLUE

Description The blue component (0 to 255) of the keying color used with a CORVIEW_VAL_KEYER_TYPE_COLOR color keyer.

Type UINT32

Note Used in all color display modes except 256-color.

CORVIEW_PRM_KEYER_COLOR_GREEN

Description The green component (0 to 255) of the keying color used with a CORVIEW_VAL_KEYER_TYPE_COLOR color keyer.

Type UINT32

Note Used in all color display modes except 256-color.

CORVIEW_PRM_KEYER_COLOR_PALETTE

Description The palette keying index used with a CORVIEW_VAL_KEYER_TYPE_COLOR color keyer.

Type UINT32

Values *Unsigned* integer values [0...255]

Note Used only in the 256-color display mode.

CORVIEW_PRM_KEYER_COLOR_RED

Description The red component (0 to 255) of the keying color used with a CORVIEW_VAL_KEYER_TYPE_COLOR color keyer.

Type UINT32

Note Used in all color display modes except 256-color.

CORVIEW_PRM_LUT_ENABLE

Description Enable/disable the output LUT after checking CORVIEW_CAP_LUT_ENABLE.

Type UINT32

Values TRUE, Enable the output LUT.
FALSE, Disable the output LUT.

CORVIEW_PRM_LUT_FORMAT

Description LUT format

Type UINT32

Note Read only parameter.

CORVIEW_PRM_LUT_MAX

Description Maximum number of LUTs available, based on the current pixel depth and format.

Type UINT32

Note Read only parameter.

CORVIEW_PRM_LUT_NUMBER

Description Selects an active output LUT from among those available.

Type UINT32

Values [0...CORVIEW_CAP_LUTS-1]

Note Valid only when CORVIEW_CAP_LUTS is not zero.

CORVIEW_PRM_MASK

Description Specifies a mask to apply to the view while it is being displayed. The mask must be in the pixel format of the buffer associated with the view. Bits set to 1, designate the image bits to be viewed.

Type UINT32

Values [0...0xffffffff]

Note Valid only when CORVIEW_CAP_MASK is not zero.

CORVIEW_PRM_MODE

Description Specifies the type of transfer between the buffer and the viewer. This parameter reflects the mode specified by the argument in the CorViewNew function. If the mode specified is CORVIEW_VAL_MODE_AUTO_DETECT, then the mode selected by the driver is returned.

Type UINT32

Values See CorViewNew

See Also CorBufferNew, CORVIEW_PRM_OVERLAY_MODE, CORVIEW_PRM_KEYER_COLOR_PALETTE, CORVIEW_PRM_KEYER_COLOR_RED, CORVIEW_PRM_KEYER_COLOR_GREEN, CORVIEW_PRM_KEYER_COLOR_BLUE, CORVIEW_PRM_KEYER_CHROMA_HI and CORVIEW_PRM_KEYER_CHROMA_LO

Note Read only parameter.

CORVIEW_PRM_OVERLAY_MODE

Description Specifies the behavior of a view when CORVIEW_PRM_MODE is set to CORVIEW_VAL_MODE_OVERLAY.

Type UINT32

Values

CORVIEW_VAL_OVERLAY_MODE_ALWAYS_ON_TOP
No color keying scheme is in effect. The associated buffer's contents will be displayed directly on the screen using the display adapter's overlay hardware. This is the fastest method, but its drawback is that other windows will not be displayed correctly if they come in front of the Sapera application that has the overlay.

CORVIEW_VAL_OVERLAY_MODE_AUTO_COLOR_KEYING
A destination key color scheme is enabled, i.e., a source buffer pixel will only be displayed if the corresponding pixel on the display surface has the key color. Each time CorViewShow is called, the defined key color is painted on the display surface ROI. Calling CorViewOnPaint will only repaint the key color on the portion of the display surface that has been revealed

CORVIEW_VAL_OVERLAY_MODE_MANUAL_COLOR_KEYING
Similar to auto-keying mode, but the user is responsible for painting the key color on the display surface. This gives the user more flexibility as to where the overlay image should be displayed. An easy way to paint the key color is to use a video memory off-screen buffer that contains the key color. This buffer can be copied very quickly to the display surface by the display adapter's hardware and thus minimizes the CPU load.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_BLENDING
Enables the alpha blending features of the overlay. The parameter CORVIEW_PRM_ALPHA_BLEND_MODE is used to specify the type of alpha blending.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_KEYING
Enables the alpha keying features of the overlay. The parameter CORVIEW_PRM_ALPHA_KEY_MODE is used to specify the type of alpha keying.

CORVIEW_VAL_OVERLAY_MODE_ALPHA_KEYING_BLENDING
Enables both the alpha keying and alpha blending features of the overlay hardware.

See Also CorBufferNew, CorViewNew, CORVIEW_PRM_KEYER_COLOR_PALETTE, CORVIEW_PRM_KEYER_COLOR_RED, CORVIEW_PRM_KEYER_COLOR_GREEN, CORVIEW_PRM_KEYER_COLOR_BLUE, CORVIEW_PRM_KEYER_CHROMA_HI and CORVIEW_PRM_KEYER_CHROMA_LO

CORVIEW_PRM_RANGE

Description Specifies how many of the most significant bits are not to be displayed. If the display only uses a subset of the image data (e.g., a 32-bit image on an 8-bit display), this parameter's value determines the offset of the subset from the image data high bit.

Type UINT32

Values [0...CORVIEW_CAP_RANGE_MAX]

Note Valid only when CORVIEW_CAP_RANGE is not zero. Useful when a buffer must be viewed on a display which can only display at a lower bit depth than the buffer pixel depth (e.g., 16-bit monochrome buffers). Refer to CORVIEW_CAP_RANGE_MAX for the maximum value allowed for this parameter.

CORVIEW_PRM_ROTATE

Description Specifies the angle that the source image is rotated when the view is being displayed.

Type UINT32

Values [0...359] If the CORVIEW_CAP_ROTATE capability includes the CORVIEW_VAL_ROTATE_90 flag, only values of 0, 90, 180, and 270 are valid.

Note Valid only when CORVIEW_CAP_ROTATE is not zero.

CORVIEW_PRM_ROI_DST_HEIGHT

Description Height of the destination ROI of the display surface which is associated with the view.

Type UINT32

Values [0...display height-CORVIEW_PRM_ROI_DST_TOP] or [0...window client area height-CORVIEW_PRM_ROI_DST_TOP]

Note Valid only when CORVIEW_CAP_ROI_DST is not zero. By default, the initial source ROI dimensions are the same as those of the source buffer. When the destination ROI dimensions differ from those of the source ROI, the source buffer will be zoomed.

CORVIEW_PRM_ROI_DST_LEFT

Description Left edge coordinate of the destination ROI relative to the coordinates of the display surface associated with the view. If a display window is specified, the coordinate is relative to the display window’s client area.

Type UINT32

Values [0...display width-1] or [0...window client area width-1]

Note Valid only when CORVIEW_CAP_ROI_DST is not zero. By default, the initial destination ROI dimensions are the same as those of the source buffer.

CORVIEW_PRM_ROI_DST_TOP

Description Top edge coordinate of the destination ROI relative to the coordinates of the display surface associated with the view. If a display window is specified, the coordinate is relative to the display window's client area.

Type UINT32

Values [0...display height-1] or [0...window client area height-1]

Note Valid only when CORVIEW_CAP_ROI_DST is not zero. By default, the initial destination ROI dimensions are the same as those of the source buffer.

CORVIEW_PRM_ROI_DST_WIDTH

Description Width of the destination ROI of the display surface associated with the view.

Type UINT32

Values [0...display width- CORVIEW_PRM_ROI_DST_LEFT] or [0...window client area width-CORVIEW_PRM_ROI_DST_LEFT]

Note Valid only when CORVIEW_CAP_ROI_DST is not zero. By default, the initial source ROI dimensions are the same as those of the source buffer. When the destination ROI dimensions differ from those of the source ROI, the source buffer will be zoomed.

CORVIEW_PRM_ROI_SRC_HEIGHT

Description Height of the source ROI of the buffer associated with the view.

Type UINT32

Values [0...buffer height-CORVIEW_PRM_ROI_SRC_TOP]

Note Valid only when CORVIEW_CAP_ROI_SRC is not zero. By default, the initial source ROI dimensions are the same as those of the source buffer. When the source ROI dimensions differ from those of the destination ROI, the source buffer will be zoomed.

CORVIEW_PRM_ROI_SRC_LEFT

Description Left edge coordinate of the source ROI relative to the coordinates of the buffer associated with the view.

Type UINT32

Values [0...buffer width-1]

Note Valid only when CORVIEW_CAP_ROI_SRC is not zero. By default, the initial source ROI dimensions are the same as those of the source buffer.

CORVIEW_PRM_ROI_SRC_TOP

Description Top edge coordinate of the source ROI relative to the coordinates of the buffer associated with the view.

Type UINT32

Values [0...buffer height-1]

Note Valid only when CORVIEW_CAP_ROI_SRC is not zero. By default, the initial source ROI

dimensions are the same as those of the source buffer.

CORVIEW_PRM_ROI_SRC_WIDTH

Description	Width of the source ROI of the buffer associated with the view.
Type	UINT32
Values	[0...buffer width-CORVIEW_PRM_ROI_SRC_LEFT]
Note	Valid only when CORVIEW_CAP_ROI_SRC is not zero. By default, the initial source ROI dimensions are the same as those of the source buffer. When the source ROI dimensions differ from those of the destination ROI, the source buffer will be zoomed.

CORVIEW_PRM_STOP_TEARING

Description	Enables/disables the mechanism permitting the view to be displayed without tearing.
Type	UINT32
Values	TRUE, Enables the mechanism. FALSE, Disables the mechanism.
Note	Was CORVIEW_PRM_STOPTEARING, now obsolete. Valid only when the CORVIEW_CAP_NO_TEARING capability is not zero.

CORVIEW_PRM_ZOOM_HORZ_METHOD

Description	Sets the zooming horizontal method.
Type	UINT32
Values	CORVIEW_VAL_ZOOM_METHOD_SIMPLE Simple zoom using pixel dropping or replication. CORVIEW_VAL_ZOOM_METHOD_INTERPOLATION Interpolated zoom.
See also	CORVIEW_CAP_ZOOM_HORZ_METHOD

CORVIEW_PRM_ZOOM_VERT_METHOD

Description	Sets the zooming vertical method.
Type	UINT32
Values	CORVIEW_VAL_ZOOM_METHOD_SIMPLE Simple zoom using line dropping or replication. CORVIEW_VAL_ZOOM_METHOD_INTERPOLATION Interpolated zoom.
See also	CORVIEW_CAP_ZOOM_VERT_METHOD

CORVIEW_PRM_ZORDER

Description	The relative ordering of this view in relation to other views on the same display surface. Used when there is more than one keyed view on a dedicated display, such as an overlay or pass-through.
Type	UINT32
Values	[0...topmost view on this display]
Note	Valid only when CORVIEW_CAP_ZORDER is not zero.

Functions

Function	Description
CorViewBlit	*Copies data between the off-screen/overlay buffers associated with the views*
CorViewFree	*Releases handle to a view resource*
CorViewGetCap	*Gets view capability value from a view resource*
CorViewGetLut	*Gets output LUT values from a view resource*
CorViewGetPrm	*Gets view parameter value from a view resource*
CorViewHide	*Stops displaying a view resource*
CorViewNew	*Creates in the specified server's memory a new view resource*
CorViewOnMove	*WM_MOVE handler callback function of a view resource*
CorViewOnPaint	*WM_PAINT handler callback function of a view resource*
CorViewOnSize	*WM_SIZE handler callback function of a view resource*
CorViewSetBuffer	*Set a new buffer resource for a view resource*
CorViewSetLut	*Sets output LUT values to a view resource*
CorViewSetPrm	*Sets a simple view parameter of a view resource*
CorViewSetPrmEx	*Sets a complex view parameter of a view resource*
CorViewShow	*Displays a view resource*
CorViewShowWithOps	*Displays a view resource using alternate parameters*
CorViewUpdatePos	*Updates position of a view resource*

CorViewBlit

Copies data between the off-screen/overlay buffers associated with the views

Prototype	CORSTATUS **CorViewBlit**(CORVIEW *hSource*, CORVIEW *hDest*, UINT32 *ops*, CORVIEW_BLIT_DESC **prms*);	
Description	Copies ("blits") data between the off-screen/overlay buffers associated with the views.	
Input	*hSource*	Source view resource handle
	hDest	Destination view resource handle
	ops	Operations to be applied during the blit. Can be ORed to combine operations.

CORVIEW_OPS_SRC_ROI
Override the source ROI defined for the source view.
CORVIEW_OPS_DST_ROI
Override the source ROI defined for the destination view.
CORVIEW_OPS_ROTATION
Rotate the source image by a given angle into the destination image.
CORVIEW_OPS_MIRROR_UP_DOWN
Flip the source image vertically into the destination image.
CORVIEW_OPS_MIRROR_LEFT_RIGHT
Flip the source image horizontally into the destination image
CORVIEW_OPS_COLOR_FILL
Disregard source buffer contents; fill the destination with a given color.
CORVIEW_OPS_SRC_KEY_COLOR
Apply a source key color scheme during the blit. Only the source pixels not corresponding to the key color will get copied.
CORVIEW_OPS_DST_KEY_COLOR
Apply a destination key color scheme during the blit. Only the destination pixels corresponding to the key color will get copied.

prms Pointer to a structure containing parameters that affect the blit operations. See CORVIEW_BLIT_DESC Structure Definition.

Output None

Return Value CORSTATUS_INCOMPATIBLE_VIEW, CORSTATUS_DDRAW_ERROR, CORSTATUS_ARG_INVALID_VALUE, CORSTATUS_ARG_NULL (if *prms* is NULL) and CORSTATUS_NOT_IMPLEMENTED

Note The source and destination views must be associated with either off-screen or overlay buffers. The pixel formats of the buffers should be the same. Not all operations are necessarily supported, alone or combined. Since CorViewBlit uses hardware acceleration (through DirectDraw), the supported operations may depend on the display adapter's hardware, driver version, and buffer pixel formats. The operations supported are not necessarily the same for overlay buffers as for off-screen buffers.

See Also CorViewNew

CorViewFree

Release handle to a view resource

Prototype CORSTATUS **CorViewFree**(CORVIEW *hView*);

Description Releases handle to a view resource.

Input *hView* View resource handle

Output None

Return Value CORSTATUS_DDRAW_ERROR, CORSTATUS_INVALID_HANDLE and CORSTATUS_RESOURCE_IN_USE

Note The *hView* handle is invalid after a call to this function. Only after the view has been destroyed when using this function can the buffer be destroyed. An attempt to do otherwise will cause an error in CorBufferFree. Likewise, the display may not be released before the view using it is destroyed.

See Also CorViewNew and CorBufferFree

CorViewGetCap

Get view capability value from a view resource

Prototype	CORSTATUS **CorViewGetCap**(CORVIEW *hView*, UINT32 *cap*, UINT32 **value*);	
Description	Gets view capability value from a view resource	
Input	*hView*	View resource handle
	cap	View resource capability requested
Output	*value*	Value of the capability
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_CAP_INVALID, CORSTATUS_CAP_NOT_AVAILABLE and CORSTATUS_INVALID_HANDLE	
Note	If the capability number passed as the argument does not exist, CORSTATUS_CAP_INVALID is returned. If the capability is not valid on a particular device, CORSTATUS_CAP_NOT_AVAILABLE is returned.	
See Also	CorViewGetPrm and CorViewSetPrm	

CorViewGetLut

Gets output LUT values from a view resource

Prototype	CORSTATUS **CorViewGetLut**(CORVIEW *hView*, CORLUT *hLut,* UINT32 *lutNumber*);	
Description	Copies the values of the LUT specified by *lutNumber* into hLut.	
Input	*hView*	View resource handle
	hLut	LUT resource handle
	lutNumber	LUT number in view resource
Output	None	
Return Value	CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_CAP_NOT_AVAILABLE, CORSTATUS_INCOMPATIBLE_LUT and CORSTATUS_INVALID_HANDLE	
Note	This function will succeed only if the CORVIEW_CAP_LUT capability is TRUE; otherwise, CORSTATUS_CAP_NOT_AVAILABLE is returned. If multiple LUTs are available for the view, the values of the one specified by lutNumber will be copied. The LUT number value range is [0...CORVIEW_PRM_MAX_LUT1].	
See Also	CorViewSetLut and CORVIEW_PRM_LUT_NUMBER	

CorViewGetPrm

Get view parameter value from a view resource

Prototype	CORSTATUS **CorViewGetPrm**(CORVIEW *hView*, UINT32 *prm*, void **value*);	
Description	Gets view parameter value from a view resource.	
Input	*hView*	View resource handle
	prm	View parameter requested
Output	*value*	Current value of the parameter
Return Value	CORSTATUS_ARG_NULL (if *value* is NULL), CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID and CORSTATUS_PRM_NOT_AVAILABLE	

Note	If the parameter number passed as the argument does not exist, CORSTATUS_PRM_INVALID is returned. If the parameter is not valid on a particular device, CORSTATUS_PRM_NOT_AVAILABLE is returned.
See Also	CorViewGetCap and CorViewSetPrm

CorViewHide

Stops displaying a view resource

Prototype	CORSTATUS **CorViewHide**(CORVIEW *hView*);
Description	Stops displaying a view resource.
Input	*hView* View resource handle
Output	None
Return Value	CORSTATUS_DDRAW_ERROR, CORSTATUS_INVALID_HANDLE
Note	CORSTATUS_DDRAW_ERROR is returned when an error occurs while accessing Direct Draw implementation.
See Also	CorViewShow

CorViewNew

Create in the specified server's memory a new view resource

Prototype	CORSTATUS **CorViewNew**(CORSERVER *hServer*, CORDISPLAY *hDisplay,* CORBUFFER *hBuffer,* UINT32 *mode,* CORVIEW **hView*);	
Description	Create a new view resource in a specific mode from the display and buffer parameters.	
Input	*hServer*	Server handle
	hDisplay	Display resource handle
	hBuffer	Buffer resource handle
	mode	Viewing mode. With Sapera 3.50 or later, use the auto-detect mode. CORVIEW_VAL_MODE_AUTO_DETECT, The appropriate mode will be selected among the three following modes depending on the buffer. CORVIEW_VAL_MODE_DIB, This mode uses a device-independent bitmap to represent and transfer buffer data to the display. This option can only be selected if the buffer is of type contiguous, scatter-gather, or virtual. If the buffer is of one of those types, the auto-detect option will automatically choose this option. CORVIEW_VAL_MODE_BLT, This mode uses DirectDraw to perform an efficient data transfer (BLT) from the off-screen memory to the display memory or from video memory to the display memory. This mode is faster than the DIB mode. This option can only be selected if the buffer is of type off-screen. If the buffer is of this type, the auto-detect option will automatically choose this option.

CORVIEW_VAL_MODE_OVERLAY, This mode uses the display adapter's overlay hardware to display the overlay buffer. If a keying operation is performed, the overlay buffer will be displayed only if the color of the corresponding pixel, seen on the display, has a specific value or a value within a given range. The default value of the key color can be modified. Furthermore, this is the fastest mode since it does not require any data transfer. Note that this option can only be selected if the buffer is of type *overlay*, and if it is, the auto-detect option will automatically choose this mode.

Output *hView* View resource handle

Return Value CORSTATUS_ARG_NULL (if *hView* is NULL), CORSTATUS_DDRAW_ERROR, CORSTATUS_DDRAW_NOT_AVAILABLE, CORSTATUS_INCOMPATIBLE_BUFFER, CORSTATUS_INCOMPATIBLE_LOCATION, CORSTATUS_INVALID_HANDLE and CORSTATUS_NO_MEMORY

Note If the buffer cannot be associated with the display or the desired viewing mode is not available, the view is not created and CORSTATUS_DDRAW_ERROR is returned.
If the *hBuffer* or *hDisplay* parameters are not on the same server as the *server* parameter, the view is not created and CORSTATUS_INCOMPATIBLE_LOCATION is returned.
If the buffer type is not compatible with the mode, the view is not created and CORSTATUS_INCOMPATIBLE_BUFFER is returned.
The buffer cannot be destroyed before the view using it is destroyed. An attempt to do so will cause an error in CorBufferFree. This also applies to the display handle.

See Also CORBUFFER_PRM_TYPE and CORVIEW_PRM_OVERLAY_MODE

CorViewOnMove

WM_MOVE handler callback function of a view resource

Prototype CORSTATUS **CorViewOnMove**(CORVIEW *hView*);

Description A callback function that should be called in the WM_MOVE message handler of the target window if the display is a system display and CORVIEW_PRM_HWND is not 0.

Input *hView* View resource handle

Output None

Return Value CORSTATUS_DDRAW_ERROR and CORSTATUS_INVALID_HANDLE

Note If CORVIEW_PRM_HWND is 0, CORSTATUS_PRM_INVALID_VALUE is returned.
CORSTATUS_DDRAW_ERROR is returned for errors while accessing Direct Draw.

See Also CorViewHide, CorViewShow and CorViewUpdatePos

CorViewOnPaint

WM_PAINT handler callback function of a view resource

Prototype CORSTATUS **CorViewOnPaint**(CORVIEW *hView*);

Description A callback function that should be called in the WM_PAINT message handler of the target window if the display is a system display and CORVIEW_PRM_HWND is not 0.

Input *hView* View resource handle

Output None

Return Value	CORSTATUS_DDRAW_ERROR and CORSTATUS_INVALID_HANDLE
Note	The only difference with CorViewShow is in the case of an auto-keying overlay. In that case, CorViewOnPaint will repaint the key color only in the region of the view's ROI which has been invalidated since the last call of CorViewOnPaint or CorViewShow; whereas CorViewShow will repaint the key color on the whole ROI. If CORVIEW_PRM_HWND is 0, CORSTATUS_PRM_INVALID_VALUE is returned. CORSTATUS_DDRAW_ERROR is returned when an error occurs while accessing the Direct Draw implementation.
See Also	CorViewHide, CorViewShow and CorViewUpdatePos

CorViewOnSize

WM_SIZE handler callback function of a view resource

Prototype	CORSTATUS **CorViewOnSize**(CORVIEW *hView*);
Description	A callback function that should be called in the WM_SIZE message handler of the target window if the display is a system display and CORVIEW_PRM_HWND is not 0.
Input	*hView* View resource handle
Output	None
Return Value	CORSTATUS_DDRAW_ERROR and CORSTATUS_INVALID_HANDLE
Note	If CORVIEW_PRM_HWND is 0, CORSTATUS_PRM_INVALID_VALUE is returned. CORSTATUS_DDRAW_ERROR is returned for errors while accessing Direct Draw.
See Also	CorViewHide, CorViewShow and CorViewUpdatePos

CorViewSetBuffer

Set a new buffer resource for a view resource

Prototype	CORSTATUS **CorViewSetBuffer**(CORVIEW hView, CORBUFFER *hBuffer*);
Description	Sets a new buffer resource for a view resource
Input	hView View resource handle
	hBuffer Buffer resource handle
Notes	New buffer resource must have the same size and format as the buffer resource that has been specified when creating the view resource.
Return Value	CORSTATUS_INVALID_HANDLE and CORSTATUS_ARG_INVALID
See Also	CorViewNew

CorViewSetLut

Sets output LUT values to a view resource

Prototype	CORSTATUS **CorViewSetLut**(CORVIEW *hView*, CORLUT *hLut,* UINT32 *lutNumber*);
Description	Copies the values of hLut into the LUT specified by lutNumber.
Input	*hView* View resource handle
	hLut LUT resource handle created with CorLutNew or CorLutNewFromFile.
	lutNumber LUT number in View resource.

If multiple LUTs are available for the View, the values of the one specified by lutNumber will be copied. The *lutNumber* value range is [0...CORVIEW_CAP_LUTS-1].

Output	None
Return Value	CORSTATUS_ARG_OUT_OF_RANGE, CORSTATUS_CAP_NOT_AVAILABLE, CORSTATUS_INCOMPATIBLE_LUT, CORSTATUS_INVALID_HANDLE and CORSTATUS_NOT_IMPLEMENTED
Note	This function will succeed only if the CORVIEW_CAP_LUT capability is TRUE; otherwise, CORSTATUS_CAP_NOT_AVAILABLE is returned.
See Also	CorViewGetLut and CORVIEW_PRM_LUT_NUMBER

CorViewSetPrm

Sets a simple view parameter of a view resource

Prototype	CORSTATUS **CorViewSetPrm**(CORVIEW *hView*, UINT32 *prm*, UINT32 *value*);	
Description	Sets a simple view parameter to a new value	
Input	*hView*	View resource handle
	prm	View parameter to modify
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID and CORSTATUS_PRM_NOT_AVAILABLE	
See Also	CorViewGetPrm and CorViewGetCap	

CorViewSetPrmEx

Sets a complex view parameter of a view resource

Prototype	CORSTATUS **CorViewSetPrmEx**(CORVIEW *hView*, UINT32 *param*, const void * *value*);	
Description	Sets a complex view parameter to a new value	
Input	*hView*	View resource handle
	param	View parameter to modify
	value	New value of the parameter
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_PRM_INVALID, CORSTATUS_PRM_NOT_AVAILABLE and CORSTATUS_PRM_READ_ONLY	
Note	A complex parameter is greater than a UINT32 (like CORVIEW_PRM_HWND_TITLE). If the parameter size is UINT32, use either CorViewSetPrm or CorViewSetPrmEx.	
See Also	CorViewGetPrm and CorViewGetCap	

CorViewShow

Displays a view resource

Prototype	CORSTATUS **CorViewShow**(CORVIEW *hView*);	
Description	Displays a view resource.	
Input	*hView*	View resource handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_DDRAW_ERROR CORSTATUS_NOT_ACCESSIBLE, CORSTATUS_INCOMPATIBLE_BUFFER	
Note	Displays the view using the mode specified at the creation of the view. If the buffer associated with the view is an overlay and the value of CORVIEW_PRM_MODE is CORVIEW_VAL_OVERLAY MODE_AUTO_KEYING, CorViewShow will paint the chosen keying color in the client area of the target window, provided CORVIEW_PRM_HWND is not 0 and the display is a system display.	
	If the buffer associated with the view is an overlay buffer, calling CorViewShow will return an error if the pixel format is not listed in CORDISPLAY_PRM_PIXEL_TYPE_OVERLAY. If the associated buffer is an off-screen buffer, calling CorViewShow may take more time to execute if that buffer's pixel format is not listed in CORDISPLAY_PRM_PIXEL_TYPE_OFFSCREEN. In that case, the conversion of pixel format and copying to display memory is done in software.	
See Also	CorViewHide, CorViewShowWithOps, CORVIEW_PRM_MODE and CORVIEW_PRM_OVERLAY_MODE	

CorViewShowWithOps

Displays a view resource using alternate parameters

Prototype	CORSTATUS **CorViewShowWithOps**(CORVIEW *hView,* UINT32 *ops,* CORVIEW_BLIT_DESC **prms*);	
Description	Displays a view resource using alternate parameters, using the mode specified by the buffer associated with the view.	
Input	*hView*	View resource handle
	ops	Blit operations to apply which override the view resource's parameters. Valid values are the same as for the *ops* parameter of CorViewBlit.
	prms	Pointer to a structure containing parameters that affect the blit operations. See CORVIEW_BLIT_DESC Structure Definition
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE, CORSTATUS_DDRAW_ERROR and CORSTATUS_NOT_ACCESSIBLE	
Note	Displays the view using the mode specified by the buffer associated with the view. Calling CorViewShowWithOps for a view whose associated buffer's type is not CORBUFFER_VAL_TYPE_OFFSCREEN or CORBUFFER_VAL_TYPE_OVERLAY, will return an error.	
	If the buffer associated with the view is an overlay and the value of	

CORVIEW_PRM_OVERLAY_MODE is CORVIEW_VAL_OVERLAY_MODE_AUTO_KEYING, CorViewShowWithOps will paint the chosen keying color in the client area of the target window, provided CORVIEW_PRM_HWND is not 0 and the display is a system display.

See Also CorViewBlit and CorViewHide

CorViewUpdatePos

Updates the position of a view resource

Prototype CORSTATUS **CorViewUpdatePos**(CORVIEW *hVew*);

Description Updates the position of the view on the display surface without showing it. CorViewUpdatePos will take into account the current CORVIEW_PRM_ROI_SRC and CORVIEW_PRM_ROI_DST as well as the new target window position if the display is a system display and CORVIEW_PRM_HWND is not 0.

Input *hView* View resource handle

Output None

Return Value CORSTATUS_INVALID_HANDLE

See Also CorViewHide and CorViewShow

CORVIEW_BLIT_DESC Structure Definition

```
// CORVIEW_BLIT_DESC Structure Definition
typedef struct
{
    UINT32 roi_src_left;   //roi values are coordinates in pixels,
                               //relative to the respective buffers
    UINT32 roi_src_top;
    UINT32 roi_src_height;
    UINT32 roi_src_width;
    UINT32 roi_dst_left;
    UINT32 roi_dst_top;
    UINT32 roi_dst_height;
    UINT32 roi_dst_width;
    UINT32 rotation_angle; //rotation_angle can take values from 0 to 359
    UINT32 color_fill;     //color_fill should be in the same format as the
                               //destination buffer's pixel format
    UINT32 dst_key_color;  //key_color parameters should be in the same pixel //format
                               as the source and destination buffers
    UINT32 src_key_color;
 } CORVIEW_BLIT_DESC, *PCORVIEW_BLIT_DESC;
```

PCI Device Module

The PCI Device Module permits an application to manage the configuration space of PCI devices.

Functions

Function	Description
CorPciFindClassCode	*Find a PCI device with a specific class code*
CorPciFindDevice	*Find a PCI device with a specific vendor ID and device ID*
CorPciGetByte	*Read a byte from the configuration space of a PCI device*
CorPciGetData	*Read an array of data from the configuration space of a PCI device*
CorPciGetDword	*Read a double word from the configuration space of a PCI device*
CorPciGetInfo	*Get PCI BIOS information, such as the number of PCI buses, hardware mechanisms, and version*
CorPciGetVGADevice	*Get PCI device handle to a device that has VGA class code*
CorPciGetWord	*Read a word from the configuration space of a PCI device*
CorPciNewDevice	*Create a new PCI device handle given a PCI bus number, a PCI slot number, and a PCI function number*
CorPciPutByte	*Write a byte in the configuration space of a PCI device*
CorPciPutDword	*Write a double word in the configuration space of a PCI device*
CorPciPutWord	*Write a word in the configuration space of a PCI device*
CorPciRelease	*Release a PCI device*
CorPciSetBusNumber	*Force the number of PCI buses in the system*

CorPciFindClassCode

Find a PCI device with a specific class code

Prototype	CORSTATUS **CorPciFindClassCode**(CORSERVER *hServer*, UINT32 *classcode*, UINT16 *index*, CORPCIDEVICE **hPciDevice*);	
Description	Find a PCI device with a specific class code	
Input	*hServer*	Server handle
	classCode	PCI device class code
	index	PCI device index
Output	*hPciDevice*	PCI device handle
Return Value	CORSTATUS_ARG_NULL (if *hPciDevice* is NULL), CORSTATUS_INVALID_HANDLE	
See Also	CorPciFindDevice, CorPciGetVGADevice and CorPciNewDevice	

CorPciFindDevice

Find a PCI device with a specific vendor ID and device ID

Prototype	CORSTATUS **CorPciFindDevice**(CORSERVER *hServer*, UINT16 *vendorID*, UINT16 *deviceID*, UINT16 *index*, CORPCIDEVICE **hPciDevice*);	
Description	Find a PCI device with a specific vendor ID and device ID	
Input	*hServer*	Server handle
	vendorID	PCI device vendor ID
	deviceID	PCI device device ID
	index	PCI device index (usually 0)
Output	*hPciDevice*	PCI device handle
Return Value	CORSTATUS_ARG_NULL (if *hPciDevice* is NULL)	
Note	PCI device index must be used when more than one PCI device has the same vendor ID and device ID.	
See Also	CorPciFindClassCode, CorPciGetVGADevice and CorPciNewDevice	

CorPciGetByte

Read a byte from the configuration space of a PCI device

Prototype	CORSTATUS **CorPciGetByte**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, PUINT8 *data*);	
Description	Read a byte from the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
Output	*data*	Byte read from configuration space
Return Value	CORSTATUS_ARG_NULL (if *data* is NULL), CORSTATUS_INVALID_HANDLE	
See Also	CorPciGetData, CorPciGetDword and CorPciGetWord	

CorPciGetData

Read an array of data from the configuration space of a PCI device

Prototype	CORSTATUS **CorPciGetData**(CORPCIDEVICE *hPciDevice*, UINT16 *basereg*, UINT16 *nbytes*, void* *data*);	
Description	Read an array of data from the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	basereg	Offset in configuration space
	nbytes	Number of bytes to read
Output	*data*	Data read from configuration space
Return Value	CORSTATUS_ARG_NULL (if *data* is NULL), CORSTATUS_INVALID_HANDLE	
See Also	CorPciGetDword and CorPciGetWord	

CorPciGetDword

Read a double word from the configuration space of a PCI device

Prototype	CORSTATUS **CorPciGetDword**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, PUINT32 *data*);	
Description	Read a double word from the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
Output	*data*	Double word read from configuration space
Return Value	CORSTATUS_ARG_NULL (if *data* is NULL), CORSTATUS_INVALID_HANDLE	
See Also	CorPciGetData and CorPciGetWord	

CorPciGetInfo

Get PCI BIOS information

Prototype	CORSTATUS **CorPciGetInfo**(CORSERVER *hServer*, PUINT16 *version*, PUINT8 *mechanism*, PUINT8 *nBuses*);	
Description	Get information such as the number of PCI buses, hardware mechanisms, and version.	
Input	*hServere*	Server handle
Output	*version*	BIOS version
	mechanism	Hardware mechanism
	nBuses	Number of PCI buses
Return Value	CORSTATUS_ARG_NULL (if *version, mechanism or nBuses* is NULL) CORSTATUS_INVALID_HANDLE	

CorPciGetVGADevice

Gets PCI device handle to a device that has VGA class code

Prototype	CORSTATUS **CorPciGetVGADevice**(CORSERVER *hServer*, UINT16 *index*, CORPCIDEVICE **hPciDevice*);	
Description	Gets PCI device handle to a device that has a VGA class code	
Input	*hServer*	Server handle
	index	PCI device index (usually 0)
Output	*hPciDevice*	PCI device handle
Return Value	CORSTATUS_ARG_NULL (if *hPciDevice* is NULL) and CORSTATUS_INVALID_HANDLE	
See Also	CorPciFindClassCode, CorPciFindDevice and CorPciNewDevice	

CorPciGetWord

Read a word from the configuration space of a PCI device

Prototype	CORSTATUS **CorPciGetWord**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, PUINT16 *data*);	
Description	Read a word from the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
Output	*data*	Word read from configuration space
Return Value	CORSTATUS_ARG_NULL (if *data* is NULL) and CORSTATUS_INVALID_HANDLE	
See Also	CorPciGetData and CorPciGetDword	

CorPciNewDevice

Create a PCI device handle given a PCI bus number, a PCI slot number and a PCI function number

Prototype	CORSTATUS **CorPciNewDevice**(CORSERVER *hServer*, UINT8 *bus*, UINT8 *slot*, UINT8 *func*, CORPCIDEVICE **hPciDevice*);	
Description	Create a PCI device handle given a PCI bus number, a PCI slot number and a PCI function number	
Input	*hServer*	Server handle
	bus	Bus number
	slot	Slot number
	func	Function number
Output	*hPciDevice*	PCI device handle
Return Value	CORSTATUS_ARG_NULL (if *hPciDevice* is NULL) and CORSTATUS_INVALID_HANDLE	
See Also	CorPciFindClassCode, CorPciFindDevice and CorPciGetVGADevice	

CorPciPutByte

Write a byte in the configuration space of a PCI device

Prototype	CORSTATUS **CorPciPutByte**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, UINT8 *data*);	
Description	Write a byte in the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
	data	Byte to write in configuration space
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorPciPutDword and CorPciPutWord	

CorPciPutDword

Write a double word in the configuration space of a PCI device

Prototype	CORSTATUS **CorPciGetDword**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, PUINT32 *data*);	
Description	Write a double word in the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
	data	Double word to write in configuration space
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorPciPutByte and CorPciPutWord	

CorPciPutWord

Write a word in the configuration space of a PCI device

Prototype	CORSTATUS **CorPciPutWord**(CORPCIDEVICE *hPciDevice*, UINT16 *reg*, UINT16 *data*);	
Description	Write a word in the configuration space of a PCI device.	
Input	*hPciDevice*	PCI device handle
	reg	Offset in configuration space
	data	Word to write in configuration space
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorPciPutByte and CorPciPutDword	

CorPciRelease

Release a PCI device

Prototype	CORSTATUS **CorPciRelease**(CORPCIDEVICE *hPciDevice*);	
Description	Release a PCI device	
Input	*hPciDevice*	PCI device handle
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
See Also	CorPciNewDevice	

CorPciSetBusNumber

Force the number of PCI buses in the system

Prototype	CORSTATUS **CorPciSetBusNumber**(CORSERVER *hServer*, PUINT8 *nbuses*);	
Description	Force the number of PCI buses in the system	
Input	*hServer*	Server handle
	nbuses	Number of PCI buses
Output	None	
Return Value	CORSTATUS_INVALID_HANDLE	
Note	The number of PCI buses can only be incremented.	

Error Messages

Bit Description

All functions return a 32-bit CORSTATUS value, which is in the following format:

Bits	31-28	27-22	21-20	19-8	7-0
Description	Reserved	Module ID	Level	Info	Status ID

Bit Field	Description
Status ID	8-bits: a CORSTATUS_xxx constant. Where xxx references the status ID number.
Info	12-bits: a CORSTATUS_INFO_xxx constant or CORSTATUS_xxx constant. Refer to the status message descriptions for the Info field returned.
Level	2-bits: a CORSTATUS_LEVEL_xxx: constant.
Module ID	6-bits: a CORSTATUS_MODULE_xxx constant.

The CorManGetStatusTextEx function may be used to extract a description string for each of the bit fields. Refer to the "Error Management" section of the *Sapera LT User's Manual* for more details.

Status ID

ID	Message
0x00	CORSTATUS_OK
0x01	CORSTATUS_INVALID_HANDLE
0x0a	CORSTATUS_INCOMPATIBLE_ACQ
0x0b	CORSTATUS_INCOMPATIBLE_BUFFER
0x0c	CORSTATUS_INCOMPATIBLE_CAB
0x0d	CORSTATUS_INCOMPATIBLE_CAM
0x0e	CORSTATUS_INCOMPATIBLE_DISPLAY
0x0f	CORSTATUS_INCOMPATIBLE_GRAPHIC
0x10	CORSTATUS_INCOMPATIBLE_KERNEL
0x11	CORSTATUS_INCOMPATIBLE_LUT

0x12	CORSTATUS_INCOMPATIBLE_MANAGER
0x13	CORSTATUS_INCOMPATIBLE_PRO
0x14	CORSTATUS_INCOMPATIBLE_VIC
0x15	CORSTATUS_INCOMPATIBLE_VIEW
0x16	CORSTATUS_INCOMPATIBLE_XFER
0x17	CORSTATUS_INCOMPATIBLE_STRING
0x18	CORSTATUS_INCOMPATIBLE_OBJECT
0x1a	CORSTATUS_INCOMPATIBLE_FILE
0x1e	CORSTATUS_CAP_INVALID
0x1f	CORSTATUS_CAP_NOT_AVAILABLE
0x28	CORSTATUS_PRM_INVALID
0x29	CORSTATUS_PRM_NOT_AVAILABLE
0x2a	CORSTATUS_PRM_OUT_OF_RANGE
0x2b	CORSTATUS_PRM_INVALID_VALUE
0x2c	CORSTATUS_PRM_READ_ONLY
0x2d	CORSTATUS_PRM_MUTUALLY_EXCLUSIVE
0x32	CORSTATUS_ARG_INVALID
0x33	CORSTATUS_ARG_OUT_OF_RANGE
0x34	CORSTATUS_ARG_INCOMPATIBLE
0x35	CORSTATUS_ARG_INVALID_VALUE
0x36	CORSTATUS_ARG_NULL
0x39	CORSTATUS_FILE_OPTIONS_ERROR
0x3a	CORSTATUS_FILE_OPEN_MODE_INVALID
0x3b	CORSTATUS_FILE_SEEK_ERROR
0x3c	CORSTATUS_FILE_CREATE_ERROR
0x3d	CORSTATUS_FILE_OPEN_ERROR
0x3e	CORSTATUS_FILE_READ_ERROR
0x3f	CORSTATUS_FILE_WRITE_ERROR
0x40	CORSTATUS_FILE_CLOSE_ERROR
0x41	CORSTATUS_FILE_FORMAT_UNKNOWN
0x42	CORSTATUS_FILE_VALUE_NOT_SUPPORTED
0x43	CORSTATUS_FILE_GET_FIELD_ERROR
0x44	CORSTATUS_FILE_READ_ONLY
0x45	CORSTATUS_FILE_WRITE_ONLY
0x46	CORSTATUS_NOT_IMPLEMENTED
0x47	CORSTATUS_NO_MEMORY
0x48	CORSTATUS_CLIPPING_OCCURED

0x49	CORSTATUS_HARDWARE_ERROR
0x4a	CORSTATUS_SERVICE_NOT_AVAILABLE
0x4b	CORSTATUS_NOT_ACCESSIBLE
0x4c	CORSTATUS_NOT_AVAILABLE
0x4d	CORSTATUS_ROUTING_NOT_IMPLEMENTED
0x4e	CORSTATUS_ROUTING_NOT_AVAILABLE
0x4f	CORSTATUS_ROUTING_IN_USE
0x50	CORSTATUS_INCOMPATIBLE_SIZE
0x51	CORSTATUS_INCOMPATIBLE_FORMAT
0x53	CORSTATUS_INCOMPATIBLE_LOCATION
0x54	CORSTATUS_RESOURCE_IN_USE
0x55	CORSTATUS_RESOURCE_LINKED
0x56	CORSTATUS_SOFTWARE_ERROR
0x57	CORSTATUS_PARAMETERS_LOCKED
0x58	CORSTATUS_XFER_NOT_CONNECTED
0x59	CORSTATUS_XFER_EMPTY_LIST
0x5a	CORSTATUS_XFER_CANT_CYCLE
0x5b	CORSTATUS_ROUTING_NOT_SPECIFIED
0x5d	CORSTATUS_TRANSFER_IN_PROGRESS
0x5e	CORSTATUS_API_NOT_LOCKED
0x5f	CORSTATUS_SERVER_NOT_FOUND
0x60	CORSTATUS_CANNOT_SIGNAL_EVENT
0x61	CORSTATUS_NO_MESSAGE
0x62	CORSTATUS_TIMEOUT
0x63	CORSTATUS_INVALID_ALIGNMENT
0x64	CORSTATUS_DDRAW_256_COLORS
0x65	CORSTATUS_PCI_IO_ERROR
0x66	CORSTATUS_PCI_CANNOT_ACCESS_DEVICE
0x67	CORSTATUS_EVENT_CREATE_ERROR
0x68	CORSTATUS_BOARD_NOT_READY
0x69	CORSTATUS_XFER_MAX_SIZE
0x6a	CORSTATUS_PROCESSING_ERROR
0x6b	CORSTATUS_RESOURCE_LOCKED
0x6c	CORSTATUS_NO_MESSAGING_MEMORY
0x6d	CORSTATUS_DDRAW_NOT_AVAILABLE
0x6e	CORSTATUS_DDRAW_ERROR
0x6f	CORSTATUS_RESOURCE_NOT_LOCKED

0x70	CORSTATUS_DISK_ON_CHIP_ERROR
0x73	CORSTATUS_INSUFFICIENT_BANDWIDTH
0x74	CORSTATUS_FILE_TELL_ERROR
0x75	CORSTATUS_MAX_PROCESS_EXCEEDED
0x76	CORSTATUS_XFER_COUNT_MULT_SRC_FRAME_COUNT
0x77	CORSTATUS_ACQ_CONNECTED_TO_XFER
0x78	CORSTATUS_INSUFFICIENT_BOARD_MEMORY
0x79	CORSTATUS_INSUFFICIENT_RESOURCES
0x7a	CORSTATUS_MISSING_RESOURCE
0x7b	CORSTATUS_NO_DEVICE_FOUND
0x7c	CORSTATUS_RESOURCE_NOT_CONNECTED
0x7d	CORSTATUS_SERVER_DATABASE_FULL

CORSTATUS_ACQ_CONNECTED_TO_XFER

Definition The resource cannot be modified while the acquistion is connected to a transfer module.

Info The parameter number.

Note None

CORSTATUS_API_NOT_LOCKED

Definition The API must first be locked using the function CorManLock.

Info None

Note This error and the function CorManLock are now obsolete with Sapera 1.10 and up.

CORSTATUS_ARG_INCOMPATIBLE

Definition An incompatible argument was passed to a function.

Info The argument number.

Note Arguments are numbered starting at 1.

CORSTATUS_ARG_INVALID

Definition An invalid argument was passed to a function.

Info The argument number.

Note Arguments are numbered starting at 1.

CORSTATUS_ARG_INVALID_VALUE

Definition An argument with an invalid value was passed to a function.

Info The argument number.

Note Arguments are numbered starting at 1.

CORSTATUS_ARG_NULL

Definition A null argument was passed to a function.

Info The argument number.

Note Arguments are numbered starting at 1.

CORSTATUS_ARG_OUT_OF_RANGE

Definition An out-of-range argument was passed to a function.

Info The argument number.

Note Arguments are numbered starting at 1.

CORSTATUS_BOARD_NOT_READY

Definition Sapera has been unable to bind to the board device driver.

Info None

CORSTATUS_CANNOT_SIGNAL_EVENT

Definition While trying to send a message, the Sapera message engine got an error when signaling an event object.

Info None

CORSTATUS_CAP_INVALID

Definition An invalid capability was requested.

Info The capability number.

CORSTATUS_CAP_NOT_AVAILABLE

Definition An unavailable capability was requested.

Info The capability number.

CORSTATUS_CLIPPING_OCCURED

Definition A rectangle clipping occurred while processing a buffer.

Info None

CORSTATUS_DISK_ON_CHIP_ERROR

Definition There was an error accessing the Disk-On-Chip located on a Sapera board.

Info The Disk-On-Chip error code

CORSTATUS_DDRAW_256_COLORS

Definition The VGA graphics mode must be in 256 colors or more.

Info None

CORSTATUS_DDRAW_ERROR

Definition Direct Draw cannot provide the requested service.

Info None

CORSTATUS_DDRAW_NOT_AVAILABLE

Definition Direct Draw services are not available.

Info None

CORSTATUS_EVENT_CREATE_ERROR

Definition While trying to send a message, the Sapera message engine got an error when creating an event object.

Info None

CORSTATUS_FILE_CLOSE_ERROR

Definition Error closing file.

Info None

CORSTATUS_FILE_CREATE_ERROR

Definition Error creating file.

Info None

CORSTATUS_FILE_FIELD_VALUE_NOT_SUPPORTED

Definition One or more header field values from the specified image file are not supported.

Info None

CORSTATUS_FILE_FORMAT_UNKNOWN

Definition File format is unknown.

Info None

CORSTATUS_FILE_GET_FIELD_ERROR

Definition Failed to get header field information from the specified image.

Info None

CORSTATUS_FILE_OPEN_ERROR

Definition Error opening file.

Info None

CORSTATUS_FILE_OPEN_MODE_INVALID

Definition File open mode is invalid.

Info None

CORSTATUS_FILE_OPTIONS_ERROR

Definition File options is invalid.

Info None

CORSTATUS_FILE_READ_ERROR

Definition Error reading file.

Info None

CORSTATUS_FILE_READ_ONLY

Definition File can only be read.

Info None

CORSTATUS_FILE_SEEK_ERROR

Definition A seek error occurred while accessing the file.

Info None

CORSTATUS_FILE_TELL_ERROR

Definition A seek error occurred while trying to set the current position in the specified file.

Info None

CORSTATUS_FILE_VALUE_NOT_SUPPORTED

Definition A file field value not supported by the File Module.

Info None

CORSTATUS_FILE_WRITE_ERROR

Definition Error writing file.

Info None

CORSTATUS_FILE_WRITE_ONLY

Definition File can only be written.

Info None

CORSTATUS_HARDWARE_ERROR

Definition General hardware error. This error is usually fatal.

Info Hardware specific information about the error.

CORSTATUS_INCOMPATIBLE_ACQ

Definition An acquisition parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_BUFFER

Definition A buffer is incompatible with another API object, prohibiting the two from functioning together.

Info The reason for the incompatibility.

CORSTATUS_INCOMPATIBLE_CAB

Definition A CAB parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_CAM

Definition A camera parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_DISPLAY

Definition A display parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_FILE

Definition A file parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_FORMAT

Definition The formats of two or more resources are not compatible, prohibiting them from functioning together.

Info None

CORSTATUS_INCOMPATIBLE_GRAPHIC

Definition A graphic parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_KERNEL

Definition A kernel parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_LOCATION

Definition The location of two resources are incompatible.

Info None

CORSTATUS_INCOMPATIBLE_LUT

Definition An LUT parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_MANAGER

Definition A manager parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_OBJECT

Definition A processing object is incompatible with another API object.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_PRO

Definition A processing parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_SIZE

Definition The size of two or more resources are not compatible, prohibiting them from functioning together.

Info None

CORSTATUS_INCOMPATIBLE_STRING

Definition An error was detected while parsing the C expression in the string.

Info None

CORSTATUS_INCOMPATIBLE_VIC

Definition A VIC parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_VIEW

Definition A view parameter is incompatible.

Info The parameter number.

CORSTATUS_INCOMPATIBLE_XFER

Definition A transfer parameter is incompatible.

Info The parameter number.

CORSTATUS_INSUFFICIENT_BANDWIDTH

Definition The requested data transfer bandwidth exceeds the hardware capabilities.

Info None

CORSTATUS_INSUFFICIENT_BOARD_MEMORY

Definition There is insufficient memory on the acquisition board.

Info None

CORSTATUS_INSUFFICIENT_RESOURCES

Definition A requested or required resource cannot be obtained because there are none available.

Info None

CORSTATUS_INVALID_ALIGNMENT

Definition Memory block was not aligned on a 32-bit boundary.

Info None

CORSTATUS_INVALID_HANDLE

Definition Invalid handle.

Info CORSTATUS_MODULE_xxx constant.

CORSTATUS_MAX_PROCESS_EXCEEDED

Definition The maximum number of processes, that can be bound to a driver, has been exceeded.

Info None

CORSTATUS_NO_MEMORY

Definition There is not enough memory for the required allocation.

Info None

CORSTATUS_NO_MESSAGE

Definition Even if the Sapera messaging engine has been signaled, no message is available.

Info None

CORSTATUS_NO_MESSAGING_MEMORY

Definition There is not enough messaging memory for the required allocation.

Info None.

Note Use the Sapera Configuration utility to increase the amount of messaging memory.

CORSTATUS_NOT_ACCESSIBLE

Definition An error occurred while accessing any intermediate underlying implementation in the process of executing an API function.

Info None

CORSTATUS_NOT_AVAILABLE

Definition The resource is not available.

Info None

CORSTATUS_NOT_IMPLEMENTED

Definition The function is not implemented.

Info None

CORSTATUS_OK

Definition No error.

Info None

CORSTATUS_PARAMETERS_LOCKED

Definition Module's parameters are locked (cannot be written).

Info None

CORSTATUS_PCI_CANNOT_ACCESS_DEVICE

Definition PCI module was unable to access the specified PCI device.

Info None

Note This error is now obsolete with Sapera 1.20 and up.

CORSTATUS_PCI_IO_ERROR

Definition Reading or writing to the device's PCI configuration space failed.

Info None

CORSTATUS_PRM_INVALID

Definition An invalid, nonexistent parameter was specified.

Info None

CORSTATUS_PRM_INVALID_VALUE

Definition A parameter could not be set because the value is invalid.

Info The parameter number.

CORSTATUS_PRM_MUTUALLY_EXCLUSIVE

Definition A parameter could not be set because it is mutually exclusive with another parameter.

Info The parameter number.

CORSTATUS_PRM_NOT_AVAILABLE

Definition A parameter could not be read or written because it is unavailable, usually because the capability governing it is not available.

Info The parameter number.

CORSTATUS_PRM_OUT_OF_RANGE

Definition A parameter could not be set because the value is out of range.

Info The parameter number.

CORSTATUS_PRM_READ_ONLY

Definition A parameter could not be written because it is read only.

Info The parameter number.

CORSTATUS_PROCESSING_ERROR

Definition An error occurred in the low-level image processing library.

Info None

CORSTATUS_RESOURCE_IN_USE

Definition A requested or required resource is already being used by the user or the API.

Info None

CORSTATUS_RESOURCE_LINKED

Definition The resource is linked to another resource and cannot be freed.

Info None

CORSTATUS_RESOURCE_LOCKED

Definition The resource is locked and cannot be modified. Array module functions return this status ID when the parameter LOCKED is set.

Info None

CORSTATUS_RESOURCE_NOT_LOCKED

Definition The resource needs to be locked before use.

Info None

CORSTATUS_ROUTING_IN_USE

Definition The transfer routing (path) is already used.

Info None

CORSTATUS_ROUTING_NOT_AVAILABLE

Definition The transfer routing (path) is not available.

Info None

CORSTATUS_ROUTING_NOT_IMPLEMENTED

Definition The transfer routing (path) is not implemented.

Info None

CORSTATUS_ROUTING_NOT_SPECIFIED

Definition The transfer routing cannot be established because there is no source/destination pair.

Info None

CORSTATUS_SERVER_NOT_FOUND

Definition The requested server was not found. The server name may not be present in the Sapera Server list. Check in the Sapera configuration program to verify the presence of the server. If present, it may not be responding.

Info None

CORSTATUS_SERVICE_NOT_AVAILABLE

Definition Windows service is not running.

Info None

CORSTATUS_SOFTWARE_ERROR

Definition General software error.

Info Software specific information about the error.

CORSTATUS_TIMEOUT

Definition There was no response from a module.

Info CORSTATUS_MODULE_xxx constant.

CORSTATUS_TRANSFER_IN_PROGRESS

Definition Operation could not be performed because a transfer is in progress.

Info None

CORSTATUS_XFER_CANT_CYCLE

Definition No transfer cycle is possible for the current destination resource.

Info None

CORSTATUS_XFER_COUNT_MULT_SRC_FRAME_COUNT

Definition The number of frames to acquire (count argument of function CorXferStart) is not a multiple of the number of frames output by the source. For example, if the source outputs 3 frames per external trigger, the number of frames to acquire needs to be a multiple of 3.

Info None

CORSTATUS_XFER_EMPTY_LIST

Definition There is no data to transfer.

Info None

CORSTATUS_XFER_MAX_SIZE

Definition The size of the requested transfer is greater than the maximum. Use the capability CORXFER_CAP_MAX_XFER_SIZE to get this maximum.

Info None

CORSTATUS_XFER_NOT_CONNECTED

Definition The transfer is not connected.

Info None

CORSTATUS_MISSING_RESOURCE

Definition A required supporting DLL resource is missing

Info None

CORSTATUS_NO_DEVICE_FOUND

Definition No Sapera compatible device was found

Info None

CORSTATUS_RESOURCE_NOT_CONNECTED

Definition The resource is not connected

Info None

CORSTATUS_SERVER_DATABASE_FULL

Definition The internal database containing the list of servers is full.

Info None

Level

ID	Value	Definition
0x00	CORSTATUS_LEVEL_ERR	Error
0x01	CORSTATUS_LEVEL_FAT	Fatal error
0x02	CORSTATUS_LEVEL_WRN	Warning
0x03	CORSTATUS_LEVEL_INF	Information

Module ID

ID	Value	Module name
0x01	CORSTATUS_MODULE_ACQ	Acquisition module
0x02	CORSTATUS_MODULE_BUFFER	Buffer module
0x03	CORSTATUS_MODULE_CAB	CAB module
0x04	CORSTATUS_MODULE_CAM	Camera module
0x05	CORSTATUS_MODULE_DISPLAY	Display module
0x06	CORSTATUS_MODULE_FONT	Font module
0x07	CORSTATUS_MODULE_GRAPHIC	Graphic module
0x08	CORSTATUS_MODULE_HOST	Host module
0x09	CORSTATUS_MODULE_KERNEL	Kernel module
0x0a	CORSTATUS_MODULE_LOG	Log module
0x0b	CORSTATUS_MODULE_LUT	LUT module
0x0c	CORSTATUS_MODULE_MANAGER	API control module
0x0d	CORSTATUS_MODULE_MEMORY	Memory management module
0x0e	CORSTATUS_MODULE_PCI	PCI module
0x0f	CORSTATUS_MODULE_PORT	Port module
0x10	CORSTATUS_MODULE_PRO	Processing module
0x11	CORSTATUS_MODULE_VIC	VIC module
0x12	CORSTATUS_MODULE_VIEW	View module
0x13	CORSTATUS_MODULE_XFER	Transfer module
0x14	CORSTATUS_MODULE_VDI	Video display interface module
0x15	CORSTATUS_MODULE_SERVER	Server module

ID	Value	Module name
0x16	CORSTATUS_MODULE_PIXPRO	Pixel processor module
0x17	CORSTATUS_MODULE_FILE	File module
0x18	CORSTATUS_MODULE_C60	C60 module
0x19	CORSTATUS_MODULE_GIO	General IO module
0x1a	CORSTATUS_MODULE_COUNTER	Counter module
0x1b	CORSTATUS_MODULE_ARRAY	Array module
0x1c		(reserved)
0x1d		(reserved)
0x1e		(reserved)
0x1f		(reserved)
0x20	CORSTATUS_MODULE_EVENTINFO	Event information module
0x21	CORSTATUS_MODULE_FEATURE	Feature module
0x22	CORSTATUS_MODULE_ACQDEVICE	Acquisition device module

Macro Definitions

Sapera Macros

This section describes all the macros used in Sapera. The macros should always be used within user applications to ensure code portability.

VALIDATE_HANDLE_ACQ(CORACQ hAcq)

Definition Validates acquisition module handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_ACQDEVICE(CORACQDEVICE hAcqDevice)

Definition Validates acquisition device module handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_BUFFER(CORBUFFER hBuffer)

Definition Validates buffer resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_CAB(CORCAB hCab)

Definition Validates DALSA Coreco Auxiliary Bus device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_CAM(CORCAM hCam)

Definition Validates camera resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_COUNTER(CORCOUNTER hCounter)

Definition Validates counter device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_DISPLAY(CORDISPLAY hDisplay)

Definition Validates display device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_EVENTINFO(COREVENTINFO hEventInfo)

Definition Validates event information resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_FEATURE(CORFEATURE hFeature)

Definition Validates feature resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_FILE(CORFILE hFile)

Definition Validates file resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_GIO(CORGIO hGIO)

Definition Validates global input/output device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_GRAPHIC(CORGRAPHIC hGraphic)

Definition Validates graphic device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_KERNEL(CORKERNEL hKernel)

Definition Validates kernel resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_LUT(CORLUT hLut)

Definition Validates lookup table resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_OBJECT(COROBJECT hObject)

Definition Validates object resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_PCI_DEVICE(CORPCIDEVICE hPciDevice)

Definition Validates PCI device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_PIXPRO(CORPIXPRO hPixPro)

Definition Validates pixel processor device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_PRO(CORPRO hPro)

Definition Validates processor device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_VIC(CORVIC hVIC)

Definition Validates video input conditioning resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_VIEW(CORVIEW hView)

Definition Validates view resource handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

VALIDATE_HANDLE_XFER(CORXFER hXfer)

Definition Validates transfer device handle

Returns CORSTATUS_OK if the handle is valid, CORSTATUS_INVALID_HANDLE otherwise

CORHANDLE_NULL

Definition Should be used to initialize non-allocated handles.

Data Definitions

Data Types

This section describes all the data types used in Sapera. They should always be used in your applications to ensure code portability.

BOOLEAN

```
8-bits unsigned integer
```

CORCOUNT

```
typedef union
{
    UNIT8 count8;
    UNIT16 count16;
    UNIT32 count32;
    UNIT64 count64;
} CORCOUNT, *PCORCOUNT;
```

CORDATA

```
typedef union
{
    INT32 mono;
    struct  {
        INT32 red;
        INT32 green;
        INT32 blue;
    } rgb;
    struct
    {
        INT32 h;
        INT32 s;
        INT32 i;
    } hsi;
    struct
    {
        INT32 h;
        INT32 s;
        INT32 v;
    } hsv;
    struct
    {
        INT32 y;
        INT32 u;
        INT32 v;
```

```
    } yuv;
    struct
    {
        INT32 x;
        INT32 y;
    } point;
    FLOAT flt;
    struct
    {
        UINT16 alpha;
        UINT16 red;
        UINT16 green;
        UINT16 blue;
    } rgba;
    struct
    {
        FLOAT real;
        FLOAT imag;
    } cplx;
    struct
    {
        FLOAT x;
        FLOAT y;
    } fpoint;
    struct
    {
        FLOAT red;
        FLOAT green;
        FLOAT blue;
    } frgb;
 } CORDATA, *PCORDATA;
```

CORPOINT

```
typedef struct {
    INT32 x;
    INT32 y;
} CRL_POINT, *PCRL_POINT;
```

CORSTATUS

32-bits unsigned integer

INT8

8-bits signed integer

INT16

16-bits signed integer

INT32

32-bits signed integer

PBOOLEAN

Pointer to an 8-bit unsigned integer

PCORCALLBACK

```
typedef CORSTATUS (CCONV *PCORCALLBACK) (void *context, UINT32 eventType, UINT32
eventCount);
```

PCOREVENTINFOCALLBACK

```
typedef CORSTATUS (CCONV *PCOREVENTINFOCALLBACK) (void *context, COREVENTINFO
hEventInfo);
```

PCORMANCALLBACK

```
typedef CORSTATUS (CCONV *PCORMANCALLBACK) (UINT32 cmd, void *inData, UINT32
inDataSize, void *outData, UINT32 outDataSize);
```

PINT8

Pointer to an 8-bits signed integer

PINT16

Pointer to a 16-bits signed integer

PINT32

Pointer to a 32-bits signed integer

PUINT8

Pointer to an 8-bits unsigned integer

PUINT16

Pointer to a 16-bits unsigned integer

PUINT32

Pointer to a 32-bits unsigned integer

UINT8

8-bits unsigned integer

UINT16

16-bits unsigned integer

UINT32

32-bits unsigned integer

Data Formats

This section describes all the data formats supported in Sapera. These formats are used by the Buffer, Kernel, File, and Acquisition modules. Each of these modules refers to these formats with different parameter names, although they are completely compatible. For example, the following assignment is acceptable.

```
UINT32 format;
CorBufferGetPrm(hBuffer, CORBUFFER_PRM_FORMAT, &format);// Read buffer format
CorAcqSetPrm(hAcq, CORACQ_PRM_OUTPUT_FORMAT, format);       // Force it to Acq
```

Note: The LUT module does not use these formats.
It has its own predefined formats that are not compatible with the ones mentioned above.

Data Formats:

COMPLEX	MONO1	RGB101010	YUV
FLOAT	MONO8	RGB161616	YUY2
FPOINT	MONO16	RGBP8	YVYU
HSI	MONO32	RGBP16	YUYV
HSIP8	POINT	UINT1	Y211
HSV	RGB5551	UINT8	Y411
INT8	RGB565	UINT16	
INT16	RGB888	UINT32	
INT32	RGB8888	UYVY	

COMPLEX

Related Parameter Values	CORBUFFER_VAL_FORMAT_COMPLEX
Number of Components	2
Number of Bits	32 per component, 64 total
Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e–38
Bit Organization	0-31: Real component 32-63: Imaginary component
Note	Represents a pair of floating-point numbers. This data format is always *signed*.

FLOAT

Related Parameter Values	CORBUFFER_VAL_FORMAT_FLOAT CORKERNEL_VAL_FORMAT_FLOAT
Number of Components	1
Number of Bits	32

Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e–38
Note	Represents a single floating-point number. This data format is always *signed*.

FPOINT

Related Parameter Values	CORBUFFER_VAL_FORMAT_FPOINT
Number of Components	2
Number of Bits	32 per component, 64 total
Value Range	Maximum representable: +/-3.402823466e+38 Minimum positive value: 1.175494351e–38
Bit Organization	0-31: X component 32-63: Y component
Note	Represents a pair of float. It is usually used for storing image coordinates. This data format is always *signed*.

HSI

Related Parameter Values	CORBUFFER_VAL_FORMAT_HSI
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Intensity component 8-15: Saturation component 16-23: Hue component 24-31: Alpha channel
Note	Represents a HSI color value.

HSIP8

Related Parameter Values	CORBUFFER_VAL_FORMAT_HSIP8
Number of Components	1
Number of Pages	3
Number of Bits	8 per component
Value Range	[0...255]
Note	Represents a planar HSI color value.

HSV

Related Parameter Values	CORBUFFER_VAL_FORMAT_HSV
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Value component 8-15: Saturation component

	16-23: Hue component 24-31: Alpha channel
Note	Represents a HSV color value.

INT8

Related Parameter Values	CORBUFFER_VAL_FORMAT_INT8
Number of Components	1
Number of Bits	8
Value Range	[-128...127]
Note	Represents a single monochrome value.

INT16

Related Parameter Values	CORBUFFER_VAL_FORMAT_INT16
Number of Components	1
Number of Bits	16 (pixel depth can range from 9 to 16)
Value Range	[-32768,32767]
Note	Represents a single monochrome value.

INT32

Related Parameter Values	CORBUFFER_VAL_FORMAT_INT32 CORKERNEL_VAL_FORMAT_INT32
Number of Components	1
Number of Bits	32
Value Range	[-2147483648...2147483647]
Note	Represents a single monochrome value.

MONO1

Related Parameter Values	CORBUFFER_VAL_FORMAT_MONO1
Number of Components	1
Number of Bits	1
Value Range	[0...1] (*unsigned*)
Note	Represents a single monochrome value.

MONO8

Related Parameter Values	CORBUFFER_VAL_FORMAT_MONO8 CORACQ_VAL_OUTPUT_FORMAT_MONO8
Number of Components	1
Number of Bits	8
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)

Note	Represents a single monochrome value.

MONO16

Related Parameter Values	CORBUFFER_VAL_FORMAT_MONO16 CORACQ_VAL_OUTPUT_FORMAT_MONO16
Number of Components	1
Number of Bits	16 (pixel depth can range from 9 to 16)
Value Range	[0...65535] (*unsigned*) [-32768,32767] (*signed*)
Note	Represents a single monochrome value.

MONO32

Related Parameter Values	CORBUFFER_VAL_FORMAT_MONO32 CORKERNEL_VAL_FORMAT_MONO32 CORACQ_VAL_OUTPUT_FORMAT_MONO32
Number of Components	1
Number of Bits	32
Value Range	[0...4294967295] (*unsigned*) [-2147483648...2147483647] (*signed*)
Note	Represents a single monochrome value.

POINT

Related Parameter Values	CORBUFFER_VAL_FORMAT_POINT
Number of Components	2
Number of Bits	32 per component, 64 total
Value Range	[-2147483648...2147483647]
Bit Organization	0-31: X component 32-63: Y component
Note	Represents a pair of integers. It is usually used for storing image coordinates. This data format is always *signed*.

RGB5551

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB5551 CORACQ_VAL_OUTPUT_FORMAT_RGB5551
Number of Components	3
Number of Bits	5 per component, 16 total
Value Range	[0...31] (*unsigned*) [-16...15] (*signed*)
Bit Organization	0-4: Blue component 5-9: Green component 10-14: Red component 15: 1-bit alpha channel
Note	Represents a RGB color value.

RGB565

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB565 CORACQ_VAL_OUTPUT_FORMAT_RGB565
Number of Components	3
Number of Bits	5, 6, 5 (for red, green and blue components respectively),16 total
Value Range	Red/blue: [0...31] (*unsigned*), [-16...15] (*signed*) Green: [0...63] (*unsigned*), [-32...31] (*signed*)
Bit Organization	0-4: Blue component 5-10: Green component 11-15: Red component
Note	Represents a RGB color value.

RGB888

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB888 CORACQ_VAL_OUTPUT_FORMAT_RGB888
Number of Components	3
Number of Bits	8 per component, 24 total
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Blue component 8-15: Green component 16-23: Red component
Note	Represents a RGB color value.

RGB8888

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB8888 CORACQ_VAL_OUTPUT_FORMAT_RGB8888
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255] (*unsigned*) [-128...127] (*signed*)
Bit Organization	0-7: Blue component 8-15: Green component 16-23: Red component 24-31: Alpha channel
Note	Represents a RGB color value.

RGB101010

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB101010 CORACQ_VAL_OUTPUT_FORMAT_RGB101010
Number of Components	3
Number of Bits	10 per component, 32 total
Value Range	[0...1023] (*unsigned*) [-512...511] (*signed*)

Bit Organization	0-9: Blue component 10-19: Green component 20-29: Red component 30-31: Not used
Note	Represents a RGB color value.

RGB161616

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB161616 CORACQ_VAL_OUTPUT_FORMAT_RGB161616
Number of Components	3
Number of Bits	16 per component, 48 total (pixel depth can range from 9 to 16)
Value Range	[0...65535] (*unsigned*) [-32768...32767] (*signed*)
Bit Organization	0-15: Blue component 16-31: Green component 32-47: Red component
Note	Represents a RGB color value.

RGB16161616

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGB16161616CORACQ_VAL_OUTPUT_FORMAT_RGB161616
Number of Components	4
Number of Bits	16 per component, 64 total (pixel depth can range from 9 to 16)
Value Range	[0...65535] (unsigned) [-32768...32767] (signed)
Bit Organization	0-15: Blue component 16-31: Green component 32-47: Red component 48-63: Alpha component
Note	Represents a RGBA color value.

RGBP8

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGBP8 CORACQ_VAL_OUTPUT_FORMAT_RGBP8
Number of Components	1
Number of Pages	3
Number of Bits	8
Value Range	[0...255]
Note	Represents a planar RGB value

RGBP16

Related Parameter Values	CORBUFFER_VAL_FORMAT_RGBP16 CORACQ_VAL_OUTPUT_FORMAT_RGBP16
Number of Components	1
Number of Pages	3
Number of Bits	16 (pixel depth can range from 9 to 16)
Value Range	[0...65535]
Note	Represents a planar RGB value

UINT1

Related Parameter Values	CORBUFFER_VAL_FORMAT_UINT1 CORBUFFER_VAL_FORMAT_BINARY
Number of Components	1
Number of Bits	1
Value Range	[0...1]
Note	Represents a single monochrome value.

UINT8

Related Parameter Values	CORBUFFER_VAL_FORMAT_UINT8
Number of Components	1
Number of Bits	8
Value Range	[0...255]
Note	Represents a single monochrome value.

UINT16

Related Parameter Values	CORBUFFER_VAL_FORMAT_UINT16
Number of Components	1
Number of Bits	16 (pixel depth can range from 9 to 16)
Value Range	[0...65535]
Note	Represents a single monochrome value.

UINT32

Related Parameter Values	CORBUFFER_VAL_FORMAT_UINT32
Number of Components	1
Number of Bits	32
Value Range	[0...4294967295]
Note	Represents a single monochrome value.

UYVY

Related Parameter Values	CORBUFFER_VAL_FORMAT_UYVY CORACQ_VAL_OUTPUT_FORMAT_UYVY
Number of Components	3
Number of Bits	8 per component (16 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: U_0 8–15: Y_0 Second element: 0-7: V_0 8-15: Y_1
Note	This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YUV

Related Parameter Values	CORBUFFER_VAL_FORMAT_YUV
Number of Components	3
Number of Bits	8 per component, 32 total
Value Range	[0...255]
Bit Organization	0-7: Y component 8-15: U component 16-23: V component 24-31: Alpha channel
Note	Represents a YUV color value.

YUY2

Related Parameter Values	CORBUFFER_VAL_FORMAT_YUY2 CORACQ_VAL_OUTPUT_FORMAT_YUY2
Number of Components	3
Number of Bits	8 per component (16 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: U_0 Second element: 0-7: Y_1 8-15: V_0

Note	Alias for the YUYV format. This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YVYU

Related Parameter Values	CORBUFFER_VAL_FORMAT_YVYU CORACQ_VAL_OUTPUT_FORMAT_YVYU
Number of Components	3
Number of Bits	8 per component, effectively 16 per element
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: V_0 Second element: 0-7: Y_1 8-15: U_0
Note	This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

YUYV

Related Parameter Values	CORBUFFER_VAL_FORMAT_YUYV CORACQ_VAL_OUTPUT_FORMAT_YUYV
Number of Components	3
Number of Bits	8 per component (16 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: U_0 Second element: 0-7: Y_1 8-15: V_0
Note	Alias for the YUY2 format. This is a 4:2:2 subsampled format in which for every two luminance components (Y) there is one set of color components (U, V). At least two consecutive elements (an UINT32) are needed to retrieve all the information for the individual components.

Y211

Related Parameter Values	CORBUFFER_VAL_FORMAT_Y211 CORACQ_VAL_OUTPUT_FORMAT_Y211
Number of Components	3
Number of Bits	8 per component (8 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: Y_0 8–15: U_0 Second element: 0-7: Y_2 8-15: V_0
Note	This is a 4:2:2 subsampled format with Y subsampled by 2 which for every four luminance components (Y) there is one set of color components (U, V). At least 4 consecutive elements are needed (an UINT32s) to retrieve all the information for the individual components.

Y411

Related Parameter Values	CORBUFFER_VAL_FORMAT_Y411 CORACQ_VAL_OUTPUT_FORMAT_Y411
Number of Components	3
Number of Bits	8 per component, (12 per element)
Value Range	Y: [0...255] U: [-128...127] V: [-128...127]
Bit Organization	First element: 0–7: U_0 8–15: Y_0 Second element: 0-7: V_0 8-15: Y_1
Note	This is a 4:1:1 subsampled format in which for every four luminance components (Y) there is one set of color components (U, V). At least 8 consecutive elements are needed (3 UINT32s) to retrieve all the information for the individual components.

Appendix A: Server Management

The Server Database

The section Working with Handles gives only a quick overview of how Sapera manages servers. Additional issues often need to be considered, especially when running in a Windows environment, whether on a host computer or on a remote server, such as Mamba. Some basic knowledge of the Sapera Server database is required in order to explain these concepts.

When Windows boots up, a list of all available Sapera Servers is built into Sapera's Manager module on the host computer. This list is called the "*Server database*". It contains the following types and numbers of entries:

- The 'System' entry is always present in the database. It corresponds to the host computer.
- For any Sapera-compatible board (e.g., Viper-Digital, Viper-Quad, Mamba...) physically present in the system, there is at least one entry in the database. This entry is represented by the name "BoardName_x" where "x" is a numerical value ranging from 1 to the number of boards of this type (e.g., Viper_Digital_1, Mamba_1, Mamba_2, Mamba_3, Python_1, …).
- Any board that has multiple onboard processors has additional entries that correspond to the processors. These entries are called child servers (e.g., Python_1_C60_1, Python_1_C60_2, Python_1_C60_3, Python_1_C60_4).

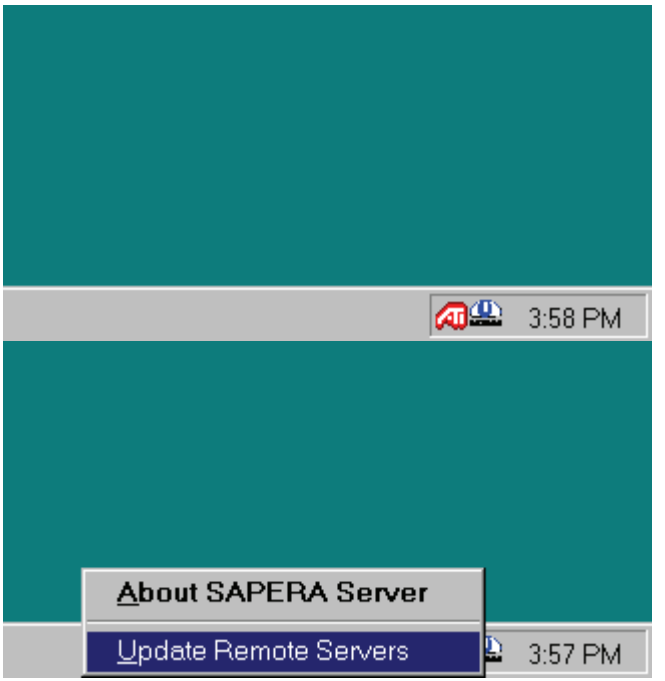
The database is made available to all application programs that are using Sapera. Use the **SapConf.exe** program to look up the contents of the database.

The Sapera Server Service

The *Sapera Server* (**SapServer.exe**) is an operating system service used for transferring the server database from the host computer to all Sapera boards on which this service runs (i.e., Sapera boards that support Sapera applications, such as Mamba). The Sapera Server service is installed only if required and is automatically launched by the operating system at boot up. Frame grabber boards, such as Bandit-II and the Viper Series, do not require *Sapera Server*.

They are several reasons why the server database may become lost on the Sapera boards, such as a Sapera application crashing on the board or a board that is rebooted.

It is then possible for the Sapera Server running on the host to send back the server database. To do so, right-click on the Sapera Server icon in the taskbar tray and select **Update Remote Servers** as shown in the following screen shots.



Additional Servers

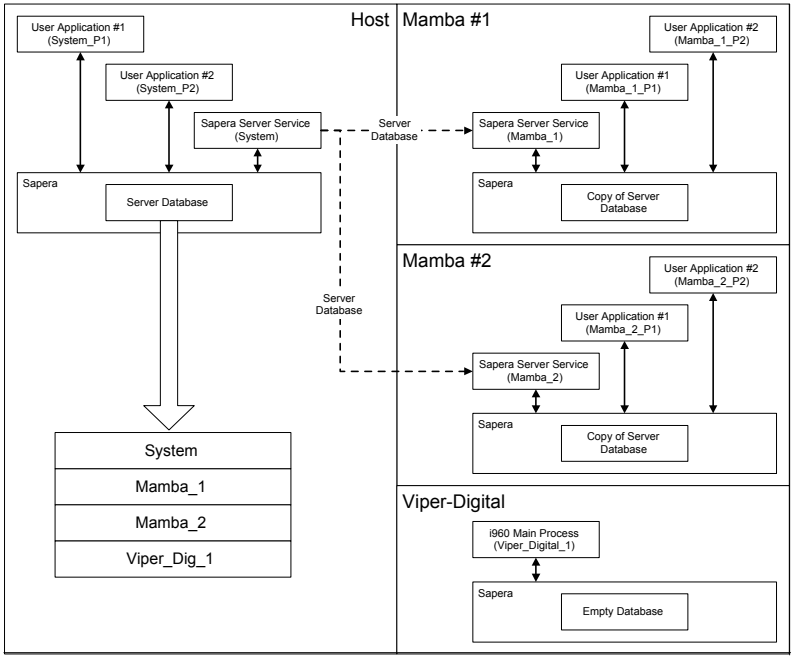
Not all servers are listed in the database. When running multiple Win32 applications at the same time, a new server is dynamically created for each application for the duration of the process only. The server for the first process will have the same name as the server listed in the database for the current Win32 environment. For example, on the host, the name corresponds to the “System” server. If the

computer has at least one Mamba installed, this first process is the Sapera Server. Servers for all other processes running Sapera applications are not part of the database.

Although servers listed in the database have recognized names that can be used directly, this is not the case for servers corresponding to Win32 applications. Sapera, however, permits each server to be assigned an alias in the form of a text string that will allow applications to retrieve any needed server handle at all times.

Server Management Diagram

The diagram below illustrates server management in a system containing two Mambas and a Viper-Digital. In the host section of the diagram, the Sapera Server is the first Sapera process initiated and inherits the name “System”. This is followed by two other applications running on the same platform and being assigned the names “System_P1” and “System_P2” respectively. These servers are not included in the database. The two Mambas behave almost exactly the same way except that the server database is a copy obtained from the host through the Sapera Server service. By distributing the server database throughout the entire system, every application is aware of all the system components. Note that the Viper-Digital does not need to receive the server database since there is no user application running on it.



Getting a Server Handle (revisited)

The previous information in this section illustrated the basics of getting server handles. The following looks more extensively into the different methods of getting server handles.

Get the server corresponding to the currently running Win32 process:

```
CORSTATUS status;                  // Declare status code
CORSERVER hCurrentServer;          // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// Get the server handle for this process
status = CorManGetLocalServer(&hCurrentServer);
```

Use the following method if the server's database index is known:

```
CORSTATUS status;     // Declare status code
UINT32 nCount;        // Declare a server count
UINT32 nIndex;        // Declare a server index
char szName[64];      // Declare a character string for returned name
CORSERVER hServer;    // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// Get the number of servers in the database
status = CorManGetServerCount(&nCount);

// Get the server handle from a database index
// The indices start at 0 (which is always 'System')
status = CorManGetServerByIndex(nIndex, szName, &hServer);
```

Use the server's database name directly if it is known:

```
CORSTATUS status;                  // Declare status code
CORSERVER hMambaServer;            // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// Get the server handle by specifying its database name
status = CorManGetServerByName("Mamba_1", &hMambaServer);
```

Note: it is important to be cautious in the way handles are retrieved or unwanted handles may be the result. The following example assumes the code runs on a Remote Server (e.g., Mamba Server):

```
CORSTATUS status;                  // Declare status code
CORSERVER hMambaServer;            // Declare a server handle
CORSERVER hLocalServer;            // Declare a server handle
```

```
// Initialize Sapera API
status = CorManOpen();

// Get the server handle for this process
hLocalServer = CorManGetLocalServer();

// Get the server handle by specifying its database name
status = CorManGetServerByName("Mamba_1", &hMambaServer);

if (hCurrentServer == hMambaServer)
{
   // This code will never be reached !
   // hLocalServer corresponds to the current process (Mamba_1_P?)
   // hMambaServer corresponds to the process for SapServer.exe (Mamba_1)
}
```

Communicating between Processes

Sending Sapera commands from one process to another is occasionally desirable. Default server names for individual processes are not documented since they may change in the future. However, using an alias for a server will achieve portable behavior.

For example, there are two processes on a Remote Server called the Producer and the Consumer. Sapera commands can be sent between the two as follows:

```
CORSTATUS status;                 // Declare status code
CORSERVER hConsumerServer;        // Declare a server handle

// This code runs in the Producer process
// Define an alias for this process
status = CorManSetLocalServerName("Producer");

// Get the Consumer server handle by specifying its alias
status = CorManGetServerByName("Consumer", &hConsumerServer);

// Send a Sapera command to the Consumer
status = CorManUserCmd(hConsumerServer, …);
```

```
CORSTATUS status;                 // Declare status code
CORSERVER hProducerServer;        // Declare a server handle

// This code runs in the Consumer process
// Define an alias for this process
status = CorManSetLocalServerName("Consumer");

// Get the Producer server handle by specifying its alias
status = CorManGetServerByName("Producer", &hProducerServer);

// Send a Sapera command to the Producer
status = CorManUserCmd(hProducerServer, …);
```

The host can then have a process that accesses both the Producer and the Consumer on the Remote server. Note that CorManGetServerByName cannot be used to get these handles since they correspond to individual processes other than the Sapera Server (and are therefore not present on the server database).

```
CORSTATUS status;                  // Declare status code
CORSERVER hMambaServer;            // Declare a server handle
CORSERVER hProducerServer;         // Declare a server handle
CORSERVER hConsumerServer;         // Declare a server handle

// Initialize Sapera API
status = CorManOpen();

// This code runs on the host
// First get the Mamba server handle by specifying its database name
status = CorManGetServerByName("Mamba_1", &hMambaServer);

// Get the Producer and Consumer server handles by specifying their aliases
status = CorManGetRemoteServerByName(hMambaServer, "Producer", &hProducerServer);
status = CorManGetRemoteServerByName(hMambaServer, "Consumer", &hConsumerServer);

// Send Sapera commands to the producer and the consumer
status = CorManUserCmd(hProducerServer, …);
status = CorManUserCmd(hConsumererServer, …);
```

Appendix B: File Formats

Buffer file formats

This section describes some buffer file formats supported in the Sapera File Module as implemented by the functions **CorFileLoad** and **CorFileSave**.

CORFILE_VAL_FORMAT_CRC

DALSA file format

Offset	Size	Description
0	UINT32	Magic Number (must be 0x1A435243)
4-12	UINT32	Reserved
16	UINT32	Buffer width in pixels
20	UINT32	Buffer height in lines
24	UINT32	ROI's horizontal minimum
28	UINT32	ROI's vertical minimum
32	UINT32	ROI's horizontal length
36	UINT32	ROI's vertical length
40-60	UNIT32	Reserved
64	UINT32	Number of bytes per pixel
68	UINT32	Number of bits per pixel
72	UINT32	Number of planes
76-152	UNIT32	Reserved
156	M	Buffer data

Where **M** is given by the following expression:
(**ROI's horizontal length** × **ROI's vertical length** × **Number of bytes per pixel** x **Number of planes**)

CORFILE_VAL_FORMAT_RAW

Raw file format

Offset	Size	Description
N	M	Buffer data

N (in bytes) is the location of the buffer data.

M is given by the following expression:
(**horizontal buffer length** × **vertical buffer length** × **Number of bytes per pixel**)

CORFILE_VAL_FORMAT_BMP

Windows Bitmap file format

Offset	Size	Description
0	UINT16	Magic Number (must be ASCII "BM")
2	UINT32	Size in bytes of the file
6	UINT16	Reserved
8	UINT16	Reserved
10	UINT32	Byte offset in files where image begins
14	UINT32	Size of the BITMAPINFO header
18	INT32	Image width in pixels
22	INT32	Image height in pixels
26	UINT16	Number of image planes, must be 1
28	UINT16	Total bits per pixels, 1, 4, 8, 16, 24, 32
30	UINT32	Compression type BI_RGB – none BI_RLE4 – RLE 4 bit BI_RLE8 – RLE 8 bit BI_BITFIELDS - Bitfields
34	UINT32	Size in bytes of compressed image, or zero
38	INT32	Horizontal resolution, in pixel/meter
42	INT32	Vertical resolution, in pixel/meter
46	UINT32	Number of colors used
50	UINT32	Number of important colors
54	RGBQUAD * N	Color map
54+4*N	M	Bitmap Data

- Where **N** is the number of colors used and **M** the number of bitmap data in bytes.
- If bpp (bits per pixel) is 24, N = 0.

- If bpp is 16 or 32,
  If the compression specification is BI_BITFIELDS (Bitfields), N = 3.
  If the compression specification is BI_RGB (uncompressed), N = 0.
- Otherwise, N = (1 << bpp).

Buffer Data Formats Supported as Input by FileSave Functions

Buffer Data Format	File Format					
	BMP	TIF	CRC	RAW	JPEG	AVI uncompressed
COMPLEX			X	X		
FLOAT			X	X		
FPOINT			X	X		
HSI			X	X		
HSIP8			X	X		
HSV			X	X		
INT8	X (1)	X	X	X		
INT16	X (1)	X	X	X		
INT32			X	X		
MONO1	X	X	X	X		X
MONO8	X	X	X	X	X	X
MONO16	X (1)	X	X	X		X (1)
MONO32			X	X		
POINT			X	X		
RGB5551	X	X (2)	X	X		X
RGB565	X	X (2)	X	X		
RGB888	X	X	X	X	X	X
RGB8888	X	X (2)	X	X	X	X
RGB101010	X	X (4)	X	X		
RGB161616	X (3)	X	X	X		
RGBP8	X (2)	X (2)	X	X		
RGBP16	X (3)	X (4)	X	X		
UINT1	X	X	X	X		X
UINT8	X	X	X	X	X	X
UINT16	X (1)	X	X	X		
UINT32			X	X		
UYVY	X (2)	X (2)	X	X		X (2)

YUV			X	X	
YUY2	X [2]	X [2]	X	X	X [2]
YVYU			X	X	
YUYV			X	X	
Y211			X	X	
Y411			X	X	

[1] Buffer data are converted to UINT8 format prior to being saved into file.

[2] Buffer data are converted to RGB888 format prior to being saved into file.

[3] Buffer data are converted to RGB101010 format prior to being saved into file.

[4] Buffer data are converted to RGB161616 format prior to being saved into file.

Graphic Font File Format

This section describes the Graphic font file format used in the Sapera Graphic module. Those files can be created using the DALSA Font Generator utility program (see *Sapera LT User's Manual* for more details). See the **CORGRAPHIC_PRM_FONTNAME** parameter, and the **CorGraphicSetFont**, **CorGraphicText** and **CorGraphicTextEx** functions.

Offset	Size	Description
0	CHAR [11]	Magic String (Must be "DALSA Coreco Font")
11	CHAR [69]	Reserved
80	UINT8	First character index
81	UINT8	Last character index
82	UINT16	First character data size in bytes (NFC)
84	UINT8	First character width in pixels
85	UINT8	First character height in pixels
86	UINT8 * NFC	First character data
.	.	.
.	.	.
.	.	.
.	UINT16	Last character data size in bytes (NLC)
.	UINT8	Last character width in pixels
.	UINT8	Last character height in pixels
.	UINT8 * NLC	Last character data

LUT File Format

This section describes the LUT file format used in the LUT module. See the **CorLutLoad** and **CorLutSave** functions.

Offset	Size	Description
0	UINT32	Lut format
4-8	UINT32	Reserved
12	UINT32	Number of entries
16-40	UINT32	Reserved
44	M	LUT data

where **M** is given by the field Size of LUT in bytes.

DALSA Contact Information

Sales Information

Visit our web site:	http://www.imaging.com/
Email:	mailto:info@dalsa-coreco.com

International/Canada

DALSA
7075 Place Robert-Joncas
Suite 142
St. Laurent, Quebec, Canada
H4M 2Z2

Tel: (514) 333-1301
Fax: (514) 333-1388

USA

DALSA
Building 8, Floor 2
900 Middlesex Turnpike
Billerica, Ma. 01821

Tel: (978) 670-2000
Fax: (978) 670-2010

Technical Support

Technical support form via our web page: Support requests for imaging product installations, Support requests for imaging applications	http://www.imaging.com/support
Or all requests can be submitted via our web site:	http://www.imaging.com/support
For camera support information:	http://www.imaging.com/camsearch
For product literature and driver updates:	http://www.imaging.com/download

When encountering hardware or software problems, please have the following documents included in your support request:

- The DALSA Diagnostic Tool XML report file
- The DALSA Log Viewer .txt file
- The PCI Diagnostic PciDump.txt file
- The DALSA Device Manager BoardInfo.txt file

Note, all these tools are available from the Windows start menu shortcut **Start•Programs•DALSA•Sapera LT•Tools**.

Glossary of Terms

Channel

A channel is a data path from a camera that includes an entire video line.

Chroma

The color portion of the composite NTSC or PAL video signal. Luma is the black-and-white portion of the signal. Often used interchangeably with Chrominance, although this is technically incorrect.

CMYK

A color model in which all colors are composed of varying intensities of the primary subtractive colors: Cyan, Magenta, Yellow, and Black. This color model is often used in print processing.

Color Key

Color keying is a method used to decide the source of a display pixel on the screen. If the graphics pixel on the VGA screen has the pixel value of the color key, then the corresponding pixel in the designated buffer will be displayed; otherwise, the VGA screen's pixel will be displayed.

Complex Parameter

A parameter with a size greater than an UINT32.

Composite Video

A single signal that includes both color video and timing information. NTSC and PAL are composite video standards, while RGB is not.

Contiguous Memory

Memory allocated as a single memory block in physical memory which is not pageable and not moveable.

Decimation

A process whereby pixels are dropped from digitized video waveforms for quick-and-easy image scaling. If 100 pixels are produced by a waveform, but only 50 are stored or used, the video waveform has been decimated by a factor of 2:1.

DLL

Dynamic Link Library. The supplied DLLs form the software interface between a Windows application and the DALSA hardware device.

Element

A data unit within the buffer, which may or may not be a pixel.

Frame Buffer

A large unit of memory used to hold the image for display onscreen.

Grayscale

In image processing, the range of available brightness levels, displayed in shades of gray. In an 8-bit system, the gray scale contains values from 0 to 255.

Host Memory

The Windows system's random-access memory. Typically refers to a frame buffer allocated in the computer system's memory.

Interlaced

The standard television method of raster scanning, in which the image is the product of two fields, each of which is made up of the image's alternate lines (i.e., one field is comprised of lines 1, 3, 5, etc., and the other is comprised of lines 2, 4, 6, etc.).

Keying Color

The Windows color which is used as a switch to the frame buffer video. Wherever the keying color is drawn, it is replaced with video from the buffer.

Lookup Table, LUT

In image processing, the segment of memory that stores values for point processes. Input pixel values are those of the original image, while output values are those altered by the chosen point process. An input lookup table destructively modifies the stored image data, whereas the output lookup table simply receives the stored data and modifies it for output only.

Luma

The black-and-white portion of the composite NTSC or PAL video signal. Chroma is the color portion of the signal. Often used interchangeably with Luminance, although this is technically incorrect.

LVDS

Low Voltage Differential Signaling: A transmission method for sending digital information by using a very low voltage swing differentially over two PCB traces or a balanced cable. LVDS is relatively immune to noise.

Monochrome

A video source with only one component, usually meant to refer to a black-and-white composite signal. A monochrome composite video source has no chroma information.

Noninterlaced

Video scanning method, in which all the lines in the frame are scanned out sequentially. Used in several different analog and digital video systems, including progressive scan analog cameras, digital video cameras and computer monitors.

NTSC

National Television Standards Committee. Color TV standard used in North America, Japan, and in several other jurisdictions. The interlaced video signal is composed of a total of 525 video lines at a frame rate of 30 Hz.

PAL

Phase Alteration by Line. Color TV standard used in most of Europe and in several other jurisdictions. The interlaced video signal is composed of a total of 625 video lines at a frame rate of 25 Hz.

PCI

Peripheral Component Interconnect. The PCI local bus is a 32-bit high performance expansion bus intended for interconnecting add-in boards, controllers, and processor/memory systems.

Pixel

A single picture element, the smallest individual digital video component. The number of pixels describes the number of digital samples taken of the analog video signal. The number of pixels per video line by the number of active video lines describes the acquisition image resolution. The binary size of each pixel (i.e., 8 bits, 15 bits, 24 bits) defines the number of gray levels or colors possible for each pixel.

Raster

The pattern of lines traced by rectilinear scanning in display systems.

RGB

Red, Green, Blue. Commonly used to refer to a non-composite video standard which uses these three colors in combination to generate a color video image.

RS-170

The original United States standard for black and white television. Now commonly used to refer to monochrome analog video signals.

Scaling

The act of changing the effective resolution of an image.

SECAM

Sequentiel Couleur avec Mémoire, a TV standard similar to PAL, in which the chroma is FM modulated and the R'-Y and B'-Y signals are transmitted line sequentially. Used primarily in France and Russia as well as in several other French-speaking and former Warsaw Pact countries.

Simple Parameter

A parameter with a size less than or equal to an UINT32.

Stride

The memory distance between two pixels that are viewed as vertically adjacent in the image.

S-Video

Separate video, also known as Y/C video, which supports separate luma (Y) and chroma (C) video inputs and outputs. Often used interchangeably with S-VHS, which is technically incorrect.

Sync

The basic piece of information which tells a video display (TV or computer monitor) where to put the picture. Horizontal sync, or HSYNC, controls the left-right dimension and vertical sync, or VSYNC controls the top-to-bottom dimension.

Tap

A tap is a data path from a camera that includes a part of a video line. An entire video line from the camera must then be constructed by combining all the taps together.

Tearing

A display artifact caused by the fact that the video display will read frame buffer memory asynchronously to the incoming video stream. Tearing is non-data destructive.

Video Input Conditioning , VIC

The act of modifying an analog video signal via bandwidth filtering or gain amplification.

Y/C

See S-Video.

YUV

A common color space used in composite video color systems. Y is the luma component while U and V are the color difference components.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X