IDT Cameras

SDK Reference Manual

Software Development Kit, 32 and 64 bit

This page is intentionally left blank)

Software Release
2.16.01

Document Revision
December 2020

Products Information
www.idtvision.com

North America
1 West Mountain Street, Suite #3
Pasadena, CA 91103
United States of America
P: Local: (+1) 626-521-5470
P: Toll-Free: (+1) 833-241-6361
F: (+1) (626) 316-7503

Europe
via Pennella, 94
I-38057 - Pergine Valsugana (TN)
Italy
P: (+39) 0461- 510522

Eekhoornstraat, 22
B-3920 - Lommel
Belgium
P: (+32) 11- 551065
F: (+32) 11- 554766

Copyright © Integrated Design Tools, Inc.
The information in this manual is for information purposes only and is subject to change without notice. Integrated Design Tools, Inc. makes no warranty of any kind with regards to the information contained in this manual, including but not limited to implied warranties of merchantability and fitness for a particular purpose. Integrated Design Tools, Inc. shall not be liable for errors contained herein nor for incidental or consequential damages from the furnishing of this information. No part of this manual may be copied, reproduced, recorded, transmitted or translated without the express written permission of Integrated Design Tools, Inc.

Table of Contents

1. Overview

The on-line documentation of the Software Development Kit and its components is divided into the following parts:

Using the SDK

This section describes how to start using the Software Development Kit.

SDK Reference

This section contains a detailed description of the SDK functions.

LabVIEW™ Interface Reference

This section contains a detailed description of the camera LabVIEW™ VIs.

MATLAB™ Interface Reference

This section contains a detailed description of the camera MATLAB™ Drivers.

RAW Reader Library

This section contains a detailed description of the RAW Reader library.

Appendix

This section provides additional information about data structures, parameters, error codes and return codes.

1.1. Directories structure

The default installation directory of the SDK is

"**C:/Program Files (x86)/IDT/CameraSDK V1.V2.V3**"

where v1.v2.v3 is the current SDK version. The directory contains a set of sub-directories:

BIN/Win32: it contains 32-bit binary files (drivers, INF, DLLs) that can be re-distributed with the camera and 32-bit applications.

BIN/x64: it contains 64-bit binary files (drivers, INF, DLLs) that can be re-distributed with the camera and 64-bit applications.

BIN\x64\DriverPCI: it contains setup files for the installation of the PCI drivers for X-Stream PCIe cameras (1440p and 720p).

BIN\x64\DriverUSB: it contains setup files for the installation of the USB 2.0 drivers for X-series and Y-series cameras, and for the MotionPro Timing Hub.

BIN\x64\DriverXSRT: it contains setup files for the installation of the Xstream-RT drivers for XS-Mini cameras.

DOCS: it contains the SDK documentation and the camera specifications manual.

INCLUDE: it contains the SDK header files (H, VB, BAS, C#, PY), helpers and misc header files.

LabVIEW: it contains the LabVIEW™ examples (VIs) and a copy of the drivers for manual install (manual_install.pdf).

LIB: it contains the SDK lib files (32 and 64-bit) for dynamic linking.

MATLAB/x64: it contains the 64-bit MATLAB™ drivers and examples (32 bit version is not supported anymore).

MCfgFiles: it contains the M-series camera configuration files for Sapera, National Instruments and Bitflow frame grabbers.

SOURCE/VC: it contains the Visual C++ examples (MSVC 2008, 2010, 2017).

SOURCE/VC#: it contains Visual C# examples (MSVS 2017).

1.2. Supported cameras

The Software development Kit supports the following camera models:

- N, NR, NX-series, R-series, O-series and Os-series (GE).
- CC cameras (Crash Cams), CC-Mini and CC-Stick (GE).
- X-Stream PCIe cameras (720p and 1440p), XS-Mini and XS-Stick (Thunderbolt 3/PCI).
- Y-series (USB + GE).
- M-series (Camera Link with frame grabbers support).
- Legacy X-Stream XS (USB).
- Legacy MotionPro HS (USB) and MotionPro X (USB + GE).
- Redlake cameras (HG-100K, HG-LE, HG-XR, HG-TH, HG-CH, HG-2000, CR-2000, HG-TX (GE only).

The 32-bit and 64-bit versions of the SDK have some differences. Some of the cameras are not fully supported as reported in the table below.

Camera	Link	Win32	x64
N/NR/NX/O/Os/CC/CCM/R	Giga-Ethernet	Yes	Yes
X-Stream PCIe	PCIe 2.0 and 3.0	**No**	Yes
XS Mini, XS-Stick	Thunderbolt 3/PCIe	**No**	Yes
MotionPro Y	USB 2.0, Giga-Ethernet	Yes	Yes
MotionScope M	Camera-Link	Yes	Yes
X-Stream XS, MotionPro HS/X	USB 2.0	Yes	Yes
MotionPro X	Giga-Ethernet	Yes	**No**
Redlake cameras	Giga-Ethernet	Yes	Yes

1.3. Redistributable Files

This section outlines the options available to third-party vendors for distributing camera drivers for Windows XP/Vista/7/8 and 10. The files that can be redistributed are in the BIN/Wi32 and BIN/x64 sub-directory of the installation directory (C:\Program Files\IDT\CameraSDK v1.v2.v3).

USB 2.0 drivers for X and y cameras (DriverUSB).

File	Windows XP/Vista/7/8/10
Xstream.inf	C:\WINDOWS\INF
Xsusbdrv.sys	C:\WINDOWS\SYSTEM32\DRIVERS

X-Stream PCIe camera drivers (Bin/x64/DriverPCI).

File	Description
PciDriver.inf	INF file
pciDriver.sys	Kernel driver
pciDriver.cat	Catalog file
WdfCoinstaller01009.dll	Support DLL

Dynamic linking libraries (the files may be copied to any directory that can be accessed by the third-party software).

File	Description
XStreamDrv.dll	SDK main interface driver
ImageFmts.dll	Support for SDK main driver
GraphMlb.dll	Support for Y/HG cameras and JPEG encoding
XsPortUSB.dll, XsPortGE.dll	USB 2.0 and GE Port drivers for HS/X
CyCamVideoRecorder.dll, CyDispExLib.dll, CyMediumLib.dll, CyEngineLib.dll, CyCamLib.dll, CyImgLib.dll, CyComLib.dll, CyUtilsLib.dll	Support for MotionPro X GE Port Driver (32-bit only)
XsPortYUSB.dll, XsPortYGE.dll	USB 2.0 and GE Port drivers for Y
XsPortNGE.dll	GE Port driver for N/NR/NX/Os/CC
XsPortCL.dll, XsPortNI.dll, XsPortBF.dll XsPortEP.dll	Camera-Link Port drivers for M (frame grabber drivers required)
XsPortPCIX.dll	PCIe driver for X-Stream
XsPortHG.dll	HG cameras Port driver
XsPortRL.dll	Port driver for Redlake Legacy cameras
XsPortRAW.dll	Port Driver for Virtual camera (RAW files)

LibH264dec.dll	Library for decoding the H264 frames
Openh264.dll	Library for deconding the H264 frames

The table below lists the camera files for M-Series and National Instruments IMAQ. The files should be copied to the NI-IMAQ Data folder.

File	Description
“IDT MotionScope M3.icd”	M3 camera configuration file for NI-IMAQ
“IDT MotionScope M5.icd”	M5 camera configuration file for NI-IMAQ

The table below lists the camera files for M-Series and Sapera LT Library (Dalsa-Coreco frame grabber). The files should be copied to the Windows System32 folder.

File	Description
M3.ccf	M3 camera configuration file for Sapera
M5.ccf	M5 camera configuration file for Sapera

The table below lists the camera files for M-Series and Bitflow SysReg (Bitflow frame grabber).

File	Description
IDT-M3-FreeRun.r64	M3 camera configuration file for Bitflow
IDT-M5-FreeRun.r64	M5 camera configuration file for Bitflow

The setup files in the directories **Bin\x64\DriverPCI** and **Bin\x64\DriverUSB** can be distributed with the cameras.

1.4. Camera calibration file distribution (XS, XS-Stick, PCIe, M)

Some IDT camera models (XSM, XS-Stick, M-series) require that the calibration file is stored on the local hard disk. The default location of calibration files directory is

"COMMON_DOCUMENTS"/IDT/CameraFiles

The value of "COMMON_DOCUMENTS" depends on the operating system. The name is also stored in the "IDTCOMMON" system variable that is created when the SDK is installed.

The directory name may be also changed as below:

Create the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\IDT\MotionProX

Then create the following string value:

CalibrationFileDirectory

In the string value write the full path to the directory where the calibration file is stored.

Os, NR, NX and latest Y and N cameras store the calibration file in the internal storage area (flash memory) and do not require that the calibration file is locally stored.

HG cameras do not require any calibration file.

2. Using the SDK

2.1. Overview

2.1.1. Programming Languages

A C/C++ header file is included in the SDK (**XStrmAPI.h** file in the Include sub-directory).

Most compiled languages can call functions; you will need to write your own header/import/unit equivalent based on the C header file.

Visual Basic modules are included in the SDK (**XStrmAPI.bas** file in the Include sub-directory and **XstrmAPI.vb** for VB.NET and later). VB cannot use **XsQueueOneFrame** or **XsQueueCameraSettings** or related functions, because these functions have callbacks which occur on a different thread. If you want to use VB, you might need to write some C code depending on your application's requirements. The same issue with asynchronous callbacks, above, also applies to Java.

The Windows driver is a DLL (XStreamDrv.dll) that resides in Bin/Win32 directory. The 64 bit version may be found in the Bin/x64 directory.

MS Visual C++™: A Visual C++ 6.0 stub COFF library is provided (**XStreamDrv.lib** or **XStremDrv64.lib** in the Lib sub-directory); if you are programming with Visual C++, link your application to XStreamDrv.lib. The DLL uses Windows standard calling conventions (_stdcall).

Borland C++ Builder™: the XStreamDrv.lib file is in COFF format. Borland C++ Builder requires the OMF format. To convert the library into to OMF format, run the IMPLIB Borland tool with the following syntax: “IMPLIB XStreamDrv.lib XStreamDrv.dll”.

Other compilers: the Most other compilers can create a stub library for DLLs. The DLL uses Windows standard calling conventions (_stdcall).

MS Visual C#: the **XsCamera.cs** file has been added to the include folder (for 32 and 64 bit). It wraps the APIs into a C# class.

Python: the XsCamera.py file has been added to the Include folder. It wraps the API into a python interface.

2.1.2. 64 Bit Programming

The Visual C++ stub COFF library for 64-bit programming is the **XStreamDrv64.lib** file, stored in the LIB sub-directory of the SDK.

One of the main issues in migrating software from 32 bit to 64 bit platforms is the size of types.

An “int” and a “long” are 32-bit values in on 64-bit Windows operating systems. For programs that you plan to compile for 64-bit platforms, you should be careful not to assign pointers to 32-bit variables. Pointers are 64-bit on 64-bit platforms, and you will truncate the pointer value if you assign it to a 32-bit variable.

For this reason, the parameters of the **XsPreConfigCamera** routine have been converted into “void pointers”. In some conditions, the parameters of the routine are pointers to char buffers, and then in a 64-bit environment a 32 bit “long” parameter is not enough.

2.1.3. MAC OSX Programming

When you install the SDK on MAC OSX the framework “XstreamDrv.framework” is installed in /Libraries/Frameworks. It contains the API header file (Headers/XstrmAPI.h) and the libraries that you need to integrate the IDT cameras in your project.

X-Code: to include the IDT cameras SDK framework into your project, , choose Project > Add to Project and select the framework directory. Alternatively, you can control-click your project group and choose Add Files > Existing Frameworks from the contextual menu. When you add an existing framework to your project, Xcode asks you to associate it with one or more targets in your project. Once associated, Xcode automatically links the framework against the resulting executable.

QT-creator: to include the IDT Cameras SDK framework to your QT project, add the following line to your .pro file.

LIBS += /Libraries/Frameworks/XStreamDrv.framework/XStreamDrv

2.1.4. Types

Some types have been defined in the SDK to allow the use of the file XStrmAPI.h in different platforms. See the table below**:**

Type	MS C++ compiler	Other Compilers
XSULONG32 (32 bit unsigned integer)	unsigned long	unsigned int
XSLONG32 (32 bit signed integer)	long	int
XSUINT64 (64 bit unsigned integer)	unsigned __int64	unsigned long long
XSINT64 (64 bit signed integer)	__int64	long long

2.1.5. Example

A simple program would use the following sequence of routine calls to capture images from a camera. The routines and the parameters used in this example are explained in the following topics.

```
#include "XstrmAPI.h"
#include <stdio.h>
#include <time.h>
#include <malloc.h>
#include <memory.h>

int main(int argc, char* argv[])
{
   XS_ENUMITEM xsl[10];
   XSULONG32 nEnumFlt, nListLen = sizeof(xsl)/sizeof(XS_ENUMITEM);
   XSULONG32 nSnsType, nW, nH, nPD, nValHi, nImgSize;
   XS_HANDLE hCamera;
   XS_SETTINGS xsCfg;
   unsigned char *pBuf;

   // Load the driver
   XsLoadDriver(0);

   // find Y, N or Os cameras in the GE network
   nEnumFlt = XS_EF_GE_Y|XS_EF_GE_N;

   // nListLen is the length of your XS_ENUMITEM array
   XsEnumCameras( &xsl[0], &nListLen, nEnumFlt );
   // nListLen is now the number of cameras available.
   // It may be larger than your XS_ENUMITEM array length!
   if ( nListLen==0 || xsl[0].bIsOpen==1 ) return 0;

   // Open the first camera in the list.
   XsOpenCamera( xsl[0].nCameraId, &hCamera );

   // read the configuration
   xsCfg.cbSize = sizeof(XS_SETTINGS);  // Don't forget this!!!
   XsReadCameraSettings( hCamera, &xsCfg );
   // set exposure to 1 ms and the fps to 100 (T = 10 ms)
   XsSetParameter( hCamera, &xsCfg, XSP_EXPOSURE, 1000000 );
   XsSetParameter( hCamera, &xsCfg, XSP_PERIOD, 10000000 );
   // Get the info about image and set pixel depth
   XsGetCameraInfo( hCamera, XSI_SNS_TYPE, &nSnsType, &nValHi);
   XsGetParameter( hCamera, &xsCfg, XSP_ROIWIDTH, &nW);
   XsGetParameter( hCamera, &xsCfg, XSP_ROIHEIGHT, &nH);
   if (nSnsType==XS_ST_COLOR) nPD = 24;
   else nPD = 8;
   XsSetParameter( hCamera, &xsCfg, XSP_PIX_DEPTH, nPD );
   // set the rec mode to normal
   XsSetParameter( hCamera, &xsCfg, XSP_REC_MODE, XS_RM_NORMAL );

   // Send settings to the camera
   XsRefreshCameraSettings( hCamera, &xsCfg );

   // Allocate image memory
   nImgSize = nW*nH*nPD/8;
   pBuf = (unsigned char *)malloc(nImgSize);
   memset(pBuf, 0, nImgSize);
```

```
    XSULONG32 nAddLo, nAddHi, nBusy, nSts;
    int i;

    // read the live offset and use it as start address
    XsGetCameraInfo(hCamera,XSI_LIVE_BUF_SIZE,&nAddLo,&nAddHi);
    // start the acquisition (do not install any callback)
    XsMemoryStartGrab(hCamera,nAddLo,nAddHi,100,0,NULL,0,0);

    // poll the status until the recording is OK (time out = 3 sec)
    time_t now, cur;
    time(&now);
    cur = now;
    while( difftime(cur,now)<3. )
    {
        nBusy = nSts = 0;
        XsGetCameraStatus(hCamera,&nBusy,&nSts,0,0,0,0);
        if(nBusy==0 &&
            nSts!=XSST_REC_PRETRG && nSts!=XSST_REC_POSTRG)
            break;
        // calculate time
        time(&cur);
    }
    // read 10 frames
    for (i=0;i<10;i++)
    {
        XsMemoryReadFrame(hCamera, nAddLo, nAddHi, i, pBuf);
    }
    // free the memory
    free( pBuf );
    // Close the camera
    XsCloseCamera( hCamera );
    // Unload the driver
    XsUnloadDriver();

    return 0;
}
```

2.2. Detect a camera and open it

2.2.1. Load/Unload the driver

The first call into the driver must be **XsLoadDriver**. Call **XsUnloadDriver** when you are finished.

If the parameter ***nUSBNotify*** is set to 0, the driver will not notify any disconnection of the USB cable. If the parameter is set to 1, the notification is enabled.

2.2.2. Enumerate/Open a camera

To get the list of available cameras, call **XsEnumCameras**. Use the *nCameraId* field of the camera list in your call to **XsOpenCamera**. Below a simple example of opening the first available Y, N or O Ethernet camera.

```
XS_ENUMITEM xsl[10];
XSULONG32 nEnumFlt, ListLen = sizeof(xsl)/sizeof(XS_ENUMITEM);
XSULONG32

// Load the driver
XsLoadDriver(0);

// find Y, N or Os cameras in the GE network
nEnumFlt = XS_EF_GE_Y|XS_EF_GE_NO;

// nListLen is the length of your XS_ENUMITEM array
XsEnumCameras( &xsl[0], &nListLen, nEnumFlt );

// nListLen is now the number of cameras available.
// It may be larger than your XS_ENUMITEM array length!
If ( nListLen>0 && xsl[0].bIsOpen==FALSE )
{
   XS_HANDLE hCamera;
   // Open the first camera in the list.
   XsOpenCamera( xsl[0].nCameraId, &hCamera );
   // Do something...
   ...
   // Close the camera.
   XsCloseCamera( hCamera );
}

// Unload the driver
XsUnloadDriver();
```

The camera list contains a unique ID which identifies the camera. The ID must be used to open the camera and retrieve the camera handle.

Then the camera handle must be used to call any other routine of the SDK.

2.2.3. Camera pre-configuration

Some camera or system may be configured before opening a camera. Some of those “pre-configuration” parameters are system-specific, some are camera-specific. The routine is **XsPreConfigCamera**.

System parameters

- **XSPP_DB_FOLDER**: it specifies the database folder used to enumerate “virtual” raw file cameras. The Camera ID field is ignored.
- **XSPP_NET_AD_IP**: the IP address of the computer network adapter connected to the cameras network. The driver uses the IP address to send the enumeration command only to the selected network. If the value is set to default (0xFFFFFFFF) the driver sends the enumeration command to any network adapter installed on the local computer. If the IP address of the adapter connected to the cameras network is known, it should be used to configure the XSPP_NET_AD_IP and avoid to send the enumeration command to the wrong network. The Camera ID field is ignored.
- **XSPP_DISABLE_1024**: the parameter is valid for each camera. If it’s enabled, Y4, N4, NX4 and Os4 cameras maximum resolution is set to 1016x1016. The Camera ID field is ignored.
- **XSPP_NET_ADD_CMD_PORT**: deprecated.

Camera parameters

- **XSPP_IP_ADDRESS**: the camera IP address. Call the routine with this parameter if you want to modify the camera IP address before opening the camera. The Camera ID parameter can be retrieved in the enumeration procedure.
- **XSPP_CAM_DFL_GW**: it specifies the camera default gateway. The value may be changed if two sub network are involved.
- **XSPP_GET_IP_ADDRESS**: this value is used to read the IP address from HG cameras. It is useful when the camera is connected through the DCU port and the user wants to know the actual camera IP address. The Camera ID value is retrieved in the enumeration procedure.
- **XSPP_PCIX_DMASIZE**: the value configures the size of the DMA buffer used by the camera to acquire images in the computer memory. The value units are MB.
- **XSPP_IP_ADD_EX**: deprecated.
- **XSPP_CAM_CMD_PORT**: deprecated.

The table below shows the meaning of the nValueLo and nValueHi for each value of nParamKey**.**

XS_PRE_PARAM	nCameraID	nValueLo	nValueHi
XSPP_IP_ADDRESS	Camera ID	unsigned int	unsigned int
XSPP_NET_AD_IP	Not used	unsigned int	Not used
XSPP_IP_ADD_EX	Camera MAC add	unsigned int	Not used
XSPP_CAM_CMD_PORT	Camera ID	unsigned int	Not used
XSPP_NET_AD_CMD_PORT	Not used	unsigned int	Not used

XSPP_GET_IP_ADDRESS	Camera IP add	unsigned int*	unsigned int*
XSPP_DB_FOLDER	Not used	char*	Not used
XSPP_CAM_DFL_GW	Camera ID	unsigned int	Not used
XSPP_DISABLE_1024	Not used	unsigned int	Not used
XSPP_REBOOT_FW	Camera ID	Not used	Not used
XSPP_PCIX_DMASIZE	Not used	unsigned int	Not used

The code below shows how to configure the driver to use a specific network adapter.

```
#define IP_SET(a1,a2,a3,a4) ((a1<<24)+(a2<<16)+(a3<<8)+(a4))

XSULONG32 nIPAdd;

// Load the driver
XsLoadDriver(0);

// we assume that we have detected the network adapter IP
// address and it's value is 192.168.0.2
nIPAdd = IP_SET(192,168,0,2);

// configure the driver to set the network adapter by its
// IP address
// convert the parameter into (void*)
XsPreConfigCamera(0,0,XSPP_NET_AD_IP,(void*)nIPAdd,0);

// Unload the driver
XsUnloadDriver();
```

The camera IP address must be compatible with the IP address of the computer network adapter connected to the camera network.

Example: if the network adapter IP address is 192.168.0.2 and the sub-net mask is 255.255.255.0, then the cameras IP addresses should be 192.168.0.N (where N may be from 3 to 253). The formula to detect is a camera IP address is compatible with the network adapter IP address is below.

(NetAdpIPAdd & NetAdpSNMask) = (CameraIPAdd & NetAdpSNMask)

The code below shows how to configure the IP address of a camera with an address that is is compatible with the network adapter's IP address.

```
#define IP_SET(a1,a2,a3,a4) ((a1<<24)+(a2<<16)+(a3<<8)+(a4))

XS_ENUMITEM xsl[10];
XSULONG32 nEnumFlt, ListLen = sizeof(xsl)/sizeof(XS_ENUMITEM);
XSULONG32 nIPAdd, nSNMask;
// Load the driver
XsLoadDriver(0);
// find Y, N or Os cameras in the GE network
nEnumFlt = XS_EF_GE_Y|XS_EF_GE_N;
XsEnumCameras( &xsl[0], &nListLen, nEnumFlt );
// set the IP address of first enumerated camera
nIPAdd = IP_SET(192,168,0,100);
nSNMask = IP_SET(255,255,255,0);
// convert the parameter into (void*)
XsPreConfigCamera((void*)xls[0].nCameraId,XSPP_IP_ADDRESS,
                  (void*)nIpAdd,(void*)nSNMask);;
// Unload the driver
XsUnloadDriver();
```

2.2.4. Camera speed grades

The camera speed grade is reported in the "**nSubModel**" field of the XS_ENUM structure. The table below shows the existing values of speed grade for the IDT camera models.

Legacy cameras

Camera	Model	Sub-Model	Notes
X-Stream cameras	all	0	Not used
HS cameras	all	0	Not used
X cameras	all	0	Not used
M3/M5	XS_CM_MP_M3/M5	0	Not used
HG legacy	all	0	Not used
HG-100K	XS_CM_HG_100K	0	1504x1128
HG-XR	XS_CM_HG_100K	1	1504x1128
HG-XL	XS_CM_HG_100K	2	1504x1128
HG-LE	XS_CM_HG_LE	0	752x1128
HG-TH	XS_CM_HG_TH	0	752x564
HG-CH	XS_CM_HG_TH	1	752x564

Y cameras

Camera	Model	Sub-Model	Resolution/Notes
Y3-classic	XS_CM_MP_Y3	0	1280x1024
Y3-S1	XS_CM_MP_Y3	1	1280x1024
Y3-S2	XS_CM_MP_Y3	2	1280x1024
Y3-HD	XS_CM_MP_Y3	3	1920x1080 (discontinued)
Y4-S1	XS_CM_MP_Y4	1	1024x1024
Y4-S2	XS_CM_MP_Y4	2	1024x1024
Y4-S3	XS_CM_MP_Y4	3	1024x1024
Y5	XS_CM_MP_Y5	0	2336x1728
Y5-HD	XS_CM_MP_Y5	1	H-Diablo (discontinued)
Y6	XS_CM_MP_Y6	0	1504x1128
Y7-S1	XS_CM_MP_Y7	1	1920x1080
Y7-S2	XS_CM_MP_Y7	2	1920x1080
Y7-S3	XS_CM_MP_Y7	3	1920x1080
Y8-S1	XS_CM_MP_Y8	1	1600x1200
Y8-S2	XS_CM_MP_Y8	2	1600x1200
Y8-S3	XS_CM_MP_Y8	3	1600x1200

N/NR/NX cameras

Camera	Model	Sub-Model	Notes
N3	XS_CM_MP_N3	0	Old N3 model (1280x1024)
N3-S1	XS_CM_MP_N3	1	1280x1024
N3-S2	XS_CM_MP_N3	2	1280x1024
N3-S3	XS_CM_MP_N3	3	1280x1024
N3-S4	XS_CM_MP_N3	4	1280x1024
N4	XS_CM_MP_N4	0	Old N4 model (1016x1016)
N4-S1	XS_CM_MP_N4	1	1024x1024
N4-S2	XS_CM_MP_N4	2	1024x1024
N4-S3	XS_CM_MP_N4	3	1024x1024
N5-S1	XS_CM_MP_N5	1	2336x1728
N5-S2	XS_CM_MP_N5	2	2336x1728
N7-S1	XS_CM_MP_Y7	1	1920x1080
N7-S2	XS_CM_MP_Y7	2	1920x1080
N8-S1	XS_CM_MP_Y8	1	1600x1200
N8-S2	XS_CM_MP_Y8	2	1600x1200

Os cameras

Camera	Model	Sub-Model	Notes
Os4-S1	XS_CM_MP_O4	1	1024x1024
Os5-4K	XS_CM_MP_O5	1	3840x2160
Os7-S1	XS_CM_MP_O	1	1920x1296
Os7-S2	XS_CM_MP_O	2	1920x1296
Os7-S3	XS_CM_MP_O	3	1920x1296
Os8-S1	XS_CM_MP_O	1	1600x1200
Os8-S2	XS_CM_MP_O	2	1600x1200
Os8-S3	XS_CM_MP_O	3	1600x1200
Os8-S4	XS_CM_MP_O	4	1600x1200
Os10-4K	XS_CM_MP_O	0	3840x2400

CrashCam and CC-Mini cameras

Camera	Model	Sub-Model	Notes
CC1060	XS_CM_CC_1060	0	1024x1024
CC1520	XS_CM_CC_1520	0	1440x1024
CC1540	XS_CM_CC_1540	0	1440x1024
CC4010	XS_CM_CC_4010	0	2560x1600

CrashCam mini cameras

Camera	Model	Sub-Model	Notes
CCM1510	XS_CM_CC_M1510	0	1440x1024
CCM1520	XS_CM_CC_M1520	0	1440x1024
CCM3510	XS_CM_CC_M3510	0	2560x1440
CCM-3525	XS_CM_CC_M3525	0	2560x1440
CC-Stick	XS_CM_CC_STICK	0	1920x1536
CCM-5K05	XS_CM_CC_M5K05	0	5120x2880

R-series

Camera	Model	Sub-Model	Notes
R-2K	XS_CM_R_2K	0	2048x1088

PCIe/XS-Mini

Camera	Model	Sub-Model	Notes
PCIe 720p	XS_CM_PCIE_X7	0	1280x720
PCIe 1440p	XS_CM_PCIE_X14	0	2560x1440
XSM-1540	XS_CM_XSM_1540	0	1440x1024
XSM-3520	XS_CM_XSM_3520	0	2560x1440
XSM-4KV	XS_CM_XSM_4KV	0	3840x2880
XSM-5K	XS_CM_XSM_5K	0	5120x2880
XS-Stick	XS_CM_XSM_STICK	0	1920x1536

2.2.5. Camera misc capabilities

The camera capabilities are reported in the “**nMiscCaps**” field of the XS_ENUM structure. table below shows the description of the misc capabilities bits.

Field	Value	Description
XS_CAP_NR	0x00000001	The camera is an NR
XS_CAP_NX	0x00000002	The camera is an NX
XS_CAP_NXT	0x00000004	The camera is an NXtra
XS_CAP_NXA	0x00000008	The camera is an NX-Air
XS_CAP_DNR2	0x00000010	The camera supports DNR
XS_CAP_HWBROC	0x00000020	The camera supports hardware BROC
XS_CAP_JPEG	0x00000040	The camera supports JPEG encoding
XS_CAP_1PPS	0x00000080	The camera supports 1PPS protocol
XS_CAP_BATSTS	0x00000100	The camera has battery status (NX-Air only)
XS_CAP_FBCAM	0x00000200	The camera is an FB camera (custom project)
XS_CAP_PIV	0x00000400	The camera has a PIV module (Y only)
XS_CAP_OS	0x00000800	The camera is an Os (not O)
XS_CAP_GPSMOD	0x00001000	The camera has an internal GPS module
XS_CAP_INX	0x00002000	The camera is an iNdustrial
XS_CAP_JPLROC	0x00004000	The Camera is a JPL ROC model
XS_CAP_PTP	0x00008000	The camera supports the PTP protocol
XS_CAP_IS1024	0x00010000	If this bit is 1, the camera is an N/NR or NX and supports the 1024x1024 resolution, if it's 0 the maximum resolution of the camera is 1016x1016.
XS_CAP_OS3	0x00020000	The camera is an OS Version 3 (not used)
XS_CAP_OSA	0x00040000	The camera is an OS Version 3 Airborne
XS_CAP_PLL	0x00080000	The camera supports Phase Lock Loop modes
XS_CAP_IRIGMD	0x00100000	The camera has an internal IRIG module
XS_CAP_SDI_FW	0x00200000	The camera has a special firmware for SDI output
XS_CAP_OSTRM	0x00400000	The camera is OStreaming

2.3. Camera configuration

The camera configuration is stored in the opaque **XS_SETTINGS** structure. The structure is used to read and write parameters from/to the camera.

2.3.1. Read/Write the camera configuration

Before any other operation, the user should fill the XS_SETTINGS structure with valid data and synchronize the structure with the camera.

The parameters are written to the structure through the **XsSetParameter** routine and read through the **XsGetParameter** routine. The function **XsGetParameterAttribute** provides information on a parameter's range and whether the parameter is read-only or not.

Read the default configuration: the example below shows how to read the default configuration, change the exposure and send the configuration to the camera.

```
XS_SETTINGS xsCfg;
xsCfg.cbSize = sizeof(XS_SETTINGS);     // Don't forget this!

// Read default settings from the camera.
XsReadDefaultSettings( hCamera, &xsCfg );

// Change xsCfg: set exposure to 1 ms.
XsSetParameter( hCamera, &xsCfg, XSP_EXPOSURE, 1000000 );

// Send settings to the camera
XsRefreshCameraSettings( hCamera, &xsCfg );
```

Read the camera current configuration: the example below shows how to read the default configuration, change the exposure and send the configuration to the camera.

```
XS_SETTINGS xsCfg;
xsCfg.cbSize = sizeof(XS_SETTINGS);     // Don't forget this!

// Read current settings from the camera.
XsReadCameraSettings( hCamera, &xsCfg );

// Change xsCfg: set exposure to 1 ms.
XsSetParameter( hCamera, &xsCfg, XSP_EXPOSURE, 1000000 );

// Send settings to the camera
XsRefreshCameraSettings( hCamera, &xsCfg );
```

Not all parameters are supported by all cameras. When you query or set a parameter (or get the parameter maximum/minimum) and that parameter is not supported, the error code *XS_E_NOT_SUPPORTED* is returned.

Validate a configuration: the The XS_SETTINGS structure may be validated with a call to the **XsValidateCameraSettings** routine.

2.3.2. Read/Write in camera flash memory

Giga-Ethernet cameras are supplied with a 256/512 MB flash memory. The flash memory contains important information, such as the calibration file and the camera settings. A portion of the flash memory may be used as storage area for user data. For that purpose, two routines have been added to the program interface.

XsReadUserDataFromFlash: reads a buffer of user data from the flash memory user area.

XsWriteUserDataToFlash: writes a buffer of user data to the flash memory user area.

The user must supply a unique ID to both the routines. The ID is a “signature” that let the driver identify the user data block.

NOTE: M-Series cameras, MotionPro HS cameras and X-Stream XS cameras do not have the on-board flash memory module.

2.4. Camera parameters

2.4.1. Frame rate and exposure

Exposure and frame rate are the first parameters that should be configured before recording. The frame rate is configured through its inverse (the acquisition period).

Parameter	Description
XSP_EXPOSURE	The exposure or integration time in nanoseconds
XSP_PERIOD	The inverse of the frame rate in nanoseconds (frame period)

The formulas for converting the frame rate into the period is shown below

nFps = (int)(1000000000./(double)nPeriodNS + 0.5);

The formula below shows how to convert the frame rate into nanoseconds

nPeriodNS = (int)(1000000000./(double)nFps + 0.5);

2.4.2. Pixel depth

The pixel depth parameter controls the format of the image (**XSP_PIX_DEPTH**). See the table below:

Value	Image format
8	8 bit monochrome/Bayer (1 byte per pixel)
9,10,11,12	16 bit monochrome (2 bytes per pixel)
24	24 bit color (3 bytes per pixel)
27,30,33,36	48 bit color (6 bytes per pixel)

The pixel depth may be also set with the XSP_IMG_FORMAT parameter (see below)

XSP_IMG_FMT	Pixel Depth
XS_IF_GRAY8, XS_IF_BAYER8	8
XS_IF_GRAY16, XS_IF_BAYER16	9, 10, 11, 12
XS_IF_BGR24	24
XS_IF_BGR48	27, 30, 33, 36

2.4.3. Image quality

Image quality improvement is achieved with the configuration of some of the camera parameters. A list of those parameters is shown below:

CFA (Color Filter Array) interpolation (deprecated)

The parameters XSP_CI_MODE (color interpolation mode) and XSP_CI_THR (color interpolation threshold) control how the Bayer data is converted into the RGB space. If the mode is set to XS_CIM_BILINEAR the threshold is ignored. If the mode is set to XS_CIM_ADVANCED the threshold controls the sharpness of the conversion (a value of 0 corresponds to a very sharp image with possible noise known as "worm" effect, while a value of 255 corresponds to a softer image similar to the bi-linear algorithm.

Optimal values: XSP_CI_MODE = XS_CIM_ADVANCED, XSP_CI_THR = 64.

TNK (Temporal Noise Killer)

The parameter **XSP_DYNAMIC_NR** controls the time-dependent noise reduction filter. Each pixel value is compared with the same pixel value in images acquired before and after and the result is used to eliminate the noise the component of noise that is not a fixed pattern. The parameter is not supported on all cameras (see the XSI_TNR_SUPPORT) The value of this parameter should be set to 24 and never changed.

Optimal value: XSP_DYNAMIC_NR = 24

DNR (Dynamic Noise Reduction)

The parameter **XSP_DYNAMIC_NR2** controls the space noise reduction filter. Each pixel is compared to a set of surrounding pixels in the same image and used to reduce noise. The result is a better uniformity in flat parts of the image. The parameter is not supported on all cameras (see the XSI_DNR2_SUPPORT) The value of this parameter should be set to 3 and never changed.

Optimal value: XSP_DYNAMIC_NR2 = 3

Sharpening

The parameter **XSP_SHARPEN** (sharpening value) controls the overall strength of the sharpening effect and the parameter **XSP_SHARPEN_THR** (sharpening threshold) controls the minimum brightness change that will be sharpened. This can be used to sharpen more pronounced edges, while leaving more subtle edges untouched. It's especially useful to avoid sharpening noise.

Optimal values: XSP_SHARPEN = 2 XSP_SHARPEN_THR = 25

Gaussian (Anti-alias, Blur) Filter

The parameter **XSP_GAUSS_FLT** controls the strength of the smoothing effect on the image. The effect is a reduction of image noise and a reduction of details due to the blurring.

Optimal value: XSP_GAUSS_FLT = 0

Saturation (XSP_SATURATION)

This parameter is applied to color images only. It affects the color saturation. The range is from 0 to 20. The default value is 10. A value of 0 converts the color image into gray scale. An optimal value for this parameter is 12.

Other image quality parameters are shown in the table below.

Parameter	Range	Default	Notes
XSP_GAIN	0 to 2	0	Actual gain values are 1.00, 1.41, 2.00. Some old cameras supports value 3, corresponding to gain 2.82.
XSP_GAMMA	1 to 40	10	Default corresponds to unity gamma
XSP_BRIGHTNESS	0 to 50	25	Default corresponds to unity brightness
XSP_CONTRAST	0 o 20	10	Default corresponds to unity contrast
XSP_HUE	0 to 360	180	Applied to color images. The range is from -180 to 180 (degrees).

Important note: recent cameras are shipped with values of default image parameter that have been optimized during the calibration process. The code below shows how to retrieve those values. Once read, the values should be stored and used.

```
XS_SETTINGS xsCfg;
XSULONG32 nTNK, nDNR, nGaus, nSharp, nSharpTH, nSat;

// Read default settings from the camera.
xsCfg.cbSize = sizeof(XS_SETTINGS);     // Don't forget this!
XsReadDefaultSettings( hCamera, &xsCfg );

// read and save default optimal values
XsGetParameter( hCamera, &xsCfg, XSP_DYNAMIC_NR, &nTNK );
XsGetParameter( hCamera, &xsCfg, XSP_DYNAMIC_NR2, &nDNR );
XsGetParameter( hCamera, &xsCfg, XSP_SHARPEN, &nSharp );
XsGetParameter( hCamera, &xsCfg, XSP_SHARPEN_THR, &nSharpTH );
XsGetParameter( hCamera, &xsCfg, XSP_GAUSS_FLT, &nGauss );
XsGetParameter( hCamera, &xsCfg, XSP_SATURATION, &nSat );
```

2.4.4. White Balance / Color Balance

A 9x9 matrix may be applied to RGB space to correct the colors. The array values are in 16.16 format (16 bits for integer, 16 bits for decimal).

The formulas for the conversion from double into 16.16 integer are shown below:

nIntValue = (unsigned int)(dDblValue * 65536.);

dDblValue = ((double)nIntValue) / 65536.;

The parameters involved in color balance are XSP_WB_11 to XSP_WB_33 (see below).

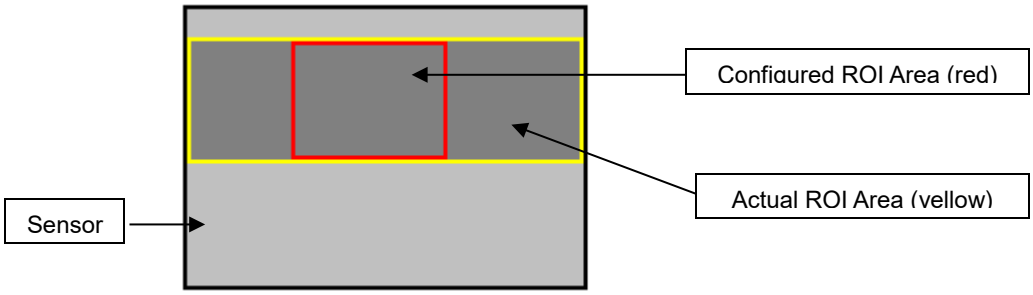
$$\begin{bmatrix} B_{out} \\ G_{out} \\ R_{out} \end{bmatrix} = \begin{bmatrix} \text{XSP\_WB\_11} & \text{XSP\_WB\_12} & \text{XSP\_WB\_13} \\ \text{XSP\_WB\_21} & \text{XSP\_WB\_22} & \text{XSP\_WB\_23} \\ \text{XSP\_WB\_31} & \text{XSP\_WB\_32} & \text{XSP\_WB\_33} \end{bmatrix} X \begin{bmatrix} B_{in} \\ G_{in} \\ R_{in} \end{bmatrix}$$

The diagonal values may be set to correct the white balance (XSP_WB_11, XSP_WB_22 and XSP_WB_33).

2.4.5. Resolution and Region of Interest (ROI)

All the IDT cameras support region of interest. The user may select an area of the sensor with some limitations. The area width must be a multiple of 16 or 32 pixels (depending on sensor), the height must be a multiple of 4 pixels.

The sensor supports vertical windowing and the camera cuts the image to match the configured ROI. See the picture below.



The picture shows the full sensor area (black rectangle), the ROI area configured by the user (red rectangle) and the actual ROI area configured on the camera (yellow rectangle). The sensor is not able to select the red area and acquires the images in the yellow area. The camera extract the red area and stores it in camera memory.

For this reason, the maximum frame rate changes only if the vertical resolution changes.

Extended resolutions

Some cameras have a set of extended resolutions that may be configured with the XSP_HD_ROI parameter. Some of those resolutions are made by up sizing a region of the sensor (U), some by downsizing a larger region of it (D). Each value of the XS_HD_ROI parameter has a different meaning according to the camera model. The table below shows the supported resolutions.

Index	Y3C/N3S1-S2	Y3-HD	Y5/NR5/NX5	Os5	Os10
0	1920x1080 (U)				
1	1280x720 (D)				
2		1504x1128 (U)	2560x1920 (U)	2560x1440 (D)	2560x1600 (D)
3			2560x1440 (U)	1920x1080 (D)	2560x1440 (D)
4			2560x1080 (U)	1280x720 (D)	1920x1080 (D)
5					1280x720 (D)

2.4.6. Record modes

Two record modes are available (**XSP_REC_MODE** parameter):

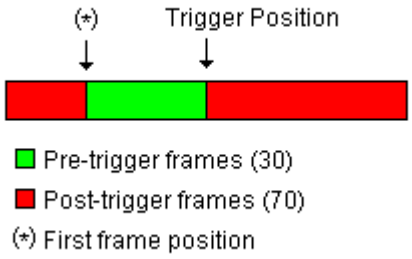
- **Normal (XS_RM_NORMAL)**: the camera starts to acquire and stops when the configured area is filled. The synchronization may be internal or external. The event trigger does not effect the acquisition.
- **Circular (XS_RM_CIRCULAR)**: the camera starts the acquisition and, when the configured memory area is filled, restarts from the beginning. The synchronization may be internal or external. The event trigger is required to complete the acquisition. When the event trigger is detected, the camera acquires the post-trigger frames and then stops. The user should configure the number of pre-trigger frames.

The number of frames are configured in the XSP_FRAMES parameter, the number of pre-trigger frames are configured in the XSP_PRE_TRIG parameter. The same values are arguments of the XsMemoryStartGrab routine.

Three options are available:

- **XSP_PRE_TRIG = 0:** the event trigger starts the acquisition. After the trigger the camera acquires **XSP_FRAMES** frames.
- **XSP_PRE_TRIG = XSP_FRAMES**: the event trigger stops the acquisition. After the trigger the camera doesn't acquire any more frame.
- **XSP_PRE_TRIG < XSP_FRAMES**: the camera acquires continuously. When the event trigger is detected, the camera acquires the post-trigger frames, that are corresponding to (**XSP_FRAMES–XSP_PRE_TRIG–1)**, then it stops.

After the acquisition, the position of the “trigger frame” may be anywhere in the recorded memory (See below).



The position of the trigger frame is returned by the **XsMemoryReadTriggerPosition** routine. order to read frames in correct order, the procedure is the following:

- Read the trigger frame position.
- Read the values of total number of frames (nFrames) and pre-trigger frames (nPreTrig).
- calculate the position of the first frame of the sequence (nFirst).
- Read from nFirst to nFrames-1, then from 0 to nFirst-1.

2.4.7. Synchronization modes

The source of the sync may be configured (**XSP_SYNCIN_CFG** parameter):

- **Internal (XS_SIC_INTERNAL)**: the camera does not care about any external sync signals and acquires at internal frame rate. The rate has been configured by the XSP_PERIOD parameter.
- **External edge-high (XS_SIC_EXT_EDGE_HI)**: the camera exposure starts when the external sync signal edge goes from low to high.
- **External edge-low (XS_SIC_EXT_EDGE_LO)**: the camera exposure starts when the external sync signal edge goes from high to low.
- **External pulse-high (XS_SIC_EXT_PULSE_HI)**: the camera exposure starts when the external signal goes from low to high and corresponds to the pulse duty cycle.
- **External pulse-low (XS_SIC_EXT_PULSE_LO)**: the camera exposure starts when the external sync signal edge goes from high to low and it corresponds to the low level.
- **External IRIG/GPS (XS_SIC_IRIG_DTS_EXT)**: the synchronization is controlled by an IRIG/GPS 1PPS signal and the camera aligns the acquisition timing to that signal. The module is external.
- **Internal IRIG/GPS (XS_SIC_IRIG_DTS_INT)**: the synchronization is controlled by an IRIG/GPS 1PPS signal and the camera aligns the acquisition timing to that signal. The module is internal.
- **External 1PPS (XS_SIC_1PPS)**: the synchronization is controlled by an external 1PPS signal connected to the camera sync in. The camera aligns the acquisition timing to that signal.
- **External Precision Time Protocol (XS_SIC_PTP)**: the synchronization is controlled by PTP and the camera aligns the acquisition timing to that signal.
- **External edge-high in Phase Lock Loop (XS_SIC_EPLL_EDGE_HI)**: the camera is sync-ed to the rising edge of the external signal and keeps recording if the signal disappears.
- **External edge-low in Phase Lock Loop (XS_SIC_EPLL_EDGE_LO)**: the camera is sync-ed to the falling edge of the external signal and keeps recording if the signal disappears.
- **External dynamic pulse-high (XS_SIC_EDYN_PULSE_HI)**: the camera acquires in external high pulse mode and dynamically follows the input signal when the width changes.
- **External dynamic pulse-low (XS_SIC_EDYN_PULSE_LO)**: the camera acquires in external low pulse mode and dynamically follows the input signal when the width changes.

2.4.8. Triggering

If the record mode is set to “Circular” the event trigger may be issued via the external “Trig-In” connector (a pulse or a switch closure) or via software (a call to the **XsTrigger** routine). The trigger may be configured as in the list below (**XSP_TRIGIN_CFG** parameter):

- **Edge High (XS_TIC_EDGE_HI)**: the trigger is detected when the input signal goes from low to high.
- **Edge Low (XS_TIC_EDGE_LO)**: the trigger is detected when the input signal goes from high to low.
- **Switch closure (XS_TIC_SWC)**: the trigger is detected when the poles of the trigger connector are shortened (it corresponds to edge-low).
- **Disabled (XS_TIC_DISABLED)**: the camera does not receive any software or hardware trigger.

2.4.9. Sync Out modes

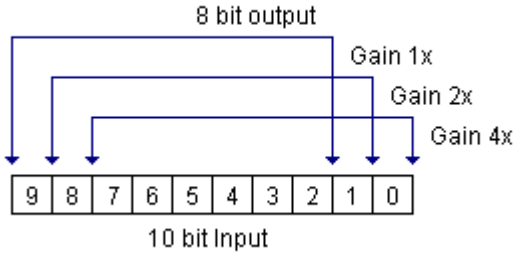
The cameras have a sync out connector that can be used to synchronize other cameras. The XSP_SYNCOUT_CFG parameters configures the sync out signal.

- **Default (XS_SOC_DFL)**: the sync out follows the camera sync. The frequency of the signal is the camera frame rate and the duty cycle is the exposure.
- **Inverted default (XS_SOC_DFL_INV)**: the same as above but inverted.
- **Configurable width (XS_SOC_CFGWID)**: the signal frequency is corresponding to the camera frequency, but the duty cycle is configurable (parameter XSP_SYNCOUT_WID).
- **Inverted Configurable width (XS_SOC_CFGWID_INV)**: same as above but inverted.
- **Disabled (XS_SOC_DISABLED)**: no signal is produced on the sync out.
- **Double exposure (XS_SOC_DBLEXP)**: the sync out reproduce the timing of the double exposure mode.
- **1PPS (XS_SOC_1PPS)**: deprecated.

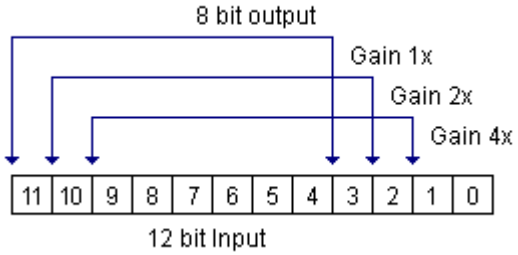
2.4.10. Pixel Gain

IDT cameras sensors are 12 bit (Os series and Y6) or 10 bit (other models). The data is stored in 10/12 bit format into the DDR. 8-bit data can be extracted from 12/10 bit data with the Pixel Gain parameter (**XSP_PIX_GAIN**).

If we call bit0 the least significant pixel and bit9 the most significant pixel, we can extract 8 bits from 10 like in the picture below.



The same extraction can be applied to the 10 most significant pixels of 12 bit data (see the picture below).



The table below shows the pixel depth of IDT camera models and pixel gain support.

Camera	Pixel depth	Pixel Gain
MotionXtra O/Os	12 bit	Yes
CrashCam	12 bit	Yes
CrashCam Mini	12 bit	Yes
MotionPro Y	10 bit	Yes
MotionXtra N/NR/NX	10 bit	Yes
X-Stream PCIe/TB	10 bit	Yes
MotionScope M	10 bit	Yes

2.4.11. Look-up Table (LUT)

The *look up table* (LUT) transformations are basic image-processing functions that may be used to improve the contrast and the brightness of an image by modifying the dynamic intensity of region with poor contrast. LUT transformations can highlight details in areas containing significant information, at the expense of other areas.

A LUT transformation converts input gray-level or color values (8, 10 or 12 bit) into other gray-level or color values. The transfer function has an intended effect on the brightness and contrast of the image. Each input value is transformed into a new value by a *transfer function*

Output value = F (input value)

Where F is a linear or nonlinear, continuous or discontinuous transfer function defined over the interval [0, max]. In case of an 8-bit image, a LUT is a table of 256/1024 or 4096 elements (depending on sensor pixel depth). Each element of the array represents an input value. Its content indicates the output value.

The SDK has a set of 5 fixed look up tables and one user LUT.

XSP_LUT parameter: the parameter configures the LUT selection (XS_LUT_A to XS_LUT_BT2020 for fixed look up tables, XS_LUT_USER for user).

XsLoadLookupTable routine: if the XSP_LUT parameter is set to XS_LUT_USER, a custom look up table may be sent to the camera. The code below shows how to build a user LUT.

XSI_SNS_PIX_DEPTH info: it returns the sensor pixel depth. The returned value is used to set the user LUT size (see the code below).

```
// Configure the camera
XSULONG32 nSnsPD=0,nHiVal=0;
XSULONG32 i,nSize,nValue;
unsigned short anLUT[4096];

XsGetCameraInfo(hCamera, XSI_SNS_PIX_DEPTH, &nSnsPD, &nHiVal);

// calculate the LUT size (example: 10 corresponds to 1024)
nSize = 1<<nSnsPD;
// Build a LUT that sets a gain of 2
for(i=0; i<nSize; i++)
{
	nValue = 2*i;
	if(nValue>(nSize-1)) anLUT[i] = nSize-1;
	else anLUT[i] = nValue;
}
// Configure the LUT
XsSetParameter(hCamera, &xsCfg, XSP_LUT, XS_LUT_USER);
XsLoadLookupTable(hCamera, anLUT, nSize);

// Configure the camera
XsRefreshCameraSettings( hCamera, &xsCfg );
```

2.4.12. Auto-exposure

The parameters that control the auto-exposure are shown below:

Parameter	Description
XSP_AE_ENABLE	It enables and disables the auto-exposure
XSP_AE_ROIX, XSP_AE_ROIY, XSP_AE_ROIWIDTH, XSP_AE_ROIHEIGHT	It configures the reaction speed to changes in the image intensity (0: slow, reacts within a few frames and averages; 7: fast, reacts within one frame).
XSP_AE_SPEED	configures the auto-exposure region of interest. That is, the area of the sensor that is used to automatically configure the exposure
XSP_AE_REFERENCE	It configures the reference value of intensity (the “Luminance” in the AE region of interest). The camera changes the exposure to make the image intensity in the AE ROI equal to this value.
XSP_AE_CUR_LUMA	It's a read-only parameter (the current value of the intensity in the AE region of interest). This value should be read before enabling the AE and used to set the reference value.

Follow the steps below to configure the auto-exposure parameters in the camera.

Open and configure the camera: Disable the auto-exposure (XSP_AE_ENABLE=0) and set the auto-exposure region of interest (XSP_AE_ROIX, XSP_AE_ROIY, XSP_AE_ROIWIDTH, XSP_AE_ROIHEIGHT).

```
// Configure the camera
XsSetParameter(hCamera, &xsCfg, XSP_AE_ENABLE, 0);
XsSetParameter(hCamera, &xsCfg, XSP_AE_ROIX, 256);
XsSetParameter(hCamera, &xsCfg, XSP_AE_ROIY, 256);
XsSetParameter(hCamera, &xsCfg, XSP_AE_ROIWIDTH, 512);
XsSetParameter(hCamera, &xsCfg, XSP_AE_ROIHEIGHT, 512);

// Configure the camera
XsRefreshCameraSettings( hCamera, &xsCfg );
```

Calculate the reference value. Snap an image with the desired intensity and read the current reference value from the camera (XSP_AE_CUR_LUMA).

```
XSULONG32 nRefLuma;
// Snap
XS_FRAME frm;
frm.pBuffer = m_pDataBuf;
frm.nBufSize = m_nBuSize;
fmr.nImages = 1;
XsSynchGrab(hCamera, &frm, 1000);
// read the reference
XsGetParameter(hCamera, &xsCfg, XSP_AE_CUR_LUMA, (nRefLuma);
```

Configure the camera for AE: set the value of AE reference (XSP_AE_REFERENCE), set the desired speed (XSP_AE_SPEED) and enable the auto-exposure (XSP_AE_ENABLE=1).

```
// set the parameters
XsSetParameter(hCamera, &xsCfg, XSP_AE_SPEED, nRefLuma);
XsSetParameter(hCamera, &xsCfg, XSP_AE_SPEED, 3);
XsSetParameter(hCamera, &xsCfg, XSP_AE_ENABLE, 1);

// Configure the camera
XsRefreshCameraSettings( hCamera, &xsCfg );
```

Record: set the camera in live mode (XsSynchGrab) or record a sequence in camera memory (XsMemoryStartGrab).

2.4.13. HDMI/SDI output and Video modes

HDMI output is active on Y cameras and SDI is active in Os/CC and Ccmini models. Those models can display live images and playback acquisitions via HDMI/SDI output independently from the computer. HDMI/SDI output is configured via the **XSP_HDMI_MODE** parameters. The available modes are:

- **XS_HDMI_OFF**: HDMI/SDI is disabled. No output is produced.
- **XS_HDMI_ON**: HDMI/SDI is enabled. Any time the XsSynchGrab or the XsMemoryReadFrame routines are called, the image is simultaneously sent to the computer (via USB or Ethernet) and to the HDMI output.
- **XS_HDMI_TRANSFER**: deprecated.
- **XS_HDMI_INDEPENDENT**: HDMI/SDI is enabled and independent from the computer. The XsReadToVideo routine should be called to generate output to the HDMI. XsSynchGrab and XsMemoryReadFrame do not affect HDMI. The user can activate Live to the computer and playback images to the HDMI, or download images to the computer and do live on HDMI.

Y and O cameras support different HDMI modes and formats. The table below shows which parameters may be configured for video output in Y cameras.

Parameter	Description
XSP_HDMI_MODE	0: Disabled 1: Enabled (images go to the PC and to the HDMI video) 3: Images may be independently shown on PC or HDMI/SDI.
XSP_VIDEO_MODE (Y)	0: 1280 x 720 – 60 Hz 1: 1920 x 1080 – 60 Hz 2: 1920 x 1080 – 25 Hz 3: 1920 x 1080 – 40 Hz 4: 1920 x 1080 – 30 Hz
XSP_VIDEO_MODE (Os/CC/CCm)	0: 1920 x 1080 – 60 Hz 1: 1920 x 1080 – 50 Hz 2: 1920 x 1080 – 30 Hz 3: 1920 x 1080 – 25 Hz 4: 1920 x 1080 – 24 Hz 5: 1280 x 720 – 60 Hz 6: 1280 x 720 – 50 Hz 7: 1280 x 720 – 30 Hz 8: 1280 x 720 – 25 Hz 9: 1280 x 720 – 24 Hz
XSP_HDMI_OVERLAY	Disable/Enable a data overlay that shows fps, exposure, resolution and other timing parameters

Some of the parameter may apply to X and HG legacy cameras.

- **X cameras**: XSP_VIDEO_MODE enables PAL mode (0) or NTSC mode (1).
- **HG-legacy cameras**: XSP_HDMI_MODE enables or disables the video output. XSP_VIDEO_MODE controls the PAL/NTSC configuration.

2.4.14. Binning

Binning is the process of combining adjacent pixels of the sensor during readout. The primary benefit of binning is improved signal to noise ratio (SNR), albeit at the expense of reduced spatial resolution.

The maximum image size for each binning is not an even division of the 1x1 maximum image size. When you switch from a value of binning to another and update the configuration to the camera, your binning values will be adjusted to fit the new binning mode. When binning, ROI is specified in “super-pixels”.

A common mistake occurs when switching from higher binning, such as 4x4, to lower binning, such as 1x1. If the caller forgets to adjust the region, they will end up with the old 4x4 size. When switching binning modes, you might want to select the largest possible region as follows:

```
XSULONG32 nMaxWid,nMaxHgt;

// Get the current maximum image size
XsGetParameter( hCamera, &xsCfg, XSP_MAX_WIDTH, &nMaxWid );
XsGetParameter( hCamera, &xsCfg, XSP_MAX_HEIGHT, &nMaxHgt );

// Reset ROI to the new maximum values
XsSetParameter( hCamera, &xsCfg, XSP_ROIX, 0 );
XsSetParameter( hCamera, &xsCfg, XSP_ROIY, 0 );
XsSetParameter( hCamera, &xsCfg, XSP_ROIWIDTH, nMaxWid );
XsSetParameter( hCamera, &xsCfg, XSP_ROIHEIGHT, nMaxHgt );
```

2.5. Image Grab in camera or computer DDR

Some camera models (Y, Nx, Os, O, CC and CC-mini) have onboard DDR. Some other models don't (Xstream PCIe and Xstream Mini) and stream images in computer memory.

2.5.1. Asynchronous Live

IDT cameras support a fast way to grab live images: XsLive(). Some old models do not support it, and the XSI_FAST_LIVE info returns 0.

An example of the code that implements fast live is shown below.

```
// example: enable fast live
XS_FRAME frame;
XSULONG32 nWidth, nHeight, nPixDepth, nBufSize;

// Image size depends on the current ROI & image format.
XsGetParameter( hCamera, &xsCfg, XSP_ROIWIDTH, &nWidth );
XsGetParameter( hCamera, &xsCfg, XSP_ROIHEIGHT, &nHeight );
XsGetParameter( hCamera, &xsCfg, XSP_PIX_DEPTH, &nPixDepth );

// Fill out fields in XS_FRAME structure.
if( nPixDepth<9 ) nBufSize = nWidth*nHeight;
else if( nPixDepth<17 ) nBufSize = 2*nWidth*nHeight;
else if( nPixDepth<25 ) nBufSize = 3*nWidth*nHeight;
else nBufSize = 6*nWidth*nHeight;
frame.nBufSize = nBufSize;
frame.pBuffer = malloc(frame.nBufSize);
frame.nImages = 1;

// start live
XsLive(m_hCam, XS_LIVE_START);

// grab images in a loop (or in a thread) and display
while()
{
     // grab
     XsMemoryPreview(m_hCam, &frame, pnFrameIndex);
     // display
     ...
}

// stop fast live
XsLive(hCam, XS_LIVE_STOP);
```

NOTE: if fast live is on, the routine XsSynchGrab cannot be called. The user should call the XsMemoryPreview routine. The XS_FRAME structure should be filled with the same information of XsSynchGrab.

NOTE2: if a camera supports fast live, we recommend to use it instead of XsSynchGrab because it's faster.

2.5.2. Synchronous Live

To synchronously grab an image from a camera, allocate an image buffer of enough size, fill an XS_FRAME structure, then call a grab function. Below is an example of a synchronous frame grab.

```
XS_FRAME frame;
XSULONG32 nWidth, nHeight, nPixDepth, nBufSize;

// Image size depends on the current ROI & image format.
XsGetParameter( hCamera, &xsCfg, XSP_ROIWIDTH, &nWidth );
XsGetParameter( hCamera, &xsCfg, XSP_ROIHEIGHT, &nHeight );
XsGetParameter( hCamera, &xsCfg, XSP_PIX_DEPTH, &nPixDepth );

// Fill out fields in XS_FRAME structure.
if( nPixDepth<9 ) nBufSize = nWidth*nHeight;
else if( nPixDepth<17 ) nBufSize = 2*nWidth*nHeight;
else if( nPixDepth<25 ) nBufSize = 3*nWidth*nHeight;
else nBufSize = 6*nWidth*nHeight;
frame.nBufSize = nBufSize;
frame.pBuffer = malloc(frame.nBufSize);
frame.nImages = 1;
// Do synchronous image grab with a 5 sec time out
XsSynchGrab( hCamera, &frame, 5000 );
// Process the data
...
// free the buffer
free(frame.pBuffer);
```

XsSynchGrab returns when the image has been acquired and read. The process is slower than XsLive because the camera executes a small acquisition of a few frames every time XsSynchgrab is called.

In double exposure mode the camera acquires 2 images, so the user must provide enough space in the buffer before calling the XsSynchGrab. The example below shows how to grab an 8 bit image pair in double exposure.

```
XS_FRAME frame;
XSULONG32 nWidth,nHeight;
// Image size depends on the current ROI & image format.
XsGetParameter( hCamera, &xsCfg, XSP_ROIWIDTH, &nWidth );
XsGetParameter( hCamera, &xsCfg, XSP_ROIHEIGHT, &nHeight );
// Fill out fields in XS_FRAME structure.
frame.nBufferSize = 2*nWidth*nHeight;
frame.pBuffer = malloc(frame.nBufferSize);
frame.nImages = 2;
// Do synchronous image grab with a 5 sec time out
XsSynchGrab( hCamera, &frame, 5000 );
// free the buffer
free(frame.pBuffer);
```

2.5.3. Image Grab in camera memory

Some digital camera models have on-board memory. The user may acquire sequences of images into camera memory and then transfer them to the host PC memory or to the hard disk. During the acquisition process, the latest frame acquired may be previewed.

First few MB of the camera memory are reserved for Live. Read **XSI_LIVE_BUF_SIZE** info and use the value as offset to recording start address.

There are two ways to detect if the acquisition has been executed.

- **Install a callback**: the source code below shows how to start a camera acquisition and set a callback routine that will be called when the acquisition is finished. The main program should wait until the callback is called.

```
HANDLE hEvent;
XSULONG32 nStartAddLo=0, nStartAddHi=0;

// create an event object and start the acquisition
XsGetCameraInfo(hCamera,XSI_LIVE_BUF_SIZE,&nStartAdd,
                &nStartAddHi)

// create an event object and start the acquisition
void StartAcquisition()
{
    hEvent = CreateEvent(NULL,FALSE,FALSE,"Event");
    XsMemoryStartGrab(hCamera,nStartAddLo,nStartAddHi,100,0,
                        fcallback, XS_CF_DONE, hEvent);
}

// callback routine (signals the event)
void XSTREAMAPI fcallback(void *pUserData, XS_ERROR nErrCode,
                                  XSULONG32 nFlags)
{
    HANDLE h = (HANDLE)pUserData;
    SetEvent(h);
}

// the main program previews and waits on the event object
BOOL WaitAcquisitionToFinish()
{
    XS_FRAME xf;
    // init XS_FRAME (see XsSynchGrab)
    ...
    // loop on the event and preview
    while( WaitForSingleObject(hEvent,50)== WAIT_TIMEOUT )
    {
        XsMemoryPreview(hCamera,&xf,NULL);
        // display the frame
        ...
    }
    return TRUE;
}
```

- **Poll the camera status** until it returns the “acquisition done” value. The code below shows how to start and acquisition and check the camera status to detect that the images have been correctly acquired.

```
XSULONG32 nStartAddLo=0, nStartAddHi=0;

// read the live offset and use it as start address
XsGetCameraInfo(hCamera,XSI_LIVE_BUF_SIZE,&nStartAdd,
                &nStartAddHi)

// start the acquisition (do not install any callback)
void StartAcquisition()
{
    XsMemoryStartGrab(hCamera,nStartAddLo,nStartAddHi,100,0,
                        NULL, 0, NULL);
}

// the main program polls the status and previews
BOOL WaitAcquisitionToFinish()
{
    XS_FRAME xf;
    XSULONG32 nBusy, nSts;
    // init XS_FRAME (see XsSynchGrab)
    ...
    // loop on the event and preview
    while( (nTime - nStarTime) < nTimeOut )
    {
        nBusy = nSts = 0;
        XsGetCameraStatus(hCamera,&nBusy,&nSts,0,0,0,0);
        if(nBusy==0 &&
           nSts!=XSST_REC_PRETRG && nSts!=XSST_REC_POSTRG)
           break;
        // preview
        XsMemoryPreview(hCamera,&xf,NULL);
        // display the frame
        ...
        // delay and calculate the time
    }
    return TRUE;
}
```

2.5.4. Multiple Acquisitions in camera memory

The camera DDR can be addressed allowing the user to make multiple acquisitions.

However, the camera DOES NOT remember the settings of each acquisition: if two or more acquisitions have been done with different configuration parameters (ROI, pixel depth, etc.) the user program MUST remember those settings and configure the camera before each image read.

The first acquisition in camera memory should start from the value returned by the XSI_LIVE_BUF_SIZE info.

```
XSULONG32 nAdd1Lo=0, nAdd1Hi=0;

// read the live offset and use it as start address
XsGetCameraInfo(hCamera, XSI_LIVE_BUF_SIZE,&nAdd1Lo, &nAdd1Hi);
// set the number of frames
XsSetParameter(hCamera, &xsCfg, XSP_FRAMES, 100);

// set other parameters

// configure
XsRefreshCameraSettings( hCamera, &xsCfg );
```

The next available address may be calculated from the number of recorded frames and the size of each recorded frame. Before configure other values of address or frames make sure that you have saved the values of your first acquisition.

```
// save old values
XSULONG32 nSize1,nFrms1,nAdd1Lo,nAdd1Hi;
XSUINT64 nAdd1;

// read address and convert into 64 bit number
XsGetParameter(hCamera, &xsCfg, XSP_STARTADDRLO, &nAdd1Lo);
XsGetParameter(hCamera, &xsCfg, XSP_STARTADDRLO, &nAdd1Hi);
nAdd1 = (XSUINT64)nAdd1Lo + (((XSUINT64)nAdd1Hi)<<32);
// read frame size of current acquisition and save it
XsGetParameter(hCamera, &xsCfg, XSP_FRAMES, &nFrms1);
XsGetParameter(hCamera, &xsCfg, XSP_FRAME_SIZE, &nSize1);
// calculate next address
XSULONG32 nAdd2Lo,nAdd2Hi,nFrms2=200;
XSUINT64 nAdd1;
nAdd2 = nAdd1 + (XSUINT64)nFrms1*(XSUINT64)nSize1;
nAdd1Lo = (XSULONG32)nAdd2;
nAdd2Hi = (XSULONG32)(nAdd2>>32);
// use values to acquire a new segment
XsSetParameter(hCamera, &xsCfg, XSP_STARTADDRLO, nAdd2Lo);
XsSetParameter(hCamera, &xsCfg, XSP_STARTADDRLO, nAdd2Hi);
XsSetParameter(hCamera, &xsCfg, XSP_FRAMES, nFrms2);
// configure
XsRefreshCameraSettings( hCamera, &xsCfg );
```

The sequence below shows how to switch from one acquisition to another and save images from two acquisitions.

- **Backup** the camera settings of both acquisitions.
- **Configure** the camera with the parameters of first acquisition.
- **Transfer** the first acquisition images.
- **Configure** the camera with the parameters of second acquisition.
- **Transfer** the second acquisition images.

2.5.5. Image Grab in computer memory (streaming cameras)

X-Stream PCIe and XS-Mini cameras do not have on-board memory and images are acquired in computer memory.

In PCIe and TB cameras, live is implemented as fast live (see “Image Live” topic above).

The DMA buffer allocated by the driver is the “virtual” camera DDR. The images may be acquired by calling XsMemoryStartGrab with address 0 (see below).

```
HANDLE hEvent;

// create an event object and start the acquisition
void StartAcquisition()
{
    hEvent = CreateEvent(NULL,FALSE,FALSE,”Event”);
    XsMemoryStartGrab(hCamera,0,0,100,0,
                      fcallback, XS_CF_DONE, hEvent);
}

// callback routine (signals the event)
void XSTREAMAPI fcallback(void *pUserData, XS_ERROR nErrCode,
                          XSULONG32 nFlags)
{
    HANDLE h = (HANDLE)pUserData;
    SetEvent(h);
}

// the main program previews and waits on the event object
BOOL WaitAcquisitionToFinish()
{
    XS_FRAME xf;
    // init XS_FRAME (see XsSynchGrab)
    ...
    // loop on the event and preview
    while( WaitForSingleObject(hEvent,50)== WAIT_TIMEOUT )
    {
        XsMemoryPreview(hCamera,&xf,NULL);
        // display the frame
        ...
    }
    return TRUE;
}
```

DMA memory cannot be partitioned, so multiple acquisitions cannot be done. Once the mages are acquired, each frame can be read with the XsMemoryReadFrame (see the topic below).

2.5.6. Read images acquired in normal or circular mode

If images have been acquired in normal mode, they are sorted in camera (or computer memory). They can be read as it is, starting from the recording address.

```
// the address below must be 0,0 for streaming cameras
// nAddLo, nAddHi

// read 10 frames
for (int i=0;i<10;i++)
{
   XsMemoryReadFrame(hCamera, nAddLo, nAddHi, i, pBuf);
}
```

In circular mode, the first (oldest) frame may be anywhere in the camera (or computer) memory buffer allocated for the acquisition. If you wish to read the images in sorted order, do the following.

- Read trigger position XsMemoryReadTriggerPosition.
- Get nFrames, nPreTriggerFrames (they can be read with XsGetParameter).
- Find first frame position (nFirstFrame).
- Read from nFirstFrame to nFrames-1, then from 0 to nFirstFrame-1.

```
// after recording read the trigger position
XSULONG32 nAddLo,nAddHi;
XSULONG32 nFrame,nPreTrig;
XSULONG32 nPosLo,nPosHi,nTrgIdx,nTime,nStartIdx;
int i;

// read trigger position to order frame indexes
XsGetParameter(hCam,&xsCfg, XSP_FRAMES, &nFrames);
XsGetParameter(hCam,&xsCfg, XSP_PRE_TRIG, &nPreTrig);

// read trigger position to order frame indexes
XsMemoryReadTriggerPosition(hCam,nPosLo,PosHi,nTrgIdx,nTime);

// find first frame position
if( nTrgIdx>=nPreTrig ) nStartIdx = nTrgIdx - nPreTrig;
else nStartIdx = nFrames - (nPreTrg - nTrgIdx);

// it can be also calculated in a single op
// nStartIdx = (nTrgIdx + nFrames - nPreTrg)%nFrames;

// sort the frames (read from start frame to nFrames - 1)
for(i=nStartIdx; i<nFrames; i++)
    XsMemoryReadFrame(hCam,nAddLo,nAddHi,i,pBuf);
// read from 0 to start index - 1
for(i=0; i<nStartIdx; i++)
    XsMemoryReadFrame(hCam,nAddLo,nAddHi,i,pBuf);
```

2.6. Image grab in camera SSD

Os, O, CC and CC-mini camera models are equipped with a solid state disk (or removable SD-card) that can store images.

2.6.1. SSD Backup mode

In backup mode, the camera records images in the DDR and, when the images are acquired, it transfers the data to the SSD. There is no limit in the frame rate, but the number of frames must fit in the camera DDR.

The procedure is shown below.

- Configure the camera with the required parameters. Set the XSP_PROP parameter to XS_PR_SSD_BACKUP. Make sure that the XSP_FRAMES parameter value is lower or equal to the maximum number of frames that fit in the camera DDR.
- Before recording images for the first time, make sure you have a clean SSD. Call the XsEraseDisk routine. If the SSD contains images that you don't want to erase, make sure that you don't overwrite them in the acquisition. To do that, check the "Read the images from the SSD" section and calculate the record address from the addresses of the previous acquisitions.
- The area used to store the images before saving them to the SSD is calculated by removing the live area (XSI_LIVE_BUF_SIZE) from the DDR size (XSI_MEMORY).
- To start recording, call the XsMemoryStartGrab. Then trigger the camera and wait for the callback or check the camera status.

```
XSULONG32 nStartAddLo, nStartAddHi, nFrames;
XSULONG32 nFrmSize, nMemLo, nMemHi, nLiveOffs, nHi;

// erase the SSD if doesn't contain useful images
XsEraseDisk(hCamera);

// Configure the camera
XsSetParameter(hCamera, &xsCfg, XSP_PROP, XS_PR_SSD_BACKUP);
XsRefreshCameraSettings( hCamera, &xsCfg );

// set the correct address
nStartAddLo = 0;
nStartAddHi = 0;

// calculate the max number of frames (fit in the DDR)
XsGetParameter(hCamera, &xsCfg, XSP_FRAME_SIZE, &nFrmSize);
XsGetCameraInfo(m_hCam, XSI_MEMORY, &nMemLo, &nMemHi);
XsGetCameraInfo(m_hCam, XSI_LIVE_BUF_SIZE, & nLiveOffs, &nHi);

// divide the DDR size minus the live area by the frame size
// the frame size changes if the ROI changes
nFrames = (XSULONG32)((((XSINT64)nMemLo+(XSINT64)(nMemHi<<32))-
    (XSINT64)nLiveOffs)/nFrmSize);

// start recording
XsMemoryStartGrab(hCamera, nStartAddLo, nStartAddHi,
    nFrames, 0,fcallback, XS_CF_DONE, this);
```

2.6.2. SSD Streaming mode

In streaming mode, the camera records images in the DDR and simultaneously stores them in the SSD. The maximum frame rate is limited by the SSD write speed. In this mode the user may acquire a number of images larger than the DDR size.

- Configure the camera with the required parameters. Set the XSP_PROP parameter to XS_PR_SSD_STREAMING. To calculate what is the number of frames you can acquire for a given frame rate, read the XSP_SSD_MAX_FRMS parameter. Make sure that the XSP_FRAMES parameter value is lower or equal to that value, otherwise you may receive an SSD write overrun error.
- Another important parameter that should be read is the XSP_SSD_STRM_PER parameter, that is, the inverse of the "streaming fps". The streaming fps is the value of the frame rate that corresponds to the write to disk speed.
- If the frame rate is slower than the "streaming fps", the SSD is "faster" than the DDR, then the camera can record any number of frames without overrun. In this case the maximum number of frames is limited by the SSD size.
- If the frame rate is faster than the "streaming fps", the SSD is "slower" than the DDR, then the camera has a limited number of frames that can be acquired before overrun (XSP_SSD_MAX_FRMS). This number is calculated assuming that the record mode is normal. If the record mode is circular, the overrun may occur if the user waits too long before triggering the camera.
- Before recording images for the first time, make sure you have a clean SSD. Call the XsEraseDisk routine. If the SSD contains images that you don't want to erase, make sure that you don't overwrite them in the acquisition. To do that, check the "Read the images from the SSD" section and calculate the record address from the addresses of the previous acquisitions.
- To start recording, call the XsMemoryStartGrab. Then trigger the camera and wait for the callback or check the camera status.

```
XSULONG32 nStartAddLo, nStartAddHi, nFrames;
XSULONG32 nFrmSize, nMemLo, nMemHi;

// nFrames is the number of images you want to acquire
nFrames = ...
// erase the SSD if doesn't contain useful images
XsEraseDisk(hCamera);

// Configure the camera
XsSetParameter(hCamera, &xsCfg, XSP_PROP, XS_PR_SSD_STREAMING);
XsRefreshCameraSettings( hCamera, &xsCfg );

// set the correct address
nStartAddLo = nStartAddHi = 0;
// calculate the max number of frames
XsGetParameter(hCamera, &xsCfg, XSP_SSD_MAX_FRMS, &nMaxFrames);

// make sure that you record less than Max frames
if(nFrames>nMaxFrames) nFrames = nMaxFrames;
// start recording
XsMemoryStartGrab(hCamera, nStartAddLo, nStartAddHi, nFrames, 0,
                  fcallback, XS_CF_DONE, this);
```

2.6.3. Read images from SSD

The images stored in the camera SSD cannot be directly read, but they have to be copied to the camera DDR. The DDR acts as a temporary buffer from which the images may be read for playback or download.

The procedure is shown below:

- Read the list of acquisitions that are already stored in the SSD. the routine is **XsReadCameraSettingsArray**. The nOption parameter need to be set to 1.
- The routine returns an array of XS_SETTINGS structure. The parameters of each acquisition may be retrieved by the structure with the **XsGetParameter** routine. Important parameters are XSP_STARTADDRLO, XSP_STARTADDRHI (read the address in the SSD address space), XSP_FRAME_SIZE (the size of a frame), XSP_FRAMES (the total number of acquired frames). Those parameters may be used to calculate the total size of the acquisition in the camera SSD.
- Select the acquisition that will be downloaded by sending the corresponding parameters to the camera (**XsRefreshCameraSettings**).
- Call the **XsMemoryReadFromDisk**. The DDR address may be set to 0 and the SSD address is set to the start address of the corresponding acquisition. Then the frames may be copied in groups of N images by setting the start and the stop index. The number of frames that will be copied from the SSD to the DDR may be the total number of frames if the acquisition fits in the camera DDR, or a multiple of 256 if it doesn't.

```
XS_SETTINGS aXCfg[256] = {0};
XSULONG32 nAddLo, nAddHi, nCount, nFrames;
XSULONG32 nFrmSz, nMemLo, nMemHi, nDDRFrms, nStart, nStop;

// read the array of configurations
nCount = sizeof(aXCfg)/sizeof(XS_SETTINGS);
XsReadCameraSettingsArray(hCam, 1, aXCfg, 0, &nCount);

// select the configuration number 2 (just an example!!!)
XsRefreshCameraSettings( hCam, &aXCfg[2] );
// read address
XsGetParameter(hCam, &aXCfg[2], XSP_STARTADDRLO, &nAddLo);
XsGetParameter(hCam, &aXCfg[2], XSP_STARTADDRHI, &nAddHi);
XsGetParameter(hCam, &aXCfg[2], XSP_FRAMES, &nFrames);

// calculate the max number of frames
XsGetParameter(hCam, &aXCfg[2], XSP_FRAME_SIZE, &nFrmSz);
XsGetCameraInfo(hCam, XSI_MEMORY, &nMemLo, &nMemHi);

// divide the DDR size by the frame size
// and find the DDR room
nDDRFrms = (XSULONG32)
    (((XSINT64)nMemLo+(((XSINT64)nMemHi)<<32))/(XSINT64)nFrmSz);

// read frames
nStart = 0;
if (nFrames<nDDRFrms) nStop = nFrames-1;
else nStop = 255;

while( nStop<=(nFrames-1) )
{
   XsMemoryReadFromDisk(m_hCam,0,0,nAddLo, nAddHi, nStart,
      nStop, pfnCallback, this);
   nStart += 256;
   nStop  += 256;
   if(nStop>(nFrames-1)) nStop = nFrames-1;
}
```

2.7. Image Streaming to disk (streaming cameras)

X-Stream PCIe cameras (720p and 1440p) and XS-Mini cameras (1540, 3520, 4K and Stick) have streaming to disk capabilities. Raw images captured in computer memory can be saved in real time to local SSD.

Procedure:

- Start camera recording.
- Enable write to disk to start streaming.
- Manage notifications from driver.
- Disable write to disk to stop streaming.
- Trigger or stop camera recording.

The code sample below shows how to do it.

```
// start the acquisition
XsMemoryStartGrab(hCamera,0,0,100,0,
                  fcallback, XS_CF_DONE, this);

...

// enable write to disk
XS_W2DCFG w2d = {0};
w2d.nDrives = 1;
w2d.nOpt = 0;
strcpy(w2d.szVolume1,"C:/");
strcpy(w2d.szDirectory,"Stream2disk");

XsConfigureWriteToDisk(hCam,1,&w2d,w2dCallback,this);
...

// disable write to disk
XS_W2DCFG w2d = {0};
XsConfigureWriteToDisk(hCam,0,&w2d,NULL,NULL);

// manage callbacks
void XSTREAMAPI w2dCallback(void *pUserData,
                            XSULONG32 nParam1,
                            XSULONG32 nParam2,
                            XSULONG32 nErrCode)
{
    // nParam1 shows the number of saved frames disk
    // nParam2 return the percentage of available DDR buffer
    // the range is from 0 to 1000
}
```

In the example above a raw file folder will be created in “C:/Stream2disk” directory.

2.8. RAW files and virtual cameras

2.8.1. Virtual cameras

Virtual cameras have been introduced to convert raw data stored in removable devices, such as SD cards, network disks or raw files. The SDK enumerates virtual cameras like any other camera model, and gives the user the instruments to convert raw data into a correct image format.

To enumerate virtual cameras and convert data, do the following.

- Call XsEnumCameras with the value of **XS_EF_VCAM** in the enumeration filter. The routine will returns the virtual cameras with different values of the nLinkType field.
- **XS_LT_SDCARD**: the driver searches among the removable storage devices for any SD card with stored data.
- **XS_LT_RAWFILE**: the driver searches in the database for raw files.
- Once a virtual cameras has been enumerated, call **XsOpenCamera** to open it, **XsReadCameraSettings** to read the stored parameters, and then **XsMemoryReadFrame** to read the images and store them to the hard disk in any file format.

If you know the path of the raw file, you may also call the **XsOpenRawCamera** routine. The first parameter (lpszRawFilePath) contains the full path to the rawfile.xml file or to the directory that contains the rawfile.xml file.

2.8.2. Save data in RAW format

The images can be downloaded in RAW format from the camera memory directly to the hard disk. To do that, use the XsMemoryDownloadRawFrame routine.

The example below shows how to save a sequence acquired in **normal** mode.

```
// create the path without the extension
// the raw file name will be file.raw
char szPath[256] = "c:\\users\\default\\Images\\Acq01\\file";
int i;

// read from frame 0 to frame N - 1
for(i=0; i<N; i++)
    XsMemoryDownloadRawFrame(hCam,szPath,nAddLo,nAddHi,i,i,N);
```

The example below shows how to save a sequence acquired in **circular** mode.

```
// create the path without the extension
// the raw file name will be file.raw
char szPath[256] = "c:\\users\\default\\Images\\Acq01\\File";
XSULONG32 nPLo,nPHi,nTrgIdx,nTime,nStartIdx;
int i,j;

// read trigger position to order frame indexes
XsMemoryReadTriggerPosition(hCam,&nPLo,&nPHi,&nTrgIdx,&nTime);

// find first frame position
if( nTrgIdx>=nPreTrig ) nStartIdx = nTrgIdx - nPreTrig;
else nStartIdx = nFrames - (nPreTrg - nTrgIdx);

// read from start index to nFrames - 1
i = 0;
for(i=0, j=nStartIdx; j<nFrames; i++, j++)
    XsMemoryDownloadRawFrame(hCam,szPath,nAddLo,nAddHi,j,i,N);
// read from 0 to start index - 1
for(j=0; j<nStartIdx; i++, j++)
    XsMemoryDownloadRawFrame(hCam,szPath,nAddLo,nAddHi,j,i,N);
```

2.8.3. Read data from RAW files

Once the acquisition has been saved to a RAW file, the images data can be read and converted to any format. The RAW file may be open like a “virtual” camera and the images read from the virtual camera memory.

There are two ways to open a raw file:

Before reading the data, use the **XsPreConfigCamera** to set the path to the folder where the raw subfolders are stored, then enumerate the virtual cameras and use the returned ID to open the file (see below).

```
// set the path to the raw acquisitions folder
XsPreConfigCamera(0,XSPP_DB_FOLDER,(void*)szRawFld,0);

// enumerate the virtual cameras
XS_ENUMITEM info[64];
XSULONG32 i, nItems = 64;
XsEnumCameras(info,&nItems,XS_EF_VCAM);

// open the virtual camera with the enumerated ID
XsOpenCamera(info[0].nCameraID,&m_hCam);
```

Open the RAW file with the full path to the folder where the file is stored.

```
// open the virtual camera with the full path
XsOpenRawCamera(szRawPath,&m_hCam);
```

Once the handle to the virtual camera is returned, read the configuration, send it to the driver to refresh the settings and use some of the parameters to detect the number of frames and the pre-trigger.

```
// set the path to the raw acquisitions folder
XS_SETTINGS xsCfg = {0};
xsCfg.cbSize = sizeof(XS_SETTINGS);
XsReadCameraSettings(hCam,&xsCfg);
XsRefreshCameraSettings(hCam,&xsCfg);

// retrieve the parameters used to read the images
XSULONG32 nWid,nHgt,nPixDep,nFrames;
XsGetParameter(m_hCam,&xsCfg,XSP_ROIWIDTH,&nWid);
XsGetParameter(m_hCam,&xsCfg,XSP_ROIHEIGHT,&nHgt);
XsGetParameter(hCam,&xsCfg,XSP_PIX_DEPTH,&nPixDep);
XsGetParameter(m_hCam,&xsCfg,XSP_FRAMES,&nFrames);
```

Then allocate the memory buffer and read the data.

```
// compute the buffer size and allocate the memory
XSULONG32 nSize;
unsigned char* pData;
if(nPD<9) nSize = nWid*nHgt;
else if(nPD<17) nSize = 2*nWid*nHgt;
else if(nPD<25) nSize = 3*nWid*nHgt;
else nSize = 6*nWid*nHgt;

pData = (unsigned char*)malloc(nSize);

// read the data from address 0
XsMemoryReadFrame(hCam,0,0,0,pData);

// convert it with your own routines
...
```

2.9. Miscellaneous

2.9.1. Bayer mode in color cameras

Color cameras store the images in CFA Bayer format (see the appendix of the user manual). The camera may convert those images in RBG format before sensing them to the host computer.

The user may access the FCA images by enabling the raw mode on the camera with the **XsEnableRawMode** routine.

If the raw mode is enabled, the image format is automatically set to Bayer (8 or 16 bit) and the user can read the CFA image from the camera (see below).

```
// the camera is a color camera
// the image format is XS_IM_BGR24
// each pixel is stored in a 3 bytes word

// Enable raw mode
XsEnableRawMode(m_hCam, 1);

// read data in 8 bit Bayer format
XsMemoryReadFrame(m_hCam, 0, 0, 0, pDataBuffer);

// Disable raw mode
XsEnableRawMode( m_hCam, 0);
```

2.9.2. Read data from a BROC session

Hardware BROC (**Burst Record on Command**) is implemented in latest cameras firmware. The acquisition is divided into sections and each section is managed a little circular acquisition. When the camera receives a trigger the firmware automatically switches to the next section and restarts. The user receives one callback only at the end of the latest section. The user may retrieve which section is currently active by reading the XSP_BROC_CURR_SECT parameter.

To detect if a camera supports the hardware BROC, read the **XSI_HW_BROC_SUPPORT** information.

To configure and start a BROC session:

Set the XSP_FRAMES and the XSP_BROC_TOT_LEN to the total number of frames to acquire. Set the XSP_BROC_LEN to the number of frames in each BROC section. Set the XSP_PRE_TRIG parameter to the number of pre-trigger frames in each BROC section. Then call XsMemoryStartGrab with the total number of frames and the pre-trigger frames of a section (see below)

```
// example: configure a BROC session of 200 frames
// with 4 sections of 50 frames each
// each section has 20 pre-trigger and 30 post-trigger frames
XS_SETTINGS xsCfg = {0};
xsCfg.cbSize = sizeof(XS_SETTINGS);

XsSetParameter(m_hCam,&xsCfg,XSP_REC_MODE, XSP_RM_BROC);

XsSetParameter(m_hCam,&xsCfg,XSP_FRAMES, 200);
XsSetParameter(m_hCam,&xsCfg,XSP_BROC_TOT_LEN, 200);
XsGetParameter(hCam,&xsCfg, XSP_BROC_LEN, 50);
XsGetParameter(m_hCam,&xsCfg,XSP_PRE_TRIG, 20);

XsRefreshCameraSettings(hCam,&xsCfg);

// start recording
XsMemoryStartGrab(hCam,nAddLo,nAddHi,200,20,fCallback,
                                    XS_CF_DONE,NULL);
```

Read the BROC sections information and order the frames.

```
// read an array of 4 section info (200/50 = 4)
XS_BROC_SECTION bArray[4] = {0};
XsGetBrocParameters(m_hCam,&bArray[0],4);

...
```

2.9.3. IRIG/GPS data

IRIG/GPS is supported by any IDT camera model that is equipped with the corresponding module. HG cameras store the IRIG information in the border data structure, other IDT camera store the IRIG info in the frame.

XsReadGPSTiming returns the IRIG/GPS information from the latest read frame or from the camera. The routine fills a XS_GPSTIMING structure passed by the caller.

The field nSignalPresent is usually 0 or 1. In some camera models it may return 4 different values when PTP sync in mode is configured.

0: no PTP detected, timing comes from camera internal clock.

1: PTP detected but no longer present, timing free runs from last PTP sync time.

2: PTP is detected but source is not synchronized with GPS, timing follows PTP.

3: PTP is detected and source is synchronized with GPS, timing follows PTP.

2.9.4. Motorized Lens support

O cameras, Os models, XS PCIe and and some XS-Mini models support motorized lenses (Canon or MFT).

Before configuring any lens parameter, make sure that a motorized lens is mounted on the camera. To do so, read **XSI_LENS_INFO** value. Below a description of the returned information.

- Bits 0 to 3 return the mount type (0: no lens, 1:Canon mount, 2:MFT mount, 4:IDT MFT mount).
- Bit 4 returns whether the lens has variable zoom. Bit is 0 if the lens is prime.
- Bit 5 returns whether the lens has motorized zoom.
- Bit 6 to 31 are reserved for future use.

Once detected a lens and verified if the zoom can be configured, read minimum and the maximum values of lens parameters.

XSP_LENS_FOCUS: units are in pulses. Values can be positive.

XSP_LENS_FOCUS_REL: units are in incremental steps. Values can be positive or negative.

XSP_LENS_IRIS: units are F number multiplied by 100 (example: set 140 to configure F 1.4, set 2200 to configure F22).

XSP_LENS_ZOOM: units are in mm. If a lens is prime, minimum and maximum values correspond. If the lens has manual zoom, the manual value can be read, but not written.

XSP_LENS_CMD: the user can send a command to the lens. Available values for lens command are:

- XS_LCMD_POWEROFF: it powers off the lens.
- XS_LCMD_RESET: it restores the lens after a power off (no need to unplug and plug the lens).

2.9.5. Camera calibration (Background and PSC)

The architecture of the CMOS sensor introduces a noise to the acquired images. To improve image quality a calibration file is produced and distributed with the camera. The file is stored in the camera flash memory and should be downloaded to the hard disk.

The improvement of the image quality is done in two steps:

- **Background**: background images are acquired with the camera body cap on, and then they are subtracted from regular images.

- **Pixel Sensitivity Correction (PSC)**: after background subtraction, a set of coefficients is used to compensate difference of pixel sensitivity in different zones of the sensor.

The image quality may be improved in two ways:

- **Factory calibration file**: the file is stored in the camera flash memory and it may be downloaded. It cannot be cannot be overwritten. from the camera, or copied from the distribution CD. The correction with the factory file is active when the parameters XSP_NOISE_RED (background) and XSP_NOISE_SENS (PSC) are set to 1.

- **Current conditions calibration file**: the user can also acquire background and pixel sensitivity coefficients in current conditions. The values are stored in a local file that can be overwritten and deleted (see below). The corrections with the current condition file are active if the parameters XSP_NOISE_AUTO (background) and XSP_PSC_AUTO (PSC) are set to 1.

The operations on calibration may be done with the **XsCalibrateNoiseReduction** routine and different op-codes.

Calibration in optimal conditions

These operations affect the parameters stored in the factory calibration file, but they **DO NOT MODIFY** the factory calibration file. These options are supported only on cameras that have the calibration file downloaded on the local computer (HS, X and M cameras).

- **XS_C_BKG_ALL**: the camera lens cap must be on. The driver acquires background images in all the conditions and stores them in memory. The factory calibration file is not modified.
- **XS_PSC_FILE_RELOAD**: the driver loads the default camera calibration file and overwrites the background and PSC coefficients stored in memory.

Calibration in current conditions

These operations allow the user to calibrate in current operating conditions. If some of the camera parameters change after the calibration, the calibration should be executed again. The calibration data are stored in a local file that can be reloaded, overwritten or deleted. If a camera is pipeline, the local calibration file is also uploaded to the camera memory and used on-board. If a pipeline camera is powered off, the current calibration data is lost.

- **XS_C_CURRENT_BKG**: the camera lens cap must be on. The driver computes the background images in current operating conditions. The data is applied to the images if the XSP_NOISE_AUTO parameter is 1.

- **XS_C_CURRENT_PSC**: remove the lens and put a constant light in front of the sensor. The driver computes the pixel sensitivity coefficients in current operating conditions. The data is applied if the XSP_NOISE_APSC parameter is 1.
- **XS_C_CURRENT_RESET**: the current calibration local file is deleted and the current coefficients are reset.

Miscellaneous

- **XS_C_FILE_DOWNLOAD**: Giga-Ethernet cameras have on-board flash memory. This option downloads the camera calibration file from flash memory to the hard disk.
- **XS_C_ABORT**: abort any of the above calibration procedures.

2.10. Legacy cameras

2.10.1. Enumerate and Open X cameras (GE)

When an X is powered on, it does not have an IP address. The enumeration returns only a part of the XS_ENUMITEM structure because the driver is not able to establish a connection with the camera. The user has two options:

- Open the camera with the partial structure: the driver automatically assigns an IP address and fills the missing fields.
- Give the camera an IP address: the user may call the **XsPreConfigCamera** with the **XSPP_IP_ADDRESS** parameter key and then enumerate the cameras again.

The sample below shows option 2.

```
XS_ENUMITEM xsl[10];
XSULONG32 nListLen = sizeof(xsl)/sizeof(XS_ENUMITEM);
XSULONG32 nIPAdd = 0x01020304;
XSULONG32 nEnumFlt = XS_EF_GE_X;
XS_HANDLE hCamera;

// Load the driver
XsLoadDriver(0);

// nListLen is the length of your XS_ENUMITEM array
XsEnumCameras( &xsl[0], &nListLen, nEnumFlt );

// check if the first camera has an IP address
if(xsl[0].nLinkType==XS_LT_GIGAETH && xsl[0]. nGeCamIPAdd==0)
{
   XsPreConfigCamera(xsl[0].nCameraID,XSPP_IP_ADDRESS,nIPAdd);
   // check if the first camera has an IP address
   XsEnumCameras( &xsl[0], &nListLen );
}

XsOpenCamera( xsl[0].nCameraId, &hCamera );

// Do something...
...

// Close the camera.
XsCloseCamera( hCamera );

// Unload the driver
XsUnloadDriver();
```

Sometimes the camera cannot be enumerated and the IP address cannot be set. In this condition, if the user knows the camera MAC address, he can try to set the camera IP address with the **XsPreConfigCamera** routine, the **XSPP_IP_ADD_EX** key and the parameters below:

- nCameraID: the pointer to a string that contains the camera MAC address in the format 00-00-00-00-00-00.
- nValueLo: the camera IP address. The parameter that will be configured.
- nValueHi: the pointer to a string that contains the network adapter MAC address in the format 00-00-00-00-00-00.

The sample below shows how to use this option

```
XSULONG32 nIPAdd = 0x01020304;
char szCamMAC[]="00-01-02-03-04-05";
char szAdpMAC[]="00-06-07-08-09-10";

// Load the driver
XsLoadDriver(0);

// try to give the camera an IP address
XsPreConfigCamera((void *)szCamMac,     XSPP_IP_ADD_EX,
                  (void *)nIPAdd, (void *)szAdpMAC);

// Unload the driver
XsUnloadDriver();
```

2.10.2. Asynchronous operations

Asynchronous routines are deprecated and should not be used anymore.

XsQueueOneFrame is the asynchronous grab function. Many frames can be queued up at a time. The maximum number of frames that can be queued is **100**. When the image is acquired the application may receive a callback. The completed frame is removed from the queue, and the next frame takes its place.

XsQueueCameraSettings is an asynchronous function used to change the camera configuration.

The settings are placed on the same queue as your frames queued by **XsQueueOneFrame**. Actions are guaranteed to occur in the order they are queued. If you want to clear the queue, call **XsAbort**. As with **XsQueueOneFrame**, you may receive a callback when the settings have been changed.

2.10.3. N cameras memory management (non pipeline)

In old N cameras the images are stored in 10 bit compressed format. The compression may be configured by the parameter **XSP_CMP_RATIO**. The maximum number of frames that can be captured in a single acquisition is 16,380 (**MAX_N_ACQ_FRAMES**). The XsGetAddressList routine should be used to read the acquired frames. See the example below.

```
unsigned __int64 anAddList[1024];
XSULONG32 i,nSize;

// read the addresses (nFrames is the number of acquired images)
XsGetAddressList(anAddList,0,nFrames+1);

// read the frames
for( i=0; i<nFrames; i++)
{
    // compute the real frame size
    nSize = (XSULONG32)(anAddList[i+1]-anAddList[i]);
    // read the frame
    XsMemoryReadFrame(hCam,(XSULONG32)anAddList[i],
                                    (XSULONG32)(anAddList[i]>>32),
                                    nSize, pDatabuf);
}
```

2.10.4. Trigger and Sync in cameras with two BNC

Some old cameras have only two BNC connectors in the back. A single input connector is used to synchronize the acquisition with an external signal (sync in) or trigger the camera and start the acquisition (event trigger).

The user may read the number of BNC connectors by calling the XsGetCameraInfo API with the parameter XSI_BNC_CONNECTORS.

The table below shows the allowed parameters for 2 BNC.

XSP_REC_MODE	XSP_SYNCIN_CFG	XSP_TRIGIN_CFG
XS_RM_NORMAL	All values (the BNC is used to provide a sync signal)	Ignored (no BNC)
XS_RM_CIRCULAR	All values (the BNC is used to issue a trigger and the camera acquires at internal rate)	Ignored (no BNC)

2.10.5. Plus™ Mode

Plus™ is a compression feature that lets the camera acquire images at double speed and double memory size.

Plus mode may be enabled or disabled by setting the **XSP_PLUS** parameter. A camera supporting Plus mode returns 1 if the **XSI_PLUS** info parameter is read (XsGetCameraInfo).

2.10.6. XDR™ Mode

The **Extended Dynamic Range (XDR)** is an IDT-proprietary implementation that uses a capability of some IDT sensors. In XDR mode the camera dynamic range may be enhanced to 11, 12 or 13 bit.

Imagine that we are taking a picture of a high contrast scene. In the picture we may have a very bright portion (almost saturated) and a very dark portion that is almost black.

How can we change our exposure to have a good image? If we try to increase the exposure to improve the dark part, we saturate the bright one. On the other side, if we decrease to improve the bright part, we reduce the light to the dark portion and we loose information.

The solution is XDR.

With XDR the Y4 camera uses a capability of the sensor that can acquire at two different exposures at the same time: one of the pictures at lower exposure and one at higher exposure.

Then the pictures are linearly combined to obtain a third picture with increased sensitivity. The ratio between higher exposure and lower exposure gives the new pixel depth, which can be 8, 10 or 11 bits for 8 bit images or 10, 11 and 12 bits for 10 bit images.

The parameters for the activation of XDR are:

XSP_EXP_MODE: set the parameter to the **XS_EM_XDR** value.

XSP_XDR_RATIO: set the ratio between exposures. The ratio determines the extended pixel depth like in the table below.

XDR Ratio	Extended Pixel depth
2	11 bit
4	12 bit
8	13 bit

XSP_PIX_DEPTH, XSP_IMG_FORMAT: when XDR is active, change the format to generate an image with extended pixel depth.

XSP_XDR_CONTRAST: if the pixel depth is 8, the extended range image is converted to 8 bit with a logarithmic Look Up table.

3. SDK Reference

3.1. Initialization Functions

3.1.1. Overview: Initialization functions

Initialization functions allow the user to initialize the camera, enumerate the available cameras, open and close them.

XsGetVersion returns the SDK version number (64 bit) and the demo flag.

XsLoadDriver loads the driver and initializes it.

XsUnloadDriver unloads the driver.

XsEnumCameras enumerates the cameras connected to the computer.

XsPreConfigCamera configures a camera parameter before opening it.

XsOpenCamera opens a camera.

XsOpenRawcamera opens a raw sequence like a virtual camera.

XsCloseCamera closes a camera previously open.

3.1.2. XsGetVersion

XS_ERROR XsGetVersion (**PXSULONG32** **pVersionMS*, **PXSULONG32** ***pVersionLS*, **PXSULONG32** ***pIsDemo*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_GENERIC_ERROR if the version numbers could not be extracted from the driver.

Parameters

pVersionMS

Specifies the pointer to the variable that receives the most significant 32 bit of the version.

pVerMinor

Specifies the pointer to the variable that receives the least significant 32 bit of the version

pIsDemo

Specifies the pointer to the variable that receives the demo flag; If 1, the driver is demo, if 0 it isn't.

Remarks

This function must be called to retrieve the SDK version number and demo flag. If the demo flag is returned TRUE, the currently installed driver does not require the presence of the camera to operate. The MS and LS version fields contain an upper 16 bit word and a lower one. The version is the made of four numbers.

See also:

3.1.3. XsLoadDriver

XS_ERROR XsLoadDriver (**unsigned short** *nUSBNotify*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_HARDWARE_FAULT if any error occurs during the initialization.

Parameters

nUSBNotify

The parameter activates and deactivates the notification of disconnection of the USB cable.

Remarks

The routine loads the driver DLL and initializes it. It must be called before any other routine, except **XsGetVersion**. If any error occurs, the routine returns XS_E_HARDWARE_FAULT. The user may retrieve the hardware error code by calling the **XsGetHardwareError** routine.

If the *nUSbNotify* parameter is set to 1, the user receives announcements when the USB cable of any Y or X camera is unplugged. To do so, he has to install a callback by calling the XsSetAnnouncementCallback routine. For more information about installing the announcement callback see the routine description and the Appendix E for a complete list of announcements.

See also: **XsUnloadDriver**, **XsGetHardwareError, XsSetAnnouncementCallback**

3.1.4. XsUnloadDriver

void XsUnloadDriver (void)

Return values

None

Parameters

None

Remarks

This function must be called before terminating the application. This function frees any memory and resource allocated by the driver and unloads it.

See also: **XsLoadDriver**

3.1.5. XsEnumCameras

XS_ERROR XsEnumCameras (**PXS_ENUMITEM** *pItemList*, **XSULONG32** ***pItemNr,* **XSULONG32** *nEnumFlt*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_HARDWARE_FAULT if any error occurs during the cameras enumeration.

XS_E_INVALID_ARGUMENTS, if any of the parameters is not valid.

Parameters

pItemList

Specifies the pointer to an array of XS_ENUMITEM structures

pItemNr

Specifies the pointer to the variable that receives the number of detected cameras

nEnumFlt

Specifies the enumeration filter

Remarks

The routine enumerates the active cameras and fills the **XS_ENUMITEM** structures with information about them. This routine must be called before **XsOpenCamera** to find out which cameras are available. The pItemNr variable must specify the number of structures in the pItemList array and receives the number of enumerated cameras. The nEnumFlt variable specifies which camera type is going to be enumerated. If any error occurs, the routine returns XS_E_HARDWARE_FAULT. Then the user may retrieve the hardware error code by calling the **XsGetHardwareError** routine.

See also: **XsOpenCamera**, **XsGetHardwareError**

3.1.6. XsPreConfigCamera

XS_ERROR XsPreConfigCamera (**void** ***nCameraId**, **XS_PRE_PARAM** *nParamKey,* **void** ***nValueLo,* **void** ***nValueHi*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_CAMERA_ID, if the camera ID is not valid.

XS_E_CAM_ALREADY_OPEN, if the camera is open.

XS_E_HARDWARE_FAULT if any error occurs during the camera pre-configuration.

XS_E_NOT_SUPPORTED, if the parameter is not supported.

Parameters

nCameraId

Specifies the ID of the camera to be opened

nParamKey

Specifies which parameter is to be configured

nValueLo

Specifies the LS part of the parameter

nValueHi

Specifies the MS part of the parameter

Remarks

The routine configure a camera parameter before the camera is open. This routine must be called before **XsOpenCamera** to set a parameter that is important for the connection. The nParamKey specifies which parameter to configure. Some parameters are useful for all the cameras and the camera ID value is ignored. For Giga-Ethernet cameras the routine is used to configure the network adapter IP address, the camera IP address or the network performance. If any error occurs, the routine returns XS_E_HARDWARE_FAULT. Then the user may retrieve the hardware error code by calling the **XsGetHardwareError** routine. For a list of the pre-configuration parameter indexes, refer to the appendix.

See also: **XsGetHardwareError**

3.1.7. XsOpenCamera

XS_ERROR XsOpenCamera (**XSULONG32** *nCameraId*, **XS_HANDLE*** *pHandle*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_CAMERA_ID, if the camera ID is not valid.

XS_E_HARDWARE_FAULT if any error occurs during the camera opening.

Parameters

nCameraId

Specifies the ID of the camera to be opened

pHandle

Specifies the pointer to the variable that receives the camera handle

Remarks

The routine opens the camera with the *nCameraId* ID. The value can be retrieved by calling the **XsEnumCameras** routine (see the XS_ENUMITEM structure). If any error occurs during the camera opening, the routine returns XS_E_HARDWARE_FAULT. Then the user may retrieve the hardware error code by calling the **XsGetHardwareError** routine.

See also: **XsCloseCamera**, **XsGetHardwareError**

3.1.8. XsOpenRawCamera

XS_ERROR XsOpenRawCamera (**const char *** *lpszRawFilePath*, **XS_HANDLE*** *pHandle*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_CAMERA_ID, if the raw path is not valid.

XS_E_HARDWARE_FAULT if any error occurs during the camera opening.

Parameters

lpszRawFilePath

Specifies the full path to the raw file

pHandle

Specifies the pointer to the variable that receives the virtual camera handle

Remarks

The routine opens the RAW file with path *lpszRawFilePath*. The variable may contain be the full path to the rawfile.xml file or the full path to the directory that includes the file and the raw sequence.

See also: **XsCloseCamera**

3.1.9. XsCloseCamera

XS_ERROR XsCloseCamera (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

Closes an open Camera

See also: **XsOpenCamera**

3.2. Configuration Functions

3.2.1. Overview: Configuration functions

The configuration functions allow the user to control the status of the camera.

XsGetCameraInfo gets information from the camera, such as camera model, firmware version, sensor type, sensor model, etc.

XsSetCameraInfo sets information to the camera, such as camera name.

XsReadDefaultSettings reads default settings from the camera and fills the XS_SETTINGS opaque structure.

XsReadCameraSettings reads current settings from the camera and fills the XS_SETTINGS opaque structure.

XsRefreshCameraSettings sends an updated XS_SETTINGS structure to the camera and refreshes the camera settings.

XsValidateCameraSettings validates and updates a camera state.

XsReadSettingsFromFlash reads the camera settings from the onboard flash memory.

XsWriteSettingsToFlash writes the camera settings to the onboard flash memory.

XsQueueCameraSettings queues camera settings.

XsSetParameter sets one of the camera parameters in the XS_SETTINGS opaque structure.

XsGetParameter gets one of the parameters from the XS_SETTINGS opaque structure.

XsGetParameterAttribute gets a parameter's attribute, such as minimum value, maximum value, etc.

XsCalibrateNoiseReduction computes the reference image data used to reduce the acquisition noise.

XsReset resets the camera.

XsReadUserDataFromFlash reads a block of user data from the camera flash memory.

XsWriteUserDataToFlash writes a block of user data to the camera flash memory.

XsReadCameraSettingsArray reads an array of camera configurations from the DDR or SSD.

3.2.2. XsGetCameraInfo

XS_ERROR XsGetCameraInfo (**XS_HANDLE** *hCamera*, **XS_INFO** *nInfoKey*, **XSULONG32 ***pValueLo,* **XSULONG32 ***pValueHi*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_NOT_SUPPORTED, if the nInfoKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera

nInfoKey

Specifies which parameter the function has to return

pValueLo

Specifies the pointer to the variable that receives the LS part of the info value

pValueHi

Specifies the pointer to the variable that receives the MS part of the info value

Remarks

This function returns camera specific information, such as sensor type or version numbers, generally state-independent information. See the **Appendix B** for a list of all the available nInfoKey values.

See also: **XsSetCameraInfo**

3.2.3. XsSetCameraInfo

XS_ERROR XsSetCameraInfo (**XS_HANDLE** *hCamera*, **XS_INFO** *nInfoKey*, **XSULONG32** *nValueLo,* **XSULONG32** *nValueHi*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_NOT_SUPPORTED, if the nInfoKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera

nInfoKey

Specifies which parameter the function has to return

nValueLo

Specifies the LS part of the info value

nValueHi

Specifies the MS part of the info value

Remarks

This function sets camera specific information. Some of the info parameters can be changed, such as camera name. See the **Appendix B** for a list of all the available nInfoKey values.

See also: **XsGetCameraInfo**

3.2.4. XsReadDefaultSettings

XS_ERROR XsReadDefaultSettings (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure to be filled with the camera settings

Remarks

This function reads the default settings of the specified camera and fills the XS_SETTINGS structure. The structure is opaque and can be accessed only through the XsGetParameter and XsSetParameter functions. To change a parameter on the camera, the entire structure must be sent to the driver, using the XsRefreshCameraSettings function. The default state is specific to each individual camera.

See also: **XsGetParameter**, **XsSetParameter**, and **XsRefreshCameraSettings**

3.2.5. XsReadCameraSettings

XS_ERROR XsReadCameraSettings (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure to be filled with the camera settings

Remarks

This function reads the current settings of the specified camera and fills the XS_SETTINGS structure. The structure is opaque and can be accessed only through the XsGetParameter and XsSetParameter functions. To change a parameter on the camera, the entire structure must be sent to the driver, using the XsRefreshCameraSettings function.

See also: **XsGetParameter**, **XsSetParameter**, **XsRefreshCameraSettings**

3.2.6. XsRefreshCameraSettings

XS_ERROR XsRefreshCameraSettings(**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure that contains the camera settings

Remarks

The state contained in the XS_SETTINGS structure is validated, modified if necessary, and then sent to the camera. The structure is opaque and can be accessed only through the XsGetParameter and XsSetParameter functions.

See also: **XsReadDefaultSettings**, **XsReadCameraSettings**

3.2.7. XsValidateCameraSettings

XS_ERROR XsValidateCameraSettings (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure that contains the camera settings

Remarks

The state contained in the XS_SETTINGS structure is validated and modified if necessary.

See also: XsReadDefaultSettings, XsReadCameraSettings

3.2.8. XsReadSettingsFromFlash

XS_ERROR XsReadSettingsFromFlash (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_NOT_SUPPORTED, if the camera does not have flash memory

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

XS_E_NOT_IN_FLASH, if the configuration is not stored in the flash memory

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure that contains the camera settings

Remarks

The routine copies the content of the pSettings structure to the camera flash memory.

See also: XsWriteSettingsToFlash

3.2.9. XsWriteSettingsToFlash

XS_ERROR XsWriteSettingsToFlash (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_NOT_SUPPORTED, if the camera does not have flash memory

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure that contains the camera settings

Remarks

The routine reads from the camera flash memory the configuration and copies it to the pSettings structure.

See also: XsReadSettingsFromFlash

3.2.10. XsQueueCameraSettings

XS_ERROR XsQueueCameraSettings (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*, **XS_AsyncCallback** *pfnCallback*, **XSULONG32** *nFlags*, **void** **pUserData*)

Return values

SVC_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the structure that contains the camera settings

pfnCallback

Specifies the pointer to the callback routine; the routine is called by the driver when the settings are changed. See XS_AsyncCallback.

nFlags

Specifies the flags; see Appendix D

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

This function queues up a change to the camera state. This function returns immediately. When the camera state has changed, you will receive a callback if desired.

See also: XsRefreshCameraSettings

3.2.11. XsSetParameter

XS_ERROR XsSetParameter (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*, **XS_PARAM** *nParamKey*, **XSULONG32** *nValue*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

XS_E_NOT_SUPPORTED, if the nParamKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the XS_SETTINGS structure the parameter is written to.

nParamKey

Specifies which parameter the function sets.

nValue

Specifies the parameter's value

Remarks

This function writes a parameter to the opaque XS_SETTINGS structure. The parameter will not change on the camera until the entire structure is sent to the driver by calling the XsRefreshCameraSettings or XsQueueCameraSettings functions.

See also: XsGetParameter, XsRefreshCameraSettings, and XsQueueCameraSettings

3.2.12. XsGetParameter

XS_ERROR XsGetParameter (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*, **XS_PARAM** *nParamKey*, **XSULONG32** ***pValue*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

XS_E_NOT_SUPPORTED, if the nParamKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the XS_SETTINGS structure the parameter is read from

nParamKey

Specifies which parameter the function returns

pValue

Specifies the pointer to the parameter's value

Remarks

This function reads a parameter from the opaque XS_SETTINGS structure.

See also: XsSetParameter

3.2.13. XsGetParameterAttribute

XS_ERROR XsGetParameterAttribute (**XS_HANDLE** *hCamera*, **PXS_SETTINGS** *pSettings*, **XS_PARAM** *nParamKey*, **XS_ATTRIBUTE** *nParamAttr*, **XSULONG32 ***pValue*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_INVALID_CFG, if the XS_SETTINGS structure is not valid.

XS_E_NOT_SUPPORTED, if the nParamKey is not supported.

Parameters

hCamera

Specifies the handle to an open camera

pSettings

Specifies the pointer to the XS_SETTINGS structure the parameter is read from.

nParamKey

Specifies which parameter the function returns.

nParamAttr

Specifies which attribute the function returns.

pValue

Specifies the pointer to the parameter's attribute value.

Remarks

This function reads a parameter attribute depending on the nParamAttr value. It may be: minimum value, maximum value or read-only attribute (see Appendix D).

See also: XsGetParameter

3.2.14. XsCalibrateNoiseReduction

XS_ERROR XsCalibrateNoiseReduction (**XS_HANDLE** *hCamera,* **XSULONG32** *nOpCode,* **XS_ProgressCallback** *pfnCallback,* **void** * *pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_CAMERA_ID, if the camera ID is not valid.

XS_E_NOT_SUPPORTED, if the specified operation is not supported.

XS_E_NOT_IN_FLASH, if the calibration file is not stored in the camera flash memory.

XS_E_ABORTED, if the procedure has been aborted.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nOpCode

Specifies the calibration operation to do

pfnCallback

Specifies the pointer to the callback routine; the routine is called by the driver during the calibration operation. See the XS_ProgressCallback in the Appendix

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

This routine computes the reference image data used to reduce the noise on acquired images. The calibration operation depends on the value of the nOpCode parameter. For further information, refer to the paragraph 2.8.

See also:

3.2.15. XsReset

XS_ERROR XsReset (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

This routine resets the camera. The camera is reset and automatically re-configured with the current parameters.

See also:

3.2.16. XsReadUserDataFromFlash

XS_ERROR XsReadUserDataFromFlash (**XS_HANDLE** *hCamera,* **XSULONG32** *nType,* **XSULONG32** *nDataIDOrOffset,* **XSULONG32** ***pnSize,* **void** **pDataBuff*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

XS_E_NOT_IN_FLASH, if the data is not stored in the flash memory.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nType

Specifies the type of operation (0: read from flash, 1: read from memory)

nDataIDOrOffset

Specifies the unique ID that identifies the data for flash operation, or the offset in 2048 bytes blocks for memory operation

pnSize

Specifies the pointer to the variable that receives the data size

pDataBuff

Specifies the buffer that receives the stored data

Remarks

This routine reads a buffer of user data from the camera flash or RAM memory. The unique ID is a number that identifies the data block. The Offset is specified in number of blocks (each block is 2048 bytes).

See also: XsWriteUserDataToFlash

3.2.17. XsWriteUserDataToFlash

XS_ERROR XsWriteUserDataToFlash (**XS_HANDLE** *hCamera,* **XSULONG32** *nType,* **XSULONG32** *nDataIDOrOffset,* **XSULONG32** *nSize,* **void** ***pDataBuff**)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nType

Specifies the type of operation (0: write to flash, 1: write to memory)

nDataIDOrOffset

Specifies the unique ID that identifies the data for flash operation, or the offset in 2048 bytes blocks for memory operation

nSize

Specifies the size of the data block

pDataBuff

Specifies the buffer that contains the data to store

Remarks

This routine writes a buffer of user data to the camera flash or RAM memory. The unique ID is a number that identifies the data block. The Offset is specified in number of blocks (each block is 2048 bytes).

See also: XsReadUserDataFromFlash

3.2.18. XsReadCameraSettingsArray

XS_ERROR XsReadCameraSettingsArray (**XS_HANDLE** *hCamera,* **XSULONG32** *nOption*, **PXS_SETTINGS** *pCfgList,* **PXS_BROC** *pBrocList,* **PXSULONG32** *pnCfgCnt*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

Parameters

hCamera

Specifies the handle to an open camera

nOption

Specifies the source (0: camera memory 1: SSD)

pCfgList

Specifies an array of camera configuration structures

pBrocList

Specifies an array XS_BROC structures (it may be null)

pnCfgCnt

Specifies the pointer to the variable that receives the number of configurations stored in the SSD

Remarks

This routine reads an array of camera configurations stored in the camera DDR (if the option parameter is 0) or in the SSD (if the option parameter is 1). Each configuration corresponds to a set of images stored in the DDR/SSD. The variable pnCfgCnt is a pointer to a variable that has a double meaning. Before calling the routine the user should set the value equal to the number of items in the array. When the routine is returned, the variable contains the number of structures stored in the DDR/SSD. The information stored in the array may be used to read the images from the camera and download them to the local hard disk. If the images have been recorded in BROC mode, the corresponding XS_BROC section may be used to sort the images.

See also: XsMemoryReadFromDisk, XsEraseDisk

3.3. Preview Mode Grab Functions

3.3.1. Overview: Preview Mode Grab functions

Grab functions allow the user to capture streamed data from the digital camera.

The grab process may be performed in two ways:

- **Synchronous**: calling XsSynchGrab function.
- **Asynchronous**: calling XsQueueOneFrame function.

Both methods use the XS_FRAME structure to grab the data.

XsSynchGrab grabs one frame synchronously (two in double exposure).

XsQueueOneFrame grabs one frame asynchronously (two in double exposure).

XsLive starts and stops fast live on Os cameras.

XsAbort aborts any pending asynchronous grab.

3.3.2. XsSynchGrab

XS_ERROR XsSynchGrab (**XS_HANDLE** *hCamera*, **PXS_FRAME** *pFrame*, **XSULONG32** *nTimeOut*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_BUFFER_TOO_SMALL, if the frame buffer is too small for the image.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

XS_E_TIMEOUT, if the frames have not been acquired within the time out value.

Parameters

hCamera

Specifies the handle to an open camera

pFrame

Specifies the pointer to a XS_FRAME structure; the structure is used to acquire the frame

nTimeOut

Specifies the grab time out in ms

Remarks

This function grabs synchronously one frame (or two in double exposure mode). Before calling the routine the user must fill some of the XS_FRAME structure fields (**pBuffer**: the pointer to the data, **nBufSize**: the size of the data buffer in bytes, **nImages**: the number of images, e.g. 1 in single exposure or 2 in double exposure). The routine returns when the frames have been acquired or after the timeout (synchronous grab).

See also: XsQueueOneFrame

3.3.3. XsQueueOneFrame (deprecated)

XS_ERROR XsQueueOneFrame (**XS_HANDLE** *hCamera*, **PXS_FRAME** *pFrame*, **XS_AsyncCallback** *pfnCallback*, **XSULONG32** *nFlags*, **void** ***pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_BUFFER_TOO_SMALL, if the frame buffer is too small for the image.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

pFrame

Specifies the pointer to the frame structure

pfnCallback

Specifies the pointer to the callback routine; the routine is called by the driver when the settings are changed. See the XS_AsyncCallback in the Appendix

nFlags

Specifies the flags; see Appendix

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

This functions queues a frame buffer and returns immediately. It's used for asynchronous acquisitions. Before calling the routine the user must fill some of the XS_FRAME structure fields (see XsSynchGrab routine). When the frame has been captured the *pfnCallback* routine is called. The frame structure and the associated data buffer must persist until the frame has been grabbed.

See also: XsSynchGrab

3.3.4. XsLive

XS_ERROR XsLive (**XS_HANDLE** *hCamera,* **XS_LIVE** *nCmd*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

XS_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nCmd

Specifies the live command

Remarks

This routine starts or stops the fast live mode in Os cameras. If the nCmd variable is set to XS_LIVE_START the fast live is enabled. Then the user may read the live image by calling the XsMemoryPreview routine. Then the live mode should be closed with nCmd set to XS_LIVE_STOP. If the live mode is not closed the camera cannot start a recording.

See also:

3.3.5. XsAbort

XS_ERROR XsAbort (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

This function stops all the pending grab operations and clears the queue. After the function has returned no more XsQueueOneFrame or XsQueueCameraSettings callbacks occur.

See also: XsQueueCameraSettings, XsQueueOneFrame

3.4. Camera Memory Grab Functions

3.4.1. Overview: Camera Memory Mode Grab functions

Camera Memory Grab functions allow the user to capture data into the camera RAM memory, check the capture status and red the captured data into the PC memory.

XsMemoryStartGrab starts an acquisition in the camera memory.

XsMemoryStopGrab stops an acquisition in the camera memory.

XsMemoryPreview reads the latest acquired frame during an acquisition and/or reads the number of frames acquired so far.

XsMemoryReadFrame reads a frame from the camera memory.

XsMemoryReadTriggerPosition checks the acquisition triggered frame.

XsMemoryDownloadRawFrame downloads an image into a RAW file.

XsGetAddressList gets a list of addresses of the acquired frames (N-series).

XsEraseMemory erases the memory in the HG camera models.

XsTrigger issues a software trigger to the camera.

XsGetBrocParameters reads information about the current BROC sections.

XsMemoryReadFromDisk reads images form the SSD into the camera memory.

XsEraseDisk erases the images from the SSD.

3.4.2. XsMemoryStartGrab

XS_ERROR XsMemoryStartGrab (**XS_HANDLE** *hCamera*, **XSULONG32** *nStartAddLo*, **XSULONG32** *nStartAddHi*, **XSULONG32** *nFrames*, **XSULONG32** *nPreTrigFrames*, **XS_AsyncCallback** *pfnCallback*, **XSULONG32** *nFlags*, **void** ***pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_BUSY, if the camera is busy and the command cannot be performed.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrames

Specifies the number of frames which have to be acquired

nPreTrigFrames

Specifies the number of frames to be acquired before the trigger; it's valid only if the trigger source is a single pulse.

pfnCallback

Specifies the pointer to the callback routine; the routine is called by the driver when the acquisition is completed or any error occurred. See the XS_AsyncCallback in the Appendix.

nFlags

Specifies the flags; see Appendix.

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

This function starts an acquisition in the camera memory and returns immediately. When the frames have been captured or any error occurred the *pfnCallback* routine is called.

See also: XsMemoryStopGrab

3.4.3. XsMemoryStopGrab

XS_ERROR XsMemoryStopGrab (**XS_HANDLE** *hCamera,* **XSULONG32** **pnAcqFrames*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera.

pnAcqFrames

Specifies the pointer to the variable that receives the number of frames acquired so far.

Remarks

This function stops any camera memory acquisition previously started. The routine returns the number of frames that the camera has acquired after the stop. The value is stored in the variable pointed by pnAcqFrames.

See also: XsMemoryStartGrab

3.4.4. XsMemoryPreview

XS_ERROR XsMemoryPreview (**XS_HANDLE** *hCamera*, **PXS_FRAME** *pFrame*, **XSULONG32 ***pnFrameIndex*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if the routine arguments are not valid.

XS_E_BUSY, if the camera is busy and the command cannot be performed.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

pFrame

Specifies the pointer to the frame structure

pnFrameIndex

Specifies the pointer to the variable which receives the index of the latest acquired frame.

Remarks

This routine may be called during an acquisition in camera memory and it may do two operations: read the latest acquired frame into a XS_FRAME structure or read the number of frames acquired so far. One of the two additional parameters (pFrame or pnFrameIndex) may be NULL. The routine may be called to preview an acquisition. For further information about the XS_FRAME structure, see the XsSynchGrab topic.

3.4.5. XsMemoryReadFrame

XS_ERROR XsMemoryReadFrame (**XS_HANDLE** *hCamera*, **XSULONG32** *nStartAddLo*, **XSULONG32** *nStartAddHi*, **XSULONG32** *nFrameIdxOrSize*, **void*** *pDataBuff*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_BUSY, if the camera is busy and the command cannot be performed.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrameIdxOrSize

Specifies the index or the size of the frame to read

pDataBuff

Specifies the pointer to the buffer where the data has to be copied.

Remarks

This function reads a single frame from the camera memory into the specified buffer. The user must specify the starting address and the index of the frame. If a sequence of N frames has been acquired starting from address M, all the frames can be read by calling the routine with the same start address (M) and with index values from 0 to N-1. The driver uses the current camera settings to compute the frame size. The user must be sure that the current camera settings (Image format, pixel depth, etc.) are the same set before the acquisition.
For N cameras the user specifies the address of the frame (after a call to **XsGetAddressList**) and its size. If the user specifies 0 as size, the driver computes the size of the frame from the current settings.
For further information, please refer to “Multiple Acquisitions” and “Camera Memory Management” topics.

See also: XsMemoryStartGrab, XsMemoryStopGrab, XsGetAddressList

3.4.6. XsMemoryDownloadRawFrame

XS_ERROR XsMemoryDownloadRawFrame (**XS_HANDLE** *hCamera*, **const char** **lpszRawFilePath*, **XSULONG32** *nStartAddLo*, **XSULONG32** *nStartAddHi*, **XSULONG32** *nFrameIdx*, **XSULONG32** *nPageIdx,* **XSULONG32** *nTotFrames*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_BUSY, if the camera is busy and the command cannot be performed.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera.

lpszRawFilePath

Specifies the full path of the RAW file.

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address.

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address.

nFrameIdx

Specifies the index of the frame in camera memory.

nPageIdx

Specifies the index of the frame in the sequence (0 to nTotFrames).

nTotFrames

Specifies the total number of downloaded frames.

Remarks

This function downloads a frame from the camera memory into the specified Raw file. The full path of the Raw file may be specified without the extension because the driver will ad a “.raw” extension to it. The user must specify the starting address and the index of the frame in camera memory. If a sequence of N frames has been acquired in circular mode, the position of the trigger index (T) should be read and the frames indexes should be ordered (see the example in chapter 2 “Using the SDK”). Also, the pages index (from 0 to N-1) and the total number of frames (N) must be specified.

See also: XsMemoryStartGrab, XsMemoryStopGrab

3.4.7. XsMemoryReadTriggerPosition

XS_ERROR XsMemoryReadTriggerPosition (**XS_HANDLE** *hCamera,* **XSULONG32*** *pnTriggerPosLo*, **XSULONG32*** *pnTriggerPosHi*, **XSULONG32*** *pnTriggerIndex*, **XSULONG32*** *pnTriggerTime*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

pnTriggerPosLo

Specifies the pointer to the variable which receives the low order 32 bit value of the address of the triggered frame in the sequence

pnTriggerPosHi

Specifies the pointer to the variable which receives the high order 32 bit value of the address of the triggered frame in the sequence

pnTriggerIndex

Specifies the pointer to the variable which receives the index of the triggered frame in the sequence

pnTriggerTime

Specifies the pointer to the variable which receives the time between the sync and the trigger in the acquisition period

Remarks

This function is valid only if the record mode parameter has been set to XS_RM_CIRCULAR. The returned value is a 64 bit address in the camera memory and an index in the acquired sequence. The returned values are valid until a new acquisition or snap API is called.

See also: XsMemoryStartGrab

3.4.8. XsGetAddressList (N-series)

XS_ERROR XsGetAddressList (**XS_HANDLE** *hCamera,* **XSULONG32** *nStartIdx*, **XSULONG32** *nAddressCount*, **unsigned __int64*** *pnAddrList*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nStartIdx

Specifies the index of the first frame of the list

nAddressCount

Specifies the number of addresses to read

pnAddrList

Specifies the pointer to a buffer of 64-bit unsigned integers that will receive the addresses

Remarks

This function is valid only if the camera is an **N-series camera**. The returned values are 64 bit addresses in the camera memory linear space. The frames are compressed and the size of each frame is non constant. After each acquisition the user should call XsGetAddressList and use the list to read each acquired frame.

See also:

3.4.9. XsEraseMemory

XS_ERROR XsEraseMemory (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

The routine is called to erase the memory of HG cameras. If the memory is not erased the user cannot start a new acquisition.

See also: XsMemoryStartGrab

3.4.10. XsTrigger

XS_ERROR XsTrigger (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the function is not supported.

XS_E_NOT_RECORDING, if the camera is not in recording.

XS_E_BUSY, if the camera is busy.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

The routine is called to issue a software event trigger to the camera. It can be called when the camera is recording in circular mode.

See also: XsMemoryStartGrab

3.4.11. XsGetBrocParameters

XS_ERROR XsGetBrocParameters(**XS_HANDLE** *hCamera,* **PXS_BROC_SECTION** *pBrocSectArray,* **XSULONG32** *nSize*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMANTS, , if the routine arguments are not valid.

XS_E_NOT_SUPPORTED, if the function is not supported.

Parameters

hCamera

Specifies the handle to an open camera.

PXS_BROC_SECTION

Specifies the pointer to n array of structures.

nSize

Specifies the number of items in the array.

Remarks

The routine is called to read addresses and frames positions after a BROC acquisition. The information of each segment is stored in a **XS_BROC_SECTION** structure and includes starting address, position of the first frame and time from trigger.

See also: XsMemoryStartGrab

3.4.12. XsMemoryReadFromDisk

XS_ERROR XsMemoryReadFromDisk (**XS_HANDLE** *hCamera,* **XSULONG32** *nMemDstAddLo*, **XSULONG32** *nMemDstAddHi*, **XSULONG32** *nDiskSrcAddLo*, **XSULONG32** *nDiskSrcAddHi*, **XSULONG32** *nStartIdx*, **XSULONG32** *nStopIdx*, **XS_ProgressCallback** *pfnCallback*, **void** ***pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if the routine arguments are not valid.

XS_E_NOT_SUPPORTED, if the function is not supported.

Parameters

hCamera

Specifies the handle to an open camera.

nMemDstAddLo, nMemDstAddHi

Specifies the address in camera memory where the images should be copied.

nDiskSrcAddLo, nDiskSrcAddHi

Specifies the address in the SSD from which the images should be copied.

nStartIdx, nStopIdx

Specifies the start and stop indexes of the images to copy.

pfnCallback

Specifies a pointer to the callback routine that will be called from progress

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

The routine transfers data from the SSD into the camera memory. The address in the camera memory may be always 0. The address in the SSD space is where the images are stored. The number of transferred frames must fit into the DDR. If the full acquisition does not fit into the DDR the number of frames (nStartIdx - nStopIdx + 1) should be a multiple of 256.

See also:

3.4.13. XsEraseDisk

XS_ERROR XsEraseDisk (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the function is not supported.

Parameters

hCamera

Specifies the handle to an open camera.

Remarks

The routine erases all the images stored in the camera SSD.

See also:

3.5. Miscellaneous Functions

3.5.1. Overview: Miscellaneous functions

Miscellaneous functions allow the user to read hardware error codes and strings.

XsGetHardwareError reads the hardware error code and returns the error string related to the code.

XsReadGPSTiming read the IRIG/GPS data from the camera.

XsEnableDiagnosticTrace enables or disables the diagnostic trace.

XsEnableRawMode sets the camera image format to grayscale and allows reading the Bayer raw data from color cameras.

XsGetCameraStatus reads the camera status to check if the acquisition is done.

XsSetAnnouncementCallback installs an announcement callback routine.

XsReadBorderData reads border data from the camera.

XsAttach attaches to a camera and takes the control of it

XsConfigureWriteToDisk activates and deactivates direct write to disk with M, Pcie and XS-MIni cameras.

XsReadToVideo sends live images or playbacks acquired frames to the HDMI output (new design cameras only).

XsLoadLookupTable loads a user-define lookup table.

XsEnableResize has been removed because deprecated.

XsVideoPlayback activates and deactivates the asynchronous playback of images on the video (HDMI) output. The routine returns immediately. The status of the playback may be checked with a call to XsGetCameraStatus.

3.5.2. XsGetHardwareError

XSULONG32 XsGetHardwareError (**XS_HANDLE** *hCamera,* **char*** *pszBuffer*, **XSULONG32** *nSize*)

Return values

The latest hardware error code

Parameters

hCamera

Specifies the handle to an open camera

pszBuffer

Specifies the character buffer which receives the error string

nSize

Specifies the size in bytes of the char buffer

Remarks

If any of the driver's API returns XS_E_HARDWARE_FAULT, the hardware related error may be retrieved by calling XsGetHardwareError. The function returns the hardware error occurred after the latest camera operation. Also, the function fills the pszBuffer buffer with a message that describes the returned error code.

See also:

3.5.3. XsReadGPSTiming

XS_ERROR XsReadGPSTiming (**XS_HANDLE** *hCamera,* **XSULONG32** *nOption,* **PXS_GPSTIMING** *pGPS*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the option is not supported

Parameters

hCamera

Specifies the handle to an open camera

nOption

Specifies from which source the data is read (0: from the current frame, 1:from the camera)

pGPS

Specifies the pointer to the XS_GPSTIMING structure that stores the data

Remarks

This routine reads the IRIG/GPS data from the current frame or from the camera. The current frame is the latest frame that has been read from the camera. The strucutre returns timing information and it can alos return if the signal is currently locked. The IRIG/GPS capability is active if the **XSP_SYNCIN_CFG** parameter is set to XS_SIC_IRIG_DTS_INT, XS_SIC_IRIG_DTS_EXT, XS_SIC_1PPS.

See also:

3.5.4. XsEnableDiagnosticTrace

XS_ERROR XsEnableDiagnosticTrace (**XS_HANDLE** *hCamera,* **char*** *pszTraceFilePath*, **XSULONG32** *nEnable*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pszTraceFilePath

Specifies the path of the diagnostic file

nEnable

Specifies the enable/disable flag

Remarks

The routine enables or disables the driver diagnostic trace. The pszTraceFilePath specifies the path of the diagnostic file and the nEnable flag enables (1) or disables (0) the trace.

See also:

3.5.5. XsEnableRawMode

XS_ERROR XsEnableRawMode (**XS_HANDLE** *hCamera,* **XSULONG32** *nEnable*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

nEnable

Specifies the enable/disable flag

Remarks

The routine enables or disables the camera raw mode in color cameras. If the raw mode is enabled, the image format is automatically set to gray-scale and the user can read the raw Bayer frame from the camera memory.

See also:

3.5.6. XsGetCameraStatus

XS_ERROR XsGetCameraStatus (**XS_HANDLE** *hCamera,* **XSULONG32** **pnIsBusy*, **XSULONG32** **pnStatus,* **XSULONG32** **pnErrCode,* **XSULONG32** **pnInfo1,* **XSULONG32** **pnInfo2,* **XSULONG32** **pnInfo3*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pnIsBusy

Specifies the pointer to the variable that receives the busy flag

pnStatus

Specifies the pointer to the variable that receives the camera status

pnErrCode

Specifies the pointer to the variable that receives the error code

pnInfo1, pnInfo2, pnInfo3

Specify the pointers to the variables that receive info parameters

Remarks

The routine reads the camera busy flag and the status. The camera status values are listed in the XS_STATUS constants. The routine may be used during an acquisition to check if the camera has finished acquiring.

See also:

3.5.7. XsSetAnnouncementCallback

XS_ERROR XsSetAnnouncementCallback (**XS_HANDLE** *hCamera,* **XS_AnnouncementCallback** *pfnCallback*, **void** ***pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pfnCallback

Specifies the pointer to the callback routine; the routine is called by the driver any time a camera changes its status. For more info, see the XS_AnnouncementCallback topic in the Appendix G.

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

Remarks

The routine defines an announcement callback for the cameras. If the camera status changes, the camera autonomously sends messages called announcements. The callback intercepts those announcements. For a detailed description of announcements please refer to the Appendix E.

See also:

3.5.8. XsReadBorderData (HG)

XS_ERROR XsReadBorderData (**XS_HANDLE** *hCamera,* **void** ***pDataBuff**, **XSULONG32** *nSize*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

Parameters

hCamera

Specifies the handle to an open camera

pDataBuff

Specifies the bytes buffer which receives the border data

nSize

Specifies the size in bytes of the data buffer

Remarks

The routine reads the border data from the HG camera. For a detailed description of the border data structure please refer to the HG Command Protocol reference.

See also:

3.5.9. XsAttach

XS_ERROR XsAttach (**XS_HANDLE** *hCamera*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the operation is not supported.

XS_HARDWARE_FAULT, if any hardware error occurs.

Parameters

hCamera

Specifies the handle to an open camera

Remarks

The routine handles the simultaneous connection to the camera from different computers. The routine is supported by HG and Y cameras only.

See also:

3.5.10. XsConfigureWriteToDisk

XS_ERROR XsConfigureWriteToDisk (**XS_HANDLE** *hCamera*, **XSULONG32** *nEnable,* **PXS_W2DCFG** *pCfg,* **XS_StreamingCallback** *pfnCallback*, **void*** *pUserData*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the option is not supported.

Parameters

hCamera

Specifies the handle to an open camera.

nEnable

Specifies whether the option is enabled or disabled.

pCfg

Specifies a pointer to the configuration structure.

pfnCallback

Specifies a pointer to a callback.

pUserData

Specifies a pointer to user data returned by the callback.

Remarks

The routine activates and deactivates direct write to disk with M, Pcie and XS-Mini cameras.

See also:

3.5.11. XsReadToVideo

XS_ERROR XsReadToVideo (**XS_HANDLE** *hCamera*, **XSULONG32** *nStartAddLo*, **XSULONG32** *nStartAddHi*, **XSULONG32** *nFrameIdx*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

XS_E_BUSY, if the camera is busy and the command cannot be performed.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrameIdxOrSize

Specifies the index or the size of the frame to read

Remarks

This function reads a single frame from the camera memory to the HDMI output. The user must specify the starting address and the index of the frame. If the index is set to 0xFFFFFF the camera snaps and image to the HDMI output (Live). The routine is active only on “new design” Y cameras.

See also:

3.5.12. XsLoadLookupTable

XS_ERROR XsLoadLookupTable (**XS_HANDLE** *hCamera*, **unsigned short *** *pnTable*, **XSULONG32** *nSize*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if the routine arguments are not valid.

Parameters

hCamera

Specifies the handle to an open camera

pnTable

Specifies the pointer to the lookup table array

nSize

Specifies the size of the array

Remarks

This function loads a user-defined lookup table. The XSP_LUT parameter should be set to XS_LUT_USERR first. The table is an array of unsigned short elements that converts the original sensor pixel depth (8, 10 or 12 bit) into the currently configured pixel depth.

See also: XsSetParameter, XSP_LUT

3.5.13. XsVideoPlayback

XS_ERROR XsVideoPlayback (**XS_HANDLE** *hCamera*, **XSULONG32** *nOption*, **XSULONG32** *nStartAddLo*, **XSULONG32** *nStartAddHi*, **XSULONG32** *nFrames,* **XSULONG32** *nStartFrameIdx,* **XSULONG32** *nStopFrameIdx*)

Return values

XS_SUCCESS if successful, otherwise

XS_E_INVALID_HANDLE, if the camera handle is not valid.

XS_E_INVALID_ARGUMENTS, if the arguments are not valid.

XS_E_NOT_SUPPORTED, if the routine is not supported.

XS_E_HARDWARE_FAULT, if any error occurs while calling the driver.

Parameters

hCamera

Specifies the handle to an open camera

nOption

Specifies the playback mode (0: off, 1: forward, 2: rewind). See XS_VIDEO_PB.

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrames

Specifies the total number of frames to playback

nStartFrameIdx

Specifies the index of the first frame of the sequence (range 0 to nFrames -1)

nStopFrameIdx

Specifies the index of the latest frame of the sequence (range 0 to nFrames -1)

Remarks

This routine activates or deactivates an asynchronous playback of images on the video (HDMI) output. The parameters to be specified are: the mode (off, forward or rewind), the starting address, the number of frames, the indexes of the first and the latest frame in the memory segment. If the sequence has been acquired in circular mode the value of the start index may be larger than the value of the stop index. The routine is active if the value returned by the XsGetCameraInfo routine with the XSI_ASYNC_VIDEO_PB parameter is 1. The status of the playback may be checked with a call to the XsGetCameraStatus routine. If the playback is active the routine returns the XSST_VPB_ON_CON or XSST_VPB_ON_COFF values in the status field and the index of the current frame in the pnInfo1 field.

See also:

4. LabVIEW™ Interface Reference

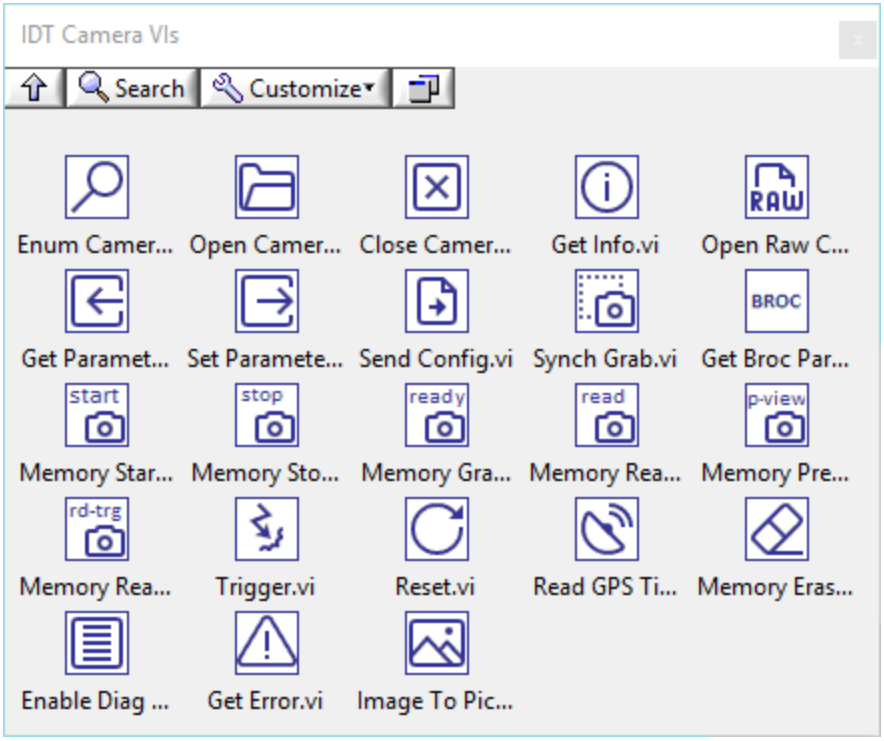
4.1. Overview

The LabVIEW™ Interface allows acquiring images and controlling the cameras from inside National Instruments LabVIEW application. It works with LabVIEW 2011 and greater, on Windows Vista, 7, 8 and 10. MAC OSX is not supported.

The LabVIEW™ Interface includes the **VIs (Virtual Instruments)** for controlling the camera and some sample VIs showing how to use the interface: the camera VIs are packaged in a library called **XS.LLB**) located in the **MotionProX** directory in the **user.lib** subdirectory of the LabVIEW folder. The examples are located in the **LabVIEW** subdirectory of the installation folder “C:\Program Files (x86)\IDT\CameraSDK xx.yy.zz”.

The VIs may be accessed by selecting the “Show Functions Palette” menu item from the Window” menu, then by clicking the “User Libraries” button and the “IDT Camera VIs” button.

The LabVIEW interface is supported on both 32 and 64 bit platforms of the SDK.



4.2. Initialization VIs

4.2.1. Overview: Initialization VIs

Initialization Virtual Instruments allow the user to initialize the cameras, enumerate the available cameras, open and close them.

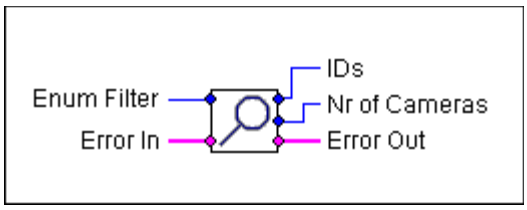
Enum Cameras enumerates the cameras currently connected to the computer.

Open Camera opens a camera.

Open Raw Camera opens a raw file like a virtual camera.

Close Camera closes a camera, previously open.

4.2.2. Enum Cameras



Inputs

Error

Specifies a standard error cluster input terminal

Enum Filter

Specifies the enumeration filter to detect differnet camera models

Outputs

Error

Specifies the return error code of the function (0 if it's is successful, non 0 otherwise)

IDs

Specifies the array containing the IDs of the enumerated cameras

Nr of Cameras

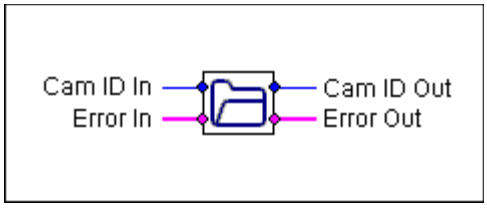
Specifies the number of enumerated cameras

Remarks

The VI enumerates the active cameras and returns a list of the enumerated cameras IDs. This VI must be set before **"Open Camera"** to find the available cameras. The "Enum Filter" input specifies which camera model is going to be enumerated. The "Nr of cameras" output contains the number of cameras. If any error occurs during the enumeration, the Error Out terminal signals the error condition.

See also: "Open Camera"

4.2.3. Open Camera



Inputs

Camera ID

Specifies the ID of the camera to be open, or 0 for the first available camera

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error condition

Camera ID

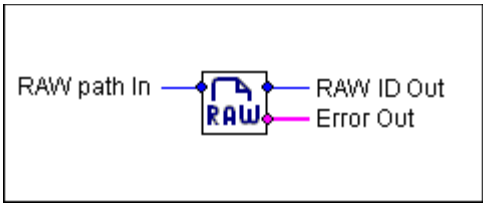
Specifies the ID of the opened camera

Remarks

The VI opens the camera with a specific ID. The value may be retrieved by calling the **"Enum Cameras"** VI. The user may supply a specific camera ID or 0: in this case the first available camera is open. If any error occurs during the open operation, the Error Out terminal signals an error code. The VI returns the ID of the open camera.

See also: "Close Camera"

4.2.4. Open Raw Camera



Inputs

Raw File In

Specifies the full path of the rawfile.xml or the directory where the file is stored.

Outputs

Error

Specifies the return error condition

Camera ID

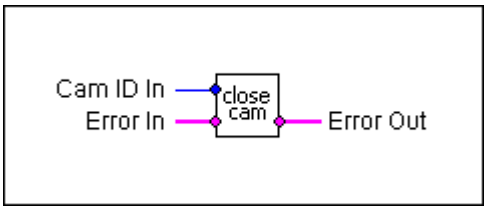
Specifies the ID of the virtual camera

Remarks

The VI opens the raw file like a virtual camera. VI. The user may supply the full path of the rawfile.xml file or the path to the directory where the file and the sequence are stored. If any error occurs during the open operation, the Error Out terminal signals an error code. The VI returns the ID of the open virtual camera.

See also: “Close Camera”

4.2.5. Close Camera



Inputs

Camera ID

Specifies the ID of the camera to be closed (it may be a camera or a raw file)

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error condition

Remarks

This VI closes a camera or a raw file previously open. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: “Open Camera”, “Open Raw Camera”

4.3. Configuration VIs

4.3.1. Overview: Configuration VIs

Configuration Virtual Instruments allow the user to read information from the camera, read configuration parameters from the camera and write configuration parameters to the camera.

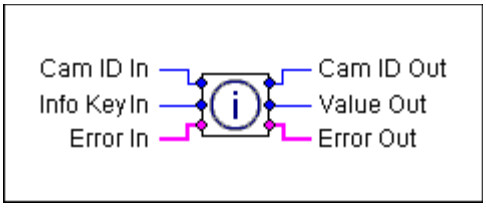
Get Info reads information from the camera, such as camera model, firmware version, etc.

Get Parameter reads a single specific parameter from the camera configuration and reads the minimum and maximum values.

Set Parameter writes a single specific parameter to the configuration.

Send Config flushes the updated configuration to the camera.

4.3.2. Get Info



Inputs

Camera ID

Specifies a valid camera ID

Info Key

Specifies which parameter has to be returned by the VI

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Value

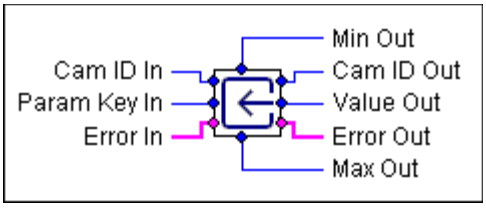
Specifies the value of the info parameter

Remarks

This VI returns camera specific information, such as sensor type or version numbers, generally state-independent information. See the Appendix B for a list of all the available Info Key values. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: “Get Parameter”, “Set Parameter”

4.3.3. Get Parameter



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Param Key

Specifies the index of the parameter

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Value

Specifies the current value of the parameter

Min

Specifies the minimum value of the parameter

Max

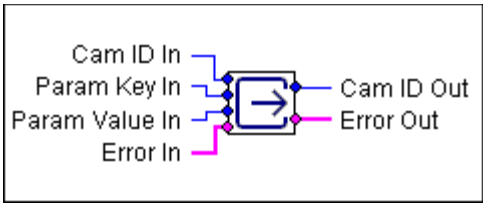
Specifies the maximum value of the parameter

Remarks

This VI reads a specific configuration parameter from the camera and returns its value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: “Set Parameter”

4.3.4. Set Parameter



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Param Key

Specifies the index of the parameter

Value

Specifies the value of the parameter

Outputs

Camera ID

Specifies the camera ID

Error

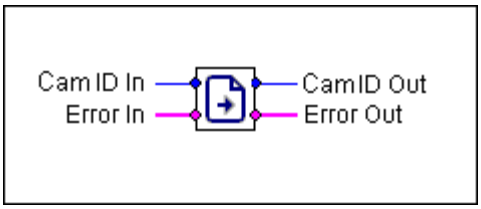
Specifies the return error condition

Remarks

This VI writes a specific configuration parameter to the configuration set. The parameter key is one of the input parameters. A list of the parameters is available in Appendix C. If any error occurs during the operation, the Error Out terminal signals an error code. The user may call the "**Set Parameter**" VI several times to set different parameters, and then call the "**Send Config**" VI to download the configuration to the camera.

See also: "Get Parameter", "Send Config"

4.3.5. Send Config



Inputs

Device ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Remarks

This VI sends the current configuration to the camera and activates it. The user may call the "**Set Parameter**" VI several times to set different parameters, and then call the "**Send Config**" VI to download the configuration to the camera. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: "Get Parameter", "Set Parameter"

4.4. Camera Memory Acquisition VIs

4.4.1. Overview

The camera memory acquisition Virtual Instruments allow the user to acquire images in the camera memory: snap synchronously, start and stop acquisitions, read images during recording, check the status of an acquisition and trigger the camera.

Synch Grab synchronously snaps an image (or two in double exposure mode) and outputs it as an image object.

Memory Start Grab starts an acquisition in the camera memory.

Memory Stop Grab stops the current acquisition in the camera memory.

Memory Preview previews images during the acquisition.

Memory Grab Ready returns the status of the current acquisition.

Memory Read Data reads images from the camera memory.

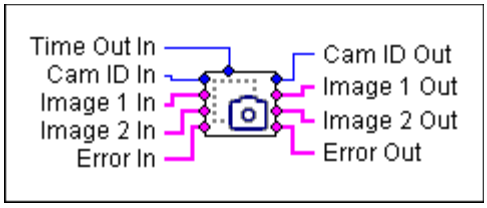
Memory Erase erases the camera memory.

Memory Read Trigger Position reads the position (index) of the frame that received the trigger.

Memory Read BROC Data reads the parameters of a BROC segment**.**

Trigger issues a software trigger to the camera.

4.4.2. Synch Grab



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Time Out

Specifies the grab time out in ms

Image 1

Specifies the first image input (requires IMAQ)

Image 2

Specifies the second image input (double exposure mode only, requires IMAQ)

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Image 1

Specifies the first image output

Image 2

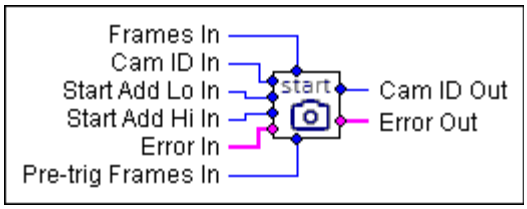
Specifies the second image output (double exposure mode only)

Remarks

This VI snaps an image (or two) from the camera. The image is snapped synchronously and the function exits when the frame has been recorded or a time out occurs. If the camera mode is set to double exposure, two frames are acquired. The VI outputs two image objects, but the second is valid only in double exposure mode. The image format depends on the image size and pixel depth (8 bit or 16 bit mono, 32 bit RGBA color).

See also:

4.4.3. Memory Start Grab



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Frames

Specifies the number of frames to acquire

Start Add Lo

Specifies the low-order 32 bit value of the memory starting address

Start Add Hi

Specifies the high-order 32 bit value of the memory starting address

Pre-trigger Frames

Specifies the number of frames to be acquired before the trigger (circular mode only)

Outputs

Camera ID

Specifies the camera ID

Error

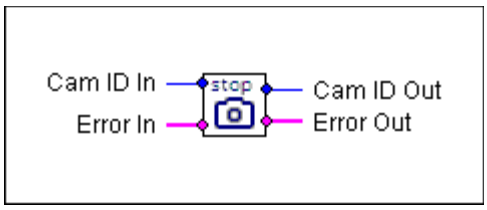
Specifies the return error condition

Remarks

This VI starts an acquisition in the camera memory and returns immediately. The user may know when the frames have been captured by calling the "**Memory Grab Ready**" VI. If any error occurs during the operation, the Error Out terminal signals an error code. The memory start address can be 0.

See also: "Memory Stop Grab"

4.4.4. Memory Stop Grab



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

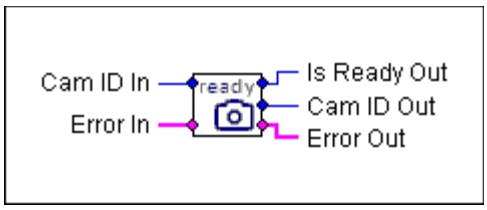
Specifies the return error condition

Remarks

This VI stops any camera memory acquisition previously started. If any error occurs during the operation, the Error Out terminal signals an error code.

See also: “Memory Start Grab“

4.4.5. Memory Grab Ready



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Is Ready

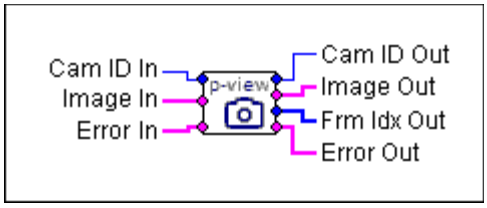
Specifies whether the acquisition is finished (1) or not (0).

Remarks

This VI returns the status of the current acquisition. If the “Is Ready” value is 1 the images have been recorded and saved to camera memory, otherwise not.

See also: “Memory Start Grab“, “Memory Stop Grab“

4.4.6. Memory Preview



Inputs

Camera ID

Specifies a valid camera ID

Image

Specifies the image input

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Image

Specifies the image input

Frame Index

Specifies the index of the previewed frame

Error

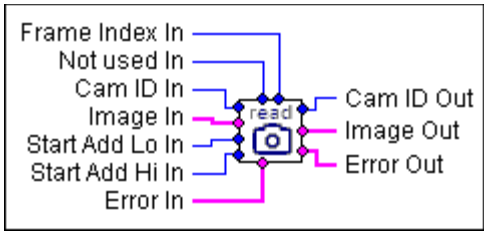
Specifies the return error condition

Remarks

This VI may be used only during a recording. It returns the latest acquired image and the index from the beginning of the sequence.

See also: “Memory Start Grab“

4.4.7. Memory Read Data



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Frame Index

Specifies the index of the frame (0 to N-1)

not Used

This parameter is not used and ignored. Set its value to 0.

Start Add Lo

Specifies the low-order 32 bit value of the memory starting address

Start Add Hi

Specifies the high-order 32 bit value of the memory starting address

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Image

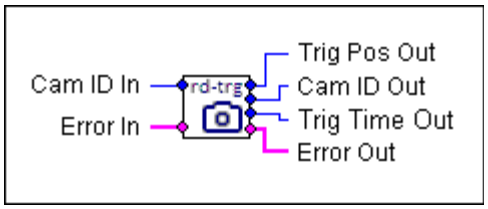
Specifies the image output

Remarks

This VI reads data from the camera memory and converts it into an image object. The user has to specify the starting address (usually 0) and the frame index. The driver uses the current camera settings to compute the frame size and converts the image into the current format (mono 8, 10, 12 bit, or color 32 bit RGBA). The image format depends on the image size and pixel depth.

See also: “Memory Start Grab“, “Memory Stop Grab“

4.4.8. Memory Read Trigger Position



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Trig Pos

Specifies the index of the triggered frame in the acquired sequence

Trig Time

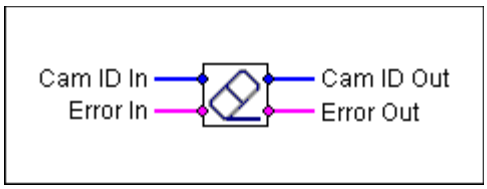
Specifies the time distance before the leading edge of frame zero and the trigger pulse (in microseconds)

Remarks

This VI may be used only if the “record mode” parameter has been set to 1 (circular) and an acquisition of images has been performed. The returned values are valid until a new acquisition is called. For further information about trigger position, please refer to the “Triggering” topic in the “Using the SDK” section.

See also: “Memory Start Grab“, “Memory Stop Grab“

4.4.9. Memory Erase



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

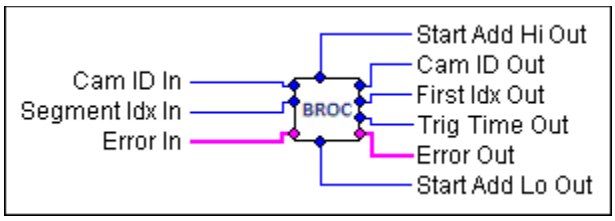
Specifies the return error condition

Remarks

This VI erases the memory on HG cameras

See also:

4.4.10. Get BROC parameters



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Segment

Specifies the index of the BROC segment

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Start Add Lo and HI

Returns the low order and the high order words of the BROC segment start address

First Index

Returns the index of the first image in the BROC segment

Trigger Time

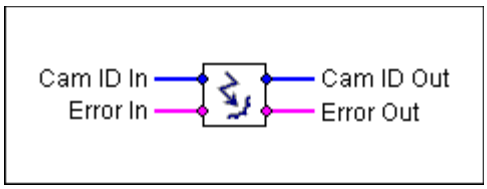
Returns the trigger time in the BROC segment

Remarks

This VI is supported on cameras that support hardware BROC. It reads the information of a BROC segment.

See also:

4.4.11. Trigger



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

Specifies the return error condition

Remarks

This VI issues a software trigger to the camera. The camera must be recording and the record mode set to 1 (circular).

See also:

4.5. Miscellaneous VIs

4.5.1. Overview: Miscellaneous VIs

Miscellaneous Virtual Instruments allow the user to convert image formats and manage the error conditions in the VIs.

Reset sends a reset command to the camera.

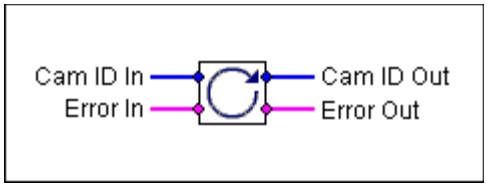
Read GPS Timing reads the IRIG/GPS/PTP time stamp and returns it as a string.

Enable Diag Trace enables and disables the diagnostic trace.

Image To Picture converts an IMAQ image to a LabVIEW picture.

Get Error manages the error conditions in the other VIs (this VI is for internal use only).

4.5.2. Reset



Inputs

Camera ID

Specifies a valid camera ID

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

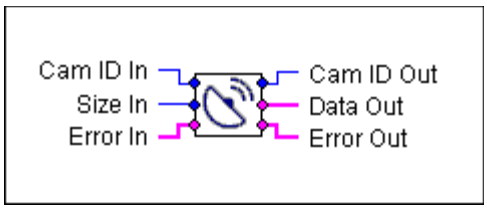
Specifies the return error condition

Remarks

This VI sends a reset command to the camera

See also:

4.5.3. Read GPS Timing



Inputs

Camera ID

Specifies a valid camera ID

Size

Specifies the size of the IRIG buffer

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Data

Specifies the IRIG/GPS/PTP time stamp output

Error

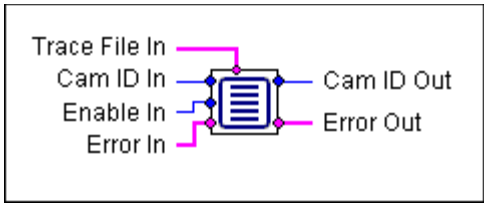
Specifies the return error condition

Remarks

This VI reads IRIG/GPS/PTP time stamp from the camera and returns it as a string. The size of the data buffer in bytes is specified in the Size input.

See also:

4.5.4. Enable Diag Trace



Inputs

Camera ID

Specifies a valid camera ID

Trace File

Specifies the path to the trace text file

Enable

Specifies if the trace is enabled or disabled

Error

Specifies a standard error cluster input terminal

Outputs

Camera ID

Specifies the camera ID

Error

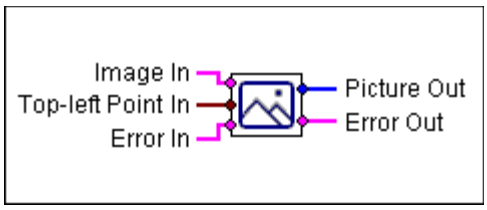
Specifies the return error condition

Remarks

This VI enables or disables the diagnostic trace in the camera driver. The trace messages are stored in a text file specified in the “Trace File” input.

See also:

4.5.5. Image To Picture



Inputs

Error

Specifies a standard error cluster input terminal

Top-Left point

Specifies the coordinates of the top-left point

Image

Specifies the image to convert

Outputs

Error

Specifies the return error condition

Picture

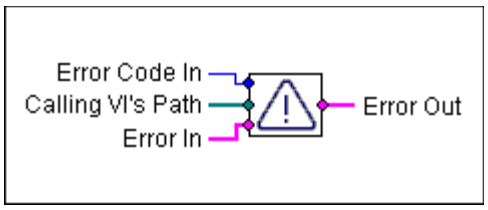
Specifies the output LabVIEW picture object

Remarks

This VI converts an IMAQ image object into a LabVIEW picture object. If any error occurs, the Error Out terminal signals this error.

See also:

4.5.6. Get Error



Inputs

Error Code

Specifies the camera error code

Calling VI's Path

Specifies the path of the VI which generates the error

Error

Specifies a standard error cluster input terminal

Outputs

Error

Specifies the return error condition

Remarks

This VI manages the error conditions in the other VIs (this VI is for internal use only).

See also:

4.6. How to use the VIs

4.6.1. Opening and closing a camera

Before calling any other VI, the camera must be open. To open a specific camera, the user supplies to the Open VI the unique ID of that camera or the value 0 to open the first available camera. To obtain the list of all available cameras you may call the “Enum Cameras” VI.

4.6.2. Configuring a camera

Before configuring a camera, several calls to the “Set Parameter” VI may be done. When the parameters have been set, a call to the “Send Config” VI downloads the new configuration and activates it. If you want to read a parameter value you may call the “Get Parameter” VI.

4.6.3. Acquiring images in real time

The camera frame stream may be acquired continuously. After opening the camera, the “Synch Grab” VI may be called.

4.6.4. Acquiring images in camera memory

To acquire a set of images in camera memory, the “Memory Start Grab” VI may be called. The VI starts an acquisition in the camera memory and returns immediately. To know when the acquisition is over, the user may call the “Memory Grab Ready” VI. The current acquisition may be stopped by the “Memory Stop Grab” VI. After the acquisition, the images may be read by the “Memory Read Data” VI.

4.6.5. Error handling

The LabVIEW interface uses the standard error cluster found in many LabVIEW VIs. The error cluster includes status, code and source parameters. When an error occurs, status is set to TRUE, source is set to the VI that caused the error, and code is set to one of the values in the table below.

Code	Description
0	Success, no error
1	Driver fault
2	Camera not found. The camera ID is incorrect or there are no cameras connected.
3	Bad image Format. Monochrome cameras support only 8 or 16 bit format.
4	Invalid value. You tried to set a parameter value that is out of range.
5	The configuration cannot be loaded from the camera.
6	The camera cannot be closed.
7	Unable to read a parameter from the camera configuration.
8	Unable to write a parameter to the camera configuration.

9	Unable to read info from the camera.
10	The configuration cannot be flushed to the camera.
11	Unable to allocate memory.
12	Unable to snap an image or to start memory acquisition.
13	Unable to read image data from the camera memory.
14	Unable to read the trigger position from the camera.
15	Unable to stop memory acquisition.
16	Bad image format. Only BGRA 32 bit images are supported.
17	Unable to calibrate background.
18	Bad image format. This monochrome camera supports only 8 bit image format.
19	Unable to trigger the camera. The software trigger is not supported or the camera is not recording.
20	Unable to read IRIG?GPS/PTP time stamp from the camera. Make sure that the 'IRIG parameter is set to 1 and the camera is properly connected.
21	Unable to reset the camera.
22	Unable to enable/disable the diagnostic trace.
23	Unable to erase the camera memory.
24	Unable to open the raw file.
25	Unable to download an image to the raw file.
100	Generic error.

4.7. Sample VIs

4.7.1. 1_enum_cameras

This sample shows how to display the result of a cameras enumeration. The output of the “Enum Cameras” VI is displayed in a group of four LED and four edit boxes. If a camera is enumerated the corresponding LED is turned on and the camera ID is displayed in the edit box.

4.7.2. 2_get_camera_info

This sample shows how to retrieve information from the camera such as camera model, firmware version, etc. The first available camera is open and the following information is retrieved and displayed: camera model, sensor model, firmware version.

4.7.3. 3_image_live

This sample shows how to continuously capture and display images from the camera. The example opens the first available camera, configures it with the default parameters, and acquires a single image. The acquisition output is displayed in a preview window, and then the camera is closed.

4.7.4. 4_image_live_error_check

This sample shows how to continuously capture and display images and how to handle an error condition. The sample opens the first available camera, configures it with the default parameters, and acquires a single image. The acquisition output is displayed in a preview window, and then the camera is closed.

4.7.5. 5_image_live_with_parameters

This sample shows how to capture images and interactively configure the camera. The sample opens the first available camera and allows the user to configure the following parameters: exposure time, frame period, sensor gain and ROI. Then the camera is configured and a single image is acquired. The acquisition output is displayed in a preview window.

4.7.6. 6_image_acquire

This example shows how to record a set of images in the camera memory using the “Memory Start Grab” VI and how to read and playback those images using the “Memory Read Data”. The user may select the configuration parameters and the number of frames to record, and then press the “Acquire” button. The current acquisition may be aborted. If the sequence has been successfully acquired a green LED is turned on. Then the user may playback the acquired sequence.

4.7.7. 7_misc

This example shows some of the advanced features of the interface. The VI enables the diagnostic trace, resets the camera, reads the IRIG/GPS/PTP time stamp and then disables the trace.

4.7.8. 8_open_raw_file

This example shows how to open a raw file like a virtual camera and playback the images using the “Memory Read Data” VI.

5. MATLAB™ Interface Reference

5.1. Overview

MATLAB™ Interface allows to acquire images and to control the cameras from inside Mathworks™ MATLAB application. The interface works with MATLAB 6.5 and greater, on Windows XP, Vista, 7, 8 and 10.

The interface includes a 64 bit 'MEX' file, packaged in a library called XStreamML.mexw64 and some example .m files to show how to use the interface.

Every routine may be called from a MATLAB™ script file in the form:

[output1, output2 ...] = XStreamML [input1, input2 …]

The number of inputs and outputs depends on the function selected. In any function call input1 is the name of the requested command (for ex. 'EnumCameras') and output1 is the result of the operation (0 = SUCCESS, otherwise ERROR).

The **idtdef.m** file contains a class that includes the definitions of all the parameters of the SDK (like in XstrmAPI.h file).

More details on the commands syntax may be retrieved by typing "help XStreamML" at MATLAB command prompt or opening the file **XStreamML.m** with a text editor.

The MATLAB interface reflects the SDK API with a few exceptions. The MATLAB interface and examples are listed below.

5.2. Initialization Functions

5.2.1. Overview: Initialization functions

Initialization functions allow the user to initialize the environment and the cameras.

Version retrieves the driver version.

SetNetAdapterIPAddress selects the IP address of the network adapter connected to the GE cameras.

InitPCIeMemory initializes the computer memory buffer for Xstream PCIe cameras.

EnumCameras enumerates the IDs of the cameras connected to the computer.

OpenCamera opens a camera.

OpenRawcamera opens a raw sequence like a virtual camera.

CloseCamera closes a camera previously open.

5.2.2. Version

[*strVersion***]** = XStreamML (‘**Version**’)

Inputs

None

Outputs

strVersion

Specifies the driver version string (for example, ‘2.13.01’)

Remarks

This function returns the MATLAB interface version string.

See also:

5.2.3. SetNetAdapterIPAddress

[*nResult***] =** XStreamML **(‘SetNetAdapterIPAddress’**, strAddress**)**

Inputs

nMB

Specifies the IP address of the network adapter (for ex. '10.10.10.2')

Outputs

nResult

Specifies the return error code (0 if the function is successful, otherwise error)

Remarks

The routine selects the IP address of the network adapter connected to the GigaEthernet cameras. This routine should be called before the **EnumCameras** routine. If the address is not specified the enumerator will search for cameras from every network adapter installed in the local computer and the enumeration process will be slower.

See also: EnumCameras

5.2.4. EnumCameras

[*nResult*, *nItems*, *svArray***] =** XStreamML **(‘EnumCameras’**, *nEnumFilter***)**

Inputs

nEnumFilter

Specifies the enumeration filter

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nItems

Specifies the number of detected cameras

svArray

Specifies the array containing the IDs of the detected cameras

Remarks

The routine enumerates the active cameras and return an array filled with the detected cameras IDs. This routine must be called before **OpenCamera** to find out which cameras are available. The nItems variable contains the number of detected cameras. The nEnumFilter variable specifies which camera model is going to be enumerated. If any error occurs during the cameras enumeration, the nResult variable contains an error code.

See also: OpenCamera

5.2.5. InitPCIeMemory

[*nResult***] =** XStreamML **(‘InitPCIeMemory’**, *nMB***)**

Inputs

nMB

Specifies the size in MB of the memory buffer

Outputs

nResult

Specifies the return error code (0 if the function is successful, otherwise error)

Remarks

The routine allocates the computer buffer for the Xstream PCIe camera. This routine should be called before **OpenCamera**.

See also:

5.2.6. OpenCamera

[*nResult*, *nCameraId***] =** XStreamML **(‘OpenCamera’**, *nInputId***)**

Inputs

nInputId

Specifies the ID of the camera to be opened, or 0 for the first available camera

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nCameraId

Specifies the ID of the opened camera

Remarks

The routine opens the camera whose ID is in the variable *nInputId*. The value can be retrieved calling the **EnumCameras** enumeration function. The user may supply a specific camera ID or 0: in this case the first available camera is opened. If any error occurs during the camera opening, the routine returns an error code in the nResult variable, otherwise it returns 0. The function also returns the camera Id.

See also: CloseCamera

5.2.7. OpenRawCamera

[*nResult, nCameraId***]** = XStreamML ('**OpenRawCamera**', *lpszRawFilePath*)

Inputs

lpszRawFilePath

Specifies the full path to the raw file.

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nCameraId

Specifies the virtual camera unique id (used in other routine calls)

Remarks

The routine opens the RAW file with path *lpszRawFilePath*. The variable may contain be the full path to the rawfile.xml file or the full path to the directory that includes the file and the raw sequence.

See also: CloseCamera

5.2.8. CloseCamera

[*nResult***]** = XStreamML (‘**CloseCamera**’, *nCameraId*)

Inputs

nCameraId

Specifies the ID of the camera to be closed

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function closes a camera previously open. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: OpenCamera

5.3. Configuration functions

5.3.1. Overview: Configuration functions

Configuration functions allow the user to read information from the camera, read configuration parameters from the camera and write configuration parameters to the camera.

GetCameraInfo reads information from the camera, such as camera model, firmware version, etc.

GetParameter reads a specific parameter from the current settings configuration.

SetParameter writes a specific parameter to the current settings configuration.

SendCfg downloads the current configuration to the camera and activates the new settings.

5.3.2. GetCameraInfo

[*nResult*, *nInfoValue***] =** XStreamML **(‘GetCameraInfo’,** *nCameraId***,** *nInfoKey***)**

Inputs

nCameraId

Specifies a valid camera ID

nInfoKey

Specifies which parameter the function has to return

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nInfoValue

Specifies the index of the info

Remarks

This function returns camera specific information, such as sensor type or version numbers, generally state-independent information. See the Appendix B for a list of all the available nInfoKey values.

See also: GetParameter

5.3.3. GetParameter

[*nResult, nValue, nMinValue, nMaxValue***] =** XStreamML **(‘GetParameter’**, *nCameraId*, *nParamKey***)**

Inputs

nCameraId

Specifies a valid camera ID

nParamKey

Specifies the index of the parameter

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nValue

Specifies the current value of the parameter

nMinValue

Specifies the minimum value of the parameter

nMaxValue

Specifies the maximum value of the parameter

Remarks

This function reads a specific configuration parameter from the camera and returns the parameter value, the minimum and the maximum. The parameter key is one of the input parameters. A list of the parameters constants is available in Appendix C. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: SetParameter

5.3.4. SetParameter

[*nResult***] =** XStreamML **(‘SetParameter’**, *nCameraId*, *nParamKey, nValue***)**

Inputs

nCameraId

Specifies a valid camera ID

nParamKey

Specifies the index of the parameter

nValue

Specifies the value of the parameter

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function writes a specific configuration parameter to the camera. The parameter key is one of the input parameters. A list of the parameters indexes is available in Appendix C. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0. Note that the new value will be active only after the SendCfg function has been called.

See also: SendCfg, GetParameter

5.3.5. SendCfg

[*nResult***] =** XStreamML **(‘SendCfg’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function downloads the current configuration to the camera and activates new settings. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: GetParameter

5.4. Camera Memory Acquisition Functions

5.4.1. Overview: Camera Memory Acquisition Functions

The camera memory acquisition functions allow the user to acquire images in the camera memory: start and stop acquisitions, read images from the camera memory and check the status of an acquisition.

SynchGrab grabs synchronously one or two images from the camera.

MemoryStartGrab starts an acquisition in the camera memory

MemoryStopGrab stops the current acquisition in the camera memory.

MemoryPreview reads the latest acquired frame during an acquisition and/or reads the number of frames acquired so far.

MemoryReadData reads image data from the camera memory.

MemoryDownloadRawFrame downloads an image into a RAW file.

MemoryReadTriggerPosition reads the trigger position in the camera memory.

MemoryErase erases the camera memory (HG cameras only).

GetBrocParameters reads the parameters of a BROC segment.

GrabIsReady checks the status of the current acquisition.

Trigger issues a software trigger to the camera.

5.4.2. SynchGrab

[*nResult*,image1,image2**] =** XStreamML **(‘SynchGrab’**, *nCameraId*, *nTimeOut***)**

Inputs

nCameraId

Specifies a valid camera ID

nTimeOut

Specifies the grab time out in ms

Outputs

nResult

Specifies the return error code (0 if the function is successful, otherwise not 0)

image1

Specifies the array where the image is stored.

Image2

Specifies the array where the second image is stored (double exposure mode only)

Remarks

It grabs an image (or two) from the camera. The image grab is synchronous and the function exits when the frame has been grabbed or a time out occurs. If the camera mode is set to double exposure, two frames are acquired and the function outputs two image buffers (arrays), otherwise only the first is valid. The array dimension depends on the image size and pixel depth: if the pixel depth is 8 or 24, the array is an ‘**unsigned char**’ array; if the pixel depth is 10/12/30 or 36, the array is an ‘**unsigned short**’ array.

See also:

5.4.3. MemoryStartGrab

[nResult**] =** XStreamML **('MemoryStartGrab'**, *nCameraId*, *nStartAddLo*, *nStartAddHi*, *nFrames*, *nPreTrigFrames***)**

Inputs

nCameraId

Specifies a valid camera ID

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrames

Specifies the number of frames which have to be acquired

nPreTrigFrames

Specifies the number of frames to be acquired before the trigger; it's valid only if the record mode is circular.

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function starts an acquisition in the camera memory and returns immediately. The user may know when the frames have been captured by calling the GrabIsReady function. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: MemoryStopGrab

5.4.4. MemoryStopGrab

[*nResult***]** = XStreamML (‘**MemoryStopGrab**’, *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function stops any camera memory acquisition previously started. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also: MemoryStartGrab

5.4.5. MemoryPreview

[nResult, image, nFrameIndex**] =** XStreamML **(‘MemoryPreview’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the returned error code (0 if the function is successful, otherwise not 0)

image

Specifies the array where the image is stored.

nFrameIndex

Specifies the index of latest acquired frame

Remarks

This routine may be called during an acquisition in camera memory. It reads the latest acquired frame and the number of frames acquired so far. The routine may be called to preview an acquisition.

See also: MemoryStartGrab

5.4.6. MemoryReadData

[nResult, image**] =** XStreamML **('MemoryReadData'**, *nCameraId*, *nIsFirst*, *nStartAddLo*, *nStartAddHi*, *nFrameIdx***)**

Inputs

nCameraId

Specifies a valid camera ID

notUsed

This parameter is not used and ignored. Set its value to 0.

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address

nFrameIdx

Specifies the index of the frame which have to be read

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

image

Specifies the array where the image is stored

Remarks

This function reads data from the camera memory into the specified buffer. The user must specify the starting address and the frame index. The driver uses the current camera settings to compute the frame size and convert each 10 bit image into the current format (8 or 10 bit). The user must be sure that the current settings of image format, pixel depth and pixel gain are the same used in the acquisition. For further information, please refer to the "Multiple Acquisitions" and "Camera Memory Management" topics in the "Using the SDK" section. The array dimension depends on the image size and pixel depth: if the pixel depth is 8, the array is an '**unsigned char**' array; if the pixel depth is 10, the array is an '**unsigned short**' array.

See also: MemoryStartGrab, MemoryStopGrab

5.4.7. MemoryDownloadRawFrame

[nResult, image**] =** XStreamML **('MemoryDownloadRawFrame'**, *nCameraId*, *lpszRawFilePath, nStartAddLo, nStartAddHi, nFrameIdx, nPageIdx, nTotFrames*)

Inputs

nCameraId

Specifies a valid camera ID.

lpszRawFilePath

Specifies the full path of the RAW file.

nStartAddLo

Specifies the low-order 32 bit value of the memory starting address.

nStartAddHi

Specifies the high-order 32 bit value of the memory starting address.

nFrameIdx

Specifies the index of the frame in camera memory.

nPageIdx

Specifies the index of the frame in the sequence (0 to nTotFrames).

nTotFrames

Specifies the total number of downloaded frames.

Remarks

This function downloads a frame from the camera memory into the specified Raw file. The full path of the Raw file may be specified without the extension because the driver will ad a ".raw" extension to it. The user must specify the starting address and the index of the frame in camera memory. If a sequence of N frames has been acquired in circular mode, the position of the trigger index (T) should be read and the frames indexes should be ordered (see the example in chapter 2 "Using the SDK"). Also, the pages index (from 0 to N-1) and the total number of frames (N) must be specified.

See also: MemoryStartGrab, MemoryStopGrab

5.4.8. MemoryReadTriggerPosition

[*nResult*, *nFrameIdx, nTriggerTime***]** = XStreamML (‘**MemoryReadTriggerPosition**’, *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nFrameIdx

Specifies the index of the triggered frame in the acquired sequence

nTriggerTime

Specifies the time distance between the leading edge of frame zero and the trigger pulse (in microseconds)

Remarks

This function is valid only if the record mode is set to circular and the routine MemoryStartGrab has been called with the parameter nPreTrigFrames <> 0. The returned values are valid until a new acquisition or snap API is called. For further information about trigger position, please refer to the “Triggering” topic in the “Using the SDK” section.

See also: MemoryStartGrab

5.4.9. MemoryErase

[*nResult***]** = XStreamML (‘**MemoryErase**’, *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

The routine is called to erase the memory of HG cameras. If the memory is not erased the user cannot start a new acquisition.

See also:

5.4.10. GetBrocParameters

[nResult, nStartAddrLo, nStartAddrHi, n1stFrmIdx, nTrgTime**]** = XStreamML('**GetBrocParameters**', nCameraId, nSectIdx)

Inputs

nCameraId

Specifies a valid camera ID

nSectIdx

Specitifes the index of the selected BROC section

Outputs

nStartAddrLo, nStartAddrHi

Specify the least significant and the most significant parts of the BROC section start address in camera memory

n1stFrmIdx

Specifies the index of the first frame of the BROC section.

nTrgTime

Specifies the trigger time delay from the sync signal edge.

Remarks

This function returns the information of a BROC section. It returns the address, the position of the first index and the trigger time in the section. The function is supported on cameras that support hardware BROC.

See also:

5.4.11. GrabIsReady

[*nResult*, *nIsReady***]** = XStreamML(‘**MemoryReadTriggerPosition**’, *nCameraId*)

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

nIsReady

Specifies whether the acquisition is finished (1) or not (0).

Remarks

This function returns the status of the current acquisition. If the returned value nIsReady is 1 the current acquisition has been completed, otherwise not.

See also: MemoryStartGrab

5.4.12. Trigger

[*nResult***] =** XStreamML **(‘Trigger’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function issues a software trigger to the camera. The software trigger is effective if the record mode is set to circular. If any error occurs during the operation, the routine returns an error code in the nResult variable, otherwise it returns 0.

See also:

5.5. Miscellaneous Functions

5.5.1. Overview: Miscellaneous Functions

Reset resets the camera

ReadGPSTiming reads the GPS/IRIG/PTP info from the current frame (string).

EnableDiagnosticTrace reads enables and disables the diagnostic trace in the driver.

5.5.2. Reset

[nResult**] =** XStreamML **(‘Reset’**, *nCameraId***)**

Inputs

nCameraId

Specifies a valid camera ID

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function resets the camera.

See also:

5.5.3. ReadGPSTiming

[nResult, pBits**] =** XStreamML **(‘ReadGPSTiming**, *nCameraId, nSize***)**

Inputs

nCameraId

Specifies a valid camera ID

nSize

Specifies the size in bytes of the IRIG buffer

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function reads GPS/IRIG/PTP data from the current frame and returns it in string format in the pBits variable

See also:

5.5.4. EnableDiagnosticTrace

[nResult**]** **=** XStreamML **(‘EnableDiagnosticTrace**’, *nCameraId, pszTraceFilePath, nEnable***)**

Inputs

nCameraId

Specifies a valid camera ID

pszTraceFilePath

Specifies the path of the trace text file

nEnable

Specifies if the trace is enabled or disabled

Outputs

nResult

Specifies the return error code of the function (0 if the function is successful, otherwise not 0)

Remarks

This function enables or disables the camera diagnostic trace in the driver. The diagnostic messages are stored in a text file specified in the pszTraceFile path parameter.

See also:

5.6. How to program with the Interface functions

5.6.1. Opening and closing a camera

Before calling any other routine, the camera must be open. To open a specific camera, the user supplies to the OpenCamera routine the unique ID of that camera or the value 0 to open the first available camera. To obtain the list of all available cameras, call the EnumCameras function.

5.6.2. Configuring a camera

Before configuring a camera, several calls to the SetParameter function may be done. When the parameters have been set, a call to the SendCfg function downloads the new configuration and activates it. If you want to read a parameter value you may call the GetParameter.

5.6.3. Previewing images in real time

The camera can continuously preview images. After opening the camera, the SynchGrab routine may be called.

5.6.4. Acquiring images in camera memory

To acquire a set of images in camera memory, you may call the MemoryStartGrab function. This routine starts an acquisition in the camera memory and then returns immediately. To know when the acquisition has been performed, use the GrabIsReady function. You may stop the current acquisition by calling the MemoryStopGrab routine and read the images recorded in camera memory by calling the MemoryReadData function.

5.6.5. Error handling

The MATLAB interface returns the same error codes displayed in the Error Handling section of the LabVIEW interface reference chapter.

5.7. Examples

5.7.1. CamEnum

This example shows how to read the list of all available cameras.

5.7.2. CamGetInfo

This example shows how to open a camera and read information.

5.7.3. CamReadParam

This example shows how to open a camera and read parameters from the current configuration.

5.7.4. CamImageSnap

This example shows how to synchronously capture and display a single image.

5.7.5. CamRecAndSave

This example shows how to capture images into the camera memory, download and save them to the local hard disk.

5.7.6. CamLiveRec

This example shows how to stream a live image or capture images in the camera memory and play them back. Also, it lets the user set some parameters, such as exposure, frame rate, pixel gain and ROI.

5.7.7. CamRawRead

This example shows how to open a RAW file, read images and save them.

6. RAW Reader Library

6.1. Overview

The RAW Reader Library is a set of routines designed to read images from the IDT RAW files. Each folder containing a raw sequence includes the following files:

Rawfile.xml: the main file that identifies the raw sequence.

RawStats.xml: a file containing statistics about the download speed. It may be ignored.

RawLut.xml: a file containing the Look up table applied to the image.

***.raw files**: file(s) containing the raw data.

***.cal file**: a file containing the calibration if the camera is non-pipeline (M-series, X-series and some old Y and N series).

Definitions and declarations are in the RawReadAPI.h file in the Include directory of the SDK. The binary modules are XRawReader32.dll (32 bit version) and XRawReader64.dll (64 bit version) in the Bin directory of the SDK.

6.2. Program Interface Reference

The interface includes routines to open, read and close a raw sequence.

XrGetVersion returns the version of the SDK.

XrOpen opens a raw file and returns a handle.

XrClose closes a file previously open.

XrReadInfo reads information from the open file, such as width, height, pixel depth.

XrReadAdvancedInfo reads advanced information from an open file, such as the camera name, the serial number and the number of pre-trigger frames.

XrRead Frame reads a frame from the raw file (if the camera is color, the frame format is Bayer).

6.2.1. XrGetVersion

XR_ERROR XrGetVersion (**unsigned int** ***pnVersionMS**, **unsigned int** ***pnVersionLS**)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_PATH, if the file path is not valid.

Parameters

pnVersionMS

Specifies the pointer to the variable that receives the most significant 32 bit of the version.

pnVersionLS

Specifies the pointer to the variable that receives the least significant 32 bit of the version.

Remarks

The routine reads the SDK version (64 bit) and returns it in the variables pointed by *pnVersionMS* (most significant 32 bit) and *pnVersionLS* (least significant 32 bit). In each 32 bit field there is an upper 16 bit number and a lower 16 bit number. The version is made of four numbers.

See also:

6.2.2. XrOpen

XR_ERROR XrOpen (**const char** *****_lpszFilePath_, **int** _nRGBMode_, **PXR_HANDLE*** _pFileHandle_)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_PATH, if the file path is not valid.

Parameters

lpszFilePath

Specifies the full path to the “rawfile.xml” file or to the folder containing the raw sequence.

nRGBMode

Specifies if the file is open as Bayer (0) or as RGB (1). If the images are monochrome, the parameter is ignored.

pHandle

Specifies the pointer to the variable that receives the file handle

Remarks

The routine opens the RAW file with path *lpszFilePath*. The variable may contain be the full path to the rawfile.xml file or the full path to the directory that includes the file and the raw sequence. If the nRGBMode parameter is set to 0, the images are open as single component (Bayer) images, if it is set to 1 the images are open as three component (BGR) images.

See also: **XrClose**

6.2.3. XrClose

XR_ERROR XrClose (**XR_HANDLE** *hFile*)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_HANDLE, if the handle is not valid.

Parameters

hFile

Specifies the handle to an open raw file

Remarks

Closes an open raw file

See also: **XrOpen**

6.2.4. XrReadInfo

XR_ERROR XrReadInfo (**XR_HANDLE** *hFile*, **int** ***pnWid*, **int** ***pnHgt*, **int** ***pnPixDepth*, **int** ***pnColor*, **int** ***pnCFAPattern*, **int** ***pnTotFrames, int ***pnRotation*)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_HANDLE, if the file handle is not valid.

XR_E_INVALID_ARGUMENTS, if one or more arguments are not valid.

Parameters

hFile

Specifies the handle to an open file

pnWid

Specifies the pointer to a variable that receives the image width in pixel

pnHgt

Specifies the pointer to a variable that receives the image height in pixel

pnPixDepth

Specifies the pointer to a variable that receives the image pixel depth (the range is 8 to 16)

pnColor

Specifies the pointer to a variable that receives whether the image is mono (0) or color (1)

pnCFAPattern

Specifies the pointer to a variable that receives the CFA pattern of the image (color only). The values are 0 (GRBG), 1 (BGGR), 2 (RGGB), 3 (GBRG).

pnTotFrames

Specifies the pointer to the variable that receives the total number of frames in the sequence

pnRotation

Specifies the pointer to the variable that receives the rotation (0: no rotation, 1: 90 degrees, 2: 180 degrees, 3: 270 degrees)

Remarks

This function returns the raw file information, necessary to open and display the frames.

See also: **XrReadAdvancedInfo**

6.2.5. XrReadAdvancedInfo

XR_ERROR XrReadAdvancedInfo (**XR_HANDLE** *hFile*, **unsigned int** ***pnCamSerial*, **char** ***pszCamName*, **int** ***pnPreTrigFrames*, **int** ***pnExposure*, **int** ***pnFps*, **int** ***pnTimeFromTrig*, **int** ***pnTimeFromStart*)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_HANDLE, if the file handle is not valid.

XR_E_INVALID_ARGUMENTS, if one or more arguments are not valid.

Parameters

hFile

Specifies the handle to an open file

pnCamSerial

Specifies the pointer to a variable that receives the camera serial number

pszCamName

Specifies the pointer to a buffer that receives the camera name

pnPreTrigFrames

Specifies the pointer to a variable that receives the number of pre-trigger frames in the sequence

pnExposure

Specifies the pointer to a variable that receives the exposure in microseconds

pnFps

Specifies the pointer to a variable that receives the speed in frames per second

pnTimeFromTrig

Specifies the pointer to a variable that receives the time between the leading edge of frame zero and the trigger pulse (microseconds).

pnTimeFromStart

Specifies the pointer to a variable that receives the time between the beginning of the recording and the leading edge of frame zero (microseconds).

Remarks

This function returns some advanced information about the raw file sequence.

See also: **XrReadInfo**

6.2.6. XrReadFrame

XR_ERROR XrReadFrame (**XR_HANDLE** *hFile*, **int** *nFrameIndex*, **void*** *pDataBuf,* **int** *nDataBufSize*)

Return values

XR_SUCCESS if successful, otherwise

XR_E_INVALID_HANDLE, if the file handle is not valid.

XR_E_READ, if any error occurs while calling the driver.

Parameters

hFile

Specifies the handle to an open camera

nFrameIndex

Specifies the index of the frame to read

pDataBuf

Specifies the pointer to the buffer where the data has to be copied.

nDataBufSize

Specifies the size of the destination buffer in bytes

Remarks

This function reads a single frame from the raw file into the specified buffer.

See also:

7. Appendix

7.1. Appendix A - Return Codes

The following table shows the values of the codes returned by the SDK APIs. The values can be found in the **XStrmAPI.h** header file in the **Include** subdirectory.

Code	Value	Notes
XS_SUCCESS	0	OK – No errors
XS_E_GENERIC_ERROR	1	Generic Error
XS_E_NOT_SUPPORTED	2	The function is not supported for this device
XS_E_INVALID_VALUE	3	Invalid parameter value
XS_E_INVALID_CFG	4	Invalid XS_SETTINGS structure. The field cbSize must match the size of the structure.
XS_E_INVALID_HANDLE	5	Invalid XS_HANDLE camera handle
XS_E_INVALID_CAMERA_ID	6	Invalid camera id used in XsOpenCamera. The ID is retrieved calling the XsEnumCameras routine
XS_E_INVALID_ARGUMENTS	7	Invalid function arguments
XS_E_READONLY	8	The parameter is read-only and cannot be modified
XS_E_CAM_ALREADY_OPEN	9	The camera is already open.
XS_E_HARDWARE_FAULT	10	Hardware error. To retrieve the hardware error code call the XsGetHardwareError routine.
XS_E_BUSY	11	The camera is busy and the operation cannot be performed
XS_E_QUEUE_FULL	12	The queue is full, cannot queue anymore
XS_E_BUFFER_TOO_SMALL	13	The buffer size is too small to perform the operation
XS_E_TIMEOUT	14	Operation time out.
XS_E_NOT_RECORDING	15	An attempt of Live while Record is done when the camera is not recording
XS_E_MALLOC	16	Unable to allocate memory
XS_E_ABORTED	17	A procedure has been aborted
XS_E_NOT_IN_FLASH	18	The requested information is not in flash memory
XS_E_EXP_LICENSE	19	The camera license has expired
XS_E_W2D_OVERRUN	20	Memory Overrun when saving data to disk in streaming mode.
XS_E_DMA_OVERRUN	21	DMA overrun error (X-Stream PCIe camera)

7.2. Appendix B – Hardware Error Codes

The following table shows the values and a brief description of the error codes returned by the XsGetHardwareError routine.

Value	Description
0	OK – No errors
1	Generic Error
2	The handle to the hardware port is not valid
3	Operation Time Out
4	Error reading internal EEPROM
5	Error initializing HDMI output
6	The time-limited license has expired
1001	Unable to open the USB driver
1002	Error on USB “DeviceIOControl” communication
1003	Unable to write to USB port
1004	Unable to read from USB port
1005	Invalid USB port (Not USB 2.0)
1006	Invalid device connected to the USB port (not IDT)
1007	The device has been disconnected from the USB cable.
2001	Ethernet Image segment is out of order
2002	Error on read from Ethernet port
2003	Error on select from Ethernet port
2004	Error opening Ethernet socket
2005	Error writing Ethernet socket options
2006	Error on Bind to Ethernet port
2007	Error sending commands through the Ethernet port
3001	Unable to Open Camera Link Serial Port
3002	Unable to write to Camera Link Serial Port
3003	Unable to read from Camera Link Serial Port
4001	Unable to open virtual camera (SD card, RAW file)
4002	Unable to read from virtual camera (SD card, RAW file)

7.3. Appendix C – Information Parameters

The following table shows the values and a brief description of the parameters that can be read calling the **XsGetCameraInfo** routine. The numeric values of the parameters can be found in the **XStrmAPI.h** header file in the **Include** sub-directory.

Parameter	Value	Description
XSI_CAMERA_MODEL	0	Camera Model (see XS_CAM_TYPE in XStrmAPI.h)
XSI_CAMERA_ID	1	Camera ID (see XS_ENUMITEM structure in XStrmAPI.h)
XS_FW_VERSION	2	EEPROM Firmware version
XSI_MEMORY	3	Camera on-board memory size
XSI_REC_SIZE_SELECTABLE	4	The user can set the record length (TRUE/FALSE)
XSI_COOLED	5	The camera has a cooler (TRUE/FALSE)
XSI_SERIAL	6	The camera serial number (10 decimal digits value)
XSI_REVISION	7	The camera hardware revision (A, B, C, D, etc.)
XSI_DACS_COUNT	8	The number of DAC of camera (depends on model)
XSI_INT_CLOCK	9	The camera internal clock rate [Hz]
XSI_EXTRA_ROWS	10	Read out extra rows
XSI_ROW_CLOCKS	11	Number of read out clocks for each image row
XSI_SNS_TYPE	12	Sensor Type (0:monochrome, 1: color)
XSI_SNS_MODEL	13	Sensor Model (see XS_SNS_MODEL in XStrmAPI.h)
XSI_SNS _WIDTH	14	Sensor Maximum width [pixels]
XSI_SNS _HEIGHT	15	Sensor Maximum Height [pixels]
XSI_CFW_VERSION	16	Controller Firmware version
XSI_LIVE_WHILE_REC	17	The camera supports Live While Record
XSI_SOFT_TRIGGER	18	The camera supports the software trigger
XSI_FCAL_FILE_ON	19	The camera has a factory calibration data file
XSI_IRIG	20	The camera supports IRIG
XSI_BNC_CONNECTORS	21	The number of BNC connectors on the camera (2 or 3)
XSI_EXP_STEPS	22	The number of exposure steps
XSI_INTENSIFIED	23	The camera is intensified
XSI_LINK	24	The camera link type (USB 2.0 or Giga Ethernet)
XSI_MIN_EXP	25	The camera minimum exposure (1 µs or 100 ns)
XSI_MISC_CAPS	26	Camera miscellaneous capabilities
XSI_CFW_CLK_1	27	The camera controller timing parameter (deprecated)

XSI_INT_REG_0,1	28,29	The camera internal registers values
XSI_LIGHT	30	The camera is Light (reduced capabilities in resolution speed, etc.)
XSI_GIGA_ETHERNET	31	The camera has both USB 2.0 and Giga-Ethernet link
XSI_FLASH_MEMORY	32	The camera has onboard flash memory
XSI_VIDEO_MODE	33	The camera has configurable video output mode
XSI_PLUS	34	The camera has the Plus™ capability, i.e. it can acquire at double speed.
XSI_CAMERA_NAME	35	The camera name. The default name is build with the serial number. if the camera has the flash memory, the name is stored in the flash memory.
XSI_LINK_FW_VERSION	36	The link firmware version. For GE is the controller firmware version, for USB is 2.0
XSI_CFW2_VERSION	37	The firmware version of the second controller (only on cameras with USB+GE connector)
XSI_ROI_MIN	38	Minimum allowed ROI size (Width, Height)
XSI_ROI_STEP	39	ROI step size (X, Y)
XSI_FG_TYPE	40	The camera link frame grabber type (M-series)
XSI_CCAL_FILE_ON	41	The calibration file with current condition data is present in the hard disk.
XSI_FCAL_FILE_FLASH	42	The factory calibration file is stored in the camera flash memory (Yes/No)
XSI_CAL_PATH	43	The path to the camera calibration files directory
XSI_CFA_PATTERN	44	The Color Filter array pattern (0:GRBG, 1:BGGR, 2:RGGB, 3:GBRG)
XSI_MAX_FRM_SIZE	45	The maximum size of a frame in camera memory (size in bytes)
XSI_NEW_DESIGN	46	The camera hardware is redesigned with internal background and image processing.
XSI_CAL_NAME	47	The camera calibration file name
XSI_MOTION_TRIG	48	The camera supports motion trigger
XSI_WRITE_2_DISK	49	The camera supports direct write to disk option (M only)
XSI_COMPRESSION	50	The camera supports compression (N cameras)
XSI_DGR_SIZE	51	The camera supports configurable datagram size
XSI_AUTO_EXPOSURE	52	The camera supports the auto-exposure
XSI_POST_REC_OP	53	The camera supports the post-recording operation
XSI_MARKER	54	The camera supports the marker option
XSI_SUB_MODEL	55	The camera sub-model (0: undefined, 1 and above: sub-model)
XSI_SNS_PIX_DEPTH	56	The sensor original pixel depth (8,10 or 12)
XSI_XDR_SUPPORT	57	The camera supports the XDR mode
XSI_EDR_SUPPORT	58	The camera supports the EDR mode

XSI_LIVE_BUF_SIZE	59	The size of the buffer that should be reserved for Live at the beginning of the camera memory. It guarantees that the live images will not overwrite previous acquisitions.
XSI_RESIZE	60	The camera supports the resize option (for fast thumbnail preview).
XSI_HW_BOARDS	61	The number of boards listed in the camera hardware configuration
XSI_HW_INFO	62	The information of the boards in the camera hardware configuration
XSI_TNR_SUPPORT	63	The camera supports Temporal Noise Reduction
XSI_PIV_READY	64	The camera is PIV ready
XSI_CUR_CAL	65	The camera supports “current conditions” calibration
XSI_JPEG_SUPPORT	66	The camera supports JPEG encoding on images
XSI_DNR2_SUPPORT	67	The camera supports DNR with TNK
XSI_HW_BROC_SUPPORT	68	The camera supports hardware BROC
XSI_ASYNC_VIDEO_PB	69	The camera supports asynchronous playback on video (HDMI) output.
XSI_EX_ROI_CNT	70	The number of extended ROI
XSI_EX_ROI_MAX_WID	71	The maximum width of extender ROI
XSI_EX_ROI_MAX_HGT	72	The maximum height of extended ROI
XSI_1PPS_SYNC_SUPPORT	73	The camera supports the 1PPS sync mode
XSI_BATTERYSTS_SUPPORT	74	The camera supports battery status and level readout
XSI_SSD_SIZE	75	Size of onboard SSD
XSI_FAST_LIVE	76	The camera supports fast live
XSI_PTP_SUPPORT	77	The camera supports PTP.
XSI_TEMP_SUPPORT	78	The camera supports temperature read
XSI_CI_THR_SUPPORT	79	The camera supports the Color Interpolation threshold
XSI_SI_PLL_SUPPORT	80	The camera supports the PLL mode (external phase lock loop synchronization)
XSI_SI_DPULSE_SUPPORT	81	The camera supports the dynamic pulse mode (external pulse mode synchronization with variable width)
XSI_LENS_SUPPORT	82	The camera supports motorized control of lens focus and iris.
XSI_BATTERY_INFO	83	The battery information (manufacturing date)
XSI_SHOCK_SNS_SUPPORT	84	The camera has a shock sensor on board
XSI_PAINT_SUPPORT	85	The camera supports paint parameters
XSI_DEFLICKER_SUPPORT	86	The camera supports deflicker
XSI_PTP_MODE_SUPPORT	87	The camera supports PTP mode
XSI_LENS_INFO	88	Lens information (see XS_LENS_INFO)

XSI_USB_VID	100	USB specific: Vendor ID.
XSI_USB_PID	101	USB specific: Product ID.
XSI_USB_PORT	102	USB specific: Port number.
XSI_GE_CAM_MACADD	200	Gigabit Ethernet specific: camera MAC address
XSI_GE_CAM_IPADD	201	Gigabit Ethernet specific: camera IP address
XSI_GE_ADP_MACADD	202	Gigabit Ethernet specific: network adapter MAC address
XSI_GE_ADP_IPADD	203	Gigabit Ethernet specific: network adapter IP address
XSI_GE_ADP_NETMASK	204	Gigabit Ethernet specific: network adapter sub-net mask
XSI_GE_CAM_NETMASK	205	Gigabit Ethernet specific: camera sub-net mask
XSI_GE_ADP_MTU	206	Gigabit Ethernet specific: network adapter MTU

7.4. Appendix D – Camera Parameters

The following table shows the values and a brief description of the parameters that can be read and written in the camera. The numeric values of the parameters can be found in the **XStrmAPI.h** header file in the **Include** sub-directory.

Parameter	Value	R/W	Description
XSP_GAIN	0	R/W	Camera sensor gain (increases sensitivity)
XSP_EXPOSURE	1	R/W	Camera exposure in nanoseconds [ns]
XSP_IMG_FORMAT	2	R/W	Image format (gray8, gray16, BGR24, BGRA32, ARGB32)
XSP_PIX_DEPTH	3	R/W	Pixel depth (8,9,10,11,12 or 24,27,30,33,36)
XSP_PIX_GAIN	4	R/W	Pixel Gain (1x, 2x, 4x)
XSP_SYNCIN_CFG	5	R/W	Sync In configuration (internal, pulse/edge, high/low, IRIG, GPS)
XSP_REC_MODE	6	R/W	Record Mode (normal/circular/BROC)
XSP_EXP_MODE	7	R/W	Exposure Mode (single/double)
XSP_BINNING	8	R/W	Binning (1x1, 2x2, 3x3, 4x4)
XSP_CI_MODE	9	R/W	Color Interpolation algorithm – color cameras only
XSP_TRIGIN_CFG	10	R/W	Trigger In configuration (edge hi/lo, switch closure, disabled)
XSP_MAX_WIDTH	11	R	Maximum image width
XSP_MAX_HEIGHT	12	R	Maximum image height
XSP_ROIX	13	R/W	Upper left x coordinate of ROI
XSP_ROIY	14	R/W	Upper left y coordinate of ROI
XSP_ROIWIDTH	15	R/W	Width of ROI, in pixels
XSP_ROIHEIGHT	16	R/W	Height of ROI, in pixels
XSP_PERIOD	17	R/W	Frame period in nanoseconds (inverse of frame rate)
XSP_PERIOD_MIN	18	R	Minimum frame period [ns] (inverse of maximum frame rate)
XSP_IRIG	19	R/W	Enable/Disable IRIG/GPS on camera (0/1)
XSP_SYNCOUT_CFG	20	R/W	Sync Out configuration
XSP_TRIGIN_DEB	21	R/W	Trigger In De-bounce (avoids detecting spikes in the line as trigger)
XSP_TRIGIN_DEL	22	R/W	Trigger In Delay
XSP_ SYNCOUT_WID	23	R/W	Sync Out Configurable Width
XSP_ SYNCOUT_DEL	24	R/W	Sync Out Delay
XSP_ SYNCIN_DEB	25	R/W	Sync In De-bounce

XSP_ SYNCIN_DEL	26	R/W	Sync In Delay
XSP_HDMI_MODE	27	R/W	HDMI/SDI output mode (disabled, enabled or both computer and monitor)
XSP_PREV_MODE	28	R/W	Preview mode (Full resolution / Low resolution chroma subsampling)
XSP_NOISE_RED	30	R/W	Noise background removal flag (on/off)
XSP_NOISE_SENS	31	R/W	Pixel sensitivity correction flag (on/off)
XSP_NOISE_DKCOL	32	R/W	Enable the use of masked (dark) columns to reduce noise (on/off)
XSP_NOISE_AUTO	33	R/W	Background with current settings (on/off)
XSP_VIDEO_MODE	34	R/W	Output video mode: PAL/NTSC for X cameras, 1080p/720p for other Y/Os/CC cameras
XSP_NET_PERFORM	35	R/W	Network performance of the Giga-Ethernet connection.
XSP_PLUS	36	R/W	Enable/disable Plus™ Mode (On/Off)
XSP_GAMMA	37	R/W	The image gamma correction parameter (0 to 30)
XSP_FRAMES	38	R	The number of frames acquired in the latest recording
XSP_PRE_TRIG	39	R	The number of pre-trigger frames acquired in the latest recording.
XSP_BROC_LEN	40	R/W	The length of each BROC segment
XSP_FRAME_CAP	41	R	The camera frame capacity
XSP_1ST_FRM_IDX	42	R	The index of the first frame of the latest acquisition
XSP_STARTADDRLO	43	R	The starting address of the latest acquisition (low part)
XSP_STARTADDRHI	44	R	The starting address of the latest acquisition (high part)
XSP_NOISE_APSC	45	R/W	Enable the pixel sensitivity correction computed in current conditions
XPS_EXT_PERIOD	46	R	Reads the external sync signal period
XSP_EXT_PULSE_WID	47	R	Reads the external sync signal pulse width
XSP_CMP_RATIO	48	R/W	Set the N camera compression ratio (40 to 100)
XSP_FRAME_SIZE	49	R	Reads the size of a frame in camera memory [bytes]
XSP_EXPOSURE_MAX	50	R	Maximum exposure allowed in current conditions [ns]
XSP_EXPOSURE_DBL	51	R	Value of second exposure in double exposure mode [ns]
XSP_XDR_RATIO	52	R/W	Extended Dynamic Range (XDR) ratio [2 to 8] (Y4 camera only)

XSP_XDR_CONTRAST	53	R/W	XDR contrast [1 to 100] (Y4 camera only)
XSP_DYNAMIC_NR	54	R/W	Dynamic Noise reduction coefficient [0 to 30]
XSP_SHARPEN	55	R/W	Sharpen [0 to 10]
XSP_BRIGHTNESS	56	R/W	Brightness [0 to 50]
XSP_CONTRAST	57	R/W	Contrast [0 to 20]
XSP_HUE	58	R/W	Hue [0 to 360]
XSP_SATURATION	59	R/W	Saturation [0 to 20]
XSP_WB_11	60	R/W	White balance matrix value 1,1 (Blue gain)
XSP_WB_12	61	R/W	White balance matrix value 1,2
XSP_WB_13	62	R/W	White balance matrix value 1,3
XSP_WB_21	63	R/W	White balance matrix value 2,1
XSP_WB_22	64	R/W	White balance matrix value 2,2 (Green gain)
XSP_WB_23	65	R/W	White balance matrix value 2,3
XSP_WB_31	66	R/W	White balance matrix value 3,1
XSP_WB_32	67	R/W	White balance matrix value 3,2
XSP_WB_33	68	R/W	White balance matrix value 3,3 (Red gain)
XSP_BOARD_TEMP	69	R	Read sensor board temperature (in 1/100 of degrees C).
XSP_MT_CFG	70	R/W	Motion Trigger modes
XSP_MT_THRESHOLD	71	R/W	Motion Trigger threshold (1 to 400)
XSP_MT_ROIX	72	R/W	Motion Trigger area X coordinate
XSP_MT_ROIY	73	R/W	Motion Trigger area Y coordinate
XSP_MT_ROIWIDTH	64	R/W	Motion Trigger area width
XSP_MT_ROIHEIGHT	75	R/W	Motion Trigger area height
XSP_DGR_SIZE	76	R/W	Datagram size
XSP_AE_ENABLE	77	R/W	Enable auto-exposure
XSP_AE_ROIX	78	R/W	Auto-exposure ROI X
XSP_AE_ROIY	79	R/W	Auto-exposure ROI Y
XSP_AE_ROIWIDTH	80	R/W	Auto-exposure ROI Width
XSP_AE_ROIHEIGHT	81	R/W	Auto-exposure ROI Height
XSP_AE_REFERENCE	82	R/W	Auto-exposure luminance reference
XSP_AE_SPEED	83	R/W	Auto-exposure reaction speed
XSP_AE_EXPOSURE	84	R	Auto-exposure current value (on each frame)
XSP_PROP	85	R/W	Enable Post-recording operation
XSP_PROP_WR_BLK	86	R/W	Number of memory blocks (512 bytes each) to write in the post-recording operation

XSP_PROP_ABORT	87	R	Abort the post-recording operation
XSP_SYNCOUT_ALIGN	88	R/W	Sync Out Alignment (auto-exposure)
XSP_MARKER_CFG	89	R/W	Configures marker input
XSP_MARKER_VAL	90	R	Read the marker from current frame
XSP_CI_THR	91	R/W	Color Interpolation threshold
XSP_CLOCK_SPEED	92	R/W	Clock speed (Y5 only)
XSP_HD_ROI	93	R/W	ROI index for HD cameras
XSP_GAUSS_FLT	94	R/W	Gaussian Filter Kernel (0: disabled – 16: max anti-alias effect)
XSP_ROT_ANGLE	95	R/W	Rotation angle (0, 90, 180, 270)
XSP_FLIP	96	R/W	Flip horizontally, vertically or both sides
XSP_LUT	97	R/W	Lookup Table selection
XSP_LUTCHN_MASK	98	R/W	Lookup Table channels mask (RGB)
XSP_HDMI_OVERLAY	99	R/W	Enables the data overlay on the HDMI output (0:OFF 1:ON)
XSP_WBTBL_TEMP	100	R/W	The current white balance table color temperature (0 for user table)
XSP_HD_ZOOM	101	R/W	The value of the digital zoom (from 1x to 8x)
XSP_HD_ZOOM_X	102	R/W	The origin X coordinate of the digital zoom
XSP_HD_ZOOM_Y	103	R/W	The origin Y coordinate of the digital zoom
XSP_SHARPEN_THR	104	R/W	The sharpening filter threshold
XSP_SYNCIN_INV	105	R/W	The “sync in” polarity inversion
XSP_DYNAMIC_NR2	106	R/W	Additional Dynamic Noise Reduction
XSP_JPEG	107	R/W	Enables JPEG encoding on camera
XSP_IMG_ZOOM	108	R/W	Sets the image zoom for live and playback
XSP_IMG_ZOOM_WID	109	R	Returns the image width after zoom
XSP_IMG_ZOOM_HGT	110	R	Returns the image height after zoom
XSP_AE_CUR_LUMA	111	R	Returns the current value of luminance in the auto-exposure window
XSP_BROC_TOT_LEN	112	R/W	The total number of frames acquired in a BROC session
XSP_BROC_CURR_SECT	113	R	The current BROC section.
XSP_HD_ZOOM_2R	114	R/W	This parameter disables the HD zoom on Y7 and keeps the original ROI
XSP_FRAME_SIZE_IMG	115	R	This parameter returns the size of the frame when transferred as an image.
XSP_SYNC_COUNT	116	R	This parameter returns the total number of sync pulses generated in the latest acquisition
XSP_AE_MIN_EXP	117	R/W	Value of minimum exposure in auto-exposure mode.

XSP_AE_MAX_EXP	118	R/W	Value of maximum exposure in auto-exposure mode.
XSP_BATTERY_STATUS	119	R	Battery status, supported by NX-Air only
XSP_JPEG_QUAL	120	R/W	Jpeg compression quality (1 to 100)
XSP_SSD_STRM_PER	121	R	The inverse of the SSD streaming fps
XSP_SSD_MAX_FRMS	122	R	The maximum number of frames that can be recorded in streaming mode on the current frame rate
XSP_GAMMA_LEVEL	123	R/W	First gamma level
XSP_GAMMA1	124	R/W	Second gamma value
XSP_GAMMA_LEVEL1	125	R/W	Second gamma level
XSP_MISS_STEP_CNT	126	R	Number of mission steps that have been configured
XSP_MISS_EXEC_CNT	127	R	Number of mission steps that have been executed
XSP_EXT_FREQ	128	R	External frequency (Hz)
XSP_RAW_RD_TSTAMP	130	R/W	Valid only on RAW files. Set the parameter to 1 if you want to read IRIG or GPS data from the RAW file without loading the full frame.
XSP_NET_SPEED	131	R/W	Network speed for HG (0:automatic, 1:100Mbps, 2:1000 Mbps)
XSP_AE_EXP_STEP	132	R/W	Auto-exposure step width [ns]
XSP_SYNCIN_JITTER	133	R/W	Returns the jitter in nanoseconds for the external sync signal in PLL mode
XSP_SYNCIN_DIV	134	R/W	Frame rate divider for the external sync signal in PLL mode.
XSP_IRIG_GPS_JITT	135	R/W	IRIG/GPS 1 PPS signal jitter in nanoseconds.
XSP_LENS_FOCUS	136	R/W	Motorized lens focus position in cm
XSP_LENS_IRIS	137	R/W	Motorized lens iris aperture in F# x 100
XSP_RAW_MIN_IDX	138	RW	Returns the minimum index of a raw file. It is useful to determine the position of the saved raw sequence from the trigger frame.
XSP_SHOCK_THR	139	RW	Shock sensor threshold in mg. The shock is considered a trigger if the acceleration is above this value
XSP_SHOCK_TIME	140	RW	Shock sensor max duration. If the acceleration is above the threshold for a time that is equal or larger than this parameter, the trigger is not issued.
XSP_LATEST_FRM_IDX	141	R	Index of the latest acquired frame. Read this parameter when the camera is recording.
XSP_LENS_ZOOM	142	R/W	Motorized lens zoom position [mm]

XSP_PAINT_GAIN_R	143	R/W	Paint parameter: Gain of R component (CC-Mini, Xstream and R)
XSP_PAINT_GAIN_G	144	R/W	Paint parameter: Gain of G component (CC-Mini, Xstream and R)
XSP_PAINT_GAIN_B	145	R/W	Paint parameter: Gain of B component (CC-Mini, Xstream and R)
XSP_PAINT_OFFS_R	146	R/W	Paint parameter: offset (pedestal) of R component (CC-Mini, Xstream and R)
XSP_PAINT_OFFS_G	147	R/W	Paint parameter: offset (pedestal) of G component (CC-Mini, Xstream and R)
XSP_PAINT_OFFS_B	148	R/W	Paint parameter: offset (pedestal) of B component (CC-Mini, Xstream and R)
XSP_DEFLICKER	149	R/W	Deflicker parameter
XSP_PTP_MODE	150	R/W	PTP mode for CC-Mini (0:Ethernet 1:UDP).
XSP_DEFLICKER_THR	151	R/W	Deflicker threshold
XSP_PTP_DELAY	152	R/W	PTP power up delay in seconds
XSP_LENS_FOCUS_REL	153	R/W	Motorized lens relative focus movement.
XSP_LENS_CMD	154	W	Send a command to the motorized lens

7.5. Appendix E – Camera Announcements

The cameras will normally generate traffic only in response of specific commands. In specific situations it will be necessary for the camera to autonomously send a message called an Announcement. The table below lists the available announcement for USB and GigaBit-Ethernet cameras.

Code	Link	Notes
DISCONNE	USB	The camera has been disconnected from the USB cable
DISCONNE_[ipaddr]	GE	The camera has been disconnected from the Ethernet cable. The announcement string is followed by underscore and the IP address in square brackets.
RESTORED	USB	The lost connection has been restored. This announcement is sent only if a previous “DISCONNE” message.
RESTORED_[ipaddr]	GE	As above. The announcement string is followed by underscore and the IP address in square brackets.
REBOOTED	USB	The lost connection has been restored but the camera has been rebooted. The software should read the configuration to the camera and reset it.
REBOOTED_[ipaddr]	GE	As above. The announcement string is followed by underscore and the camera IP address in square brackets.
DETACHED	USB	The control of the camera has been taken by another computer and the camera has been detached
DETACHED_[ipaddr]	GE	As above. The announcement string is followed by underscore and the IP address in square brackets.
IDLEMODE	USB	The camera is in idle mode (not recording).
IDLEMODE_[ipaddr]	GE	As above. The string is followed by underscore and the IP address in square brackets.
PRE_TRIG	USB	The camera is recording and waiting for trigger.
PRE_TRIG_[ipaddr]	GE	As above. The string is followed by underscore and the IP address in square brackets.
POSTTRIG	USB	The camera has received a trigger and it is recording post-trigger frames.
POSTTRIG_[ipaddr]	GE	As above. The string is followed by underscore and the IP address in square brackets.
DISKSAVE_[ipaddr]	GE	The camera is saving data to the SSD. The string is followed by underscore and the IP address in square brackets.

The announcement string for HG cameras is shown below:

#0101A000_[10.10.10.100]

BYTE 0 is the '#' character, BYTE 1 and BYTE 2 are the camera ID. BYTE 3 and BYTE 4 are '01'. The announcement code is displayed in BYTE 5 and BYTE 6 of the string. BYTE 7 and BYTE 8 contain the status code only if the announcement is A4. BYTE 9 is the underscore character and the following bytes contain the camera IP address.

The table below shows announcements for HG cameras.

Code	Cam	Link	Notes
'A0'	HG	ETH	The camera is detached. Another computer has taken control of the camera.
'A1'	HG	ETH	The camera has just established a network connectivity
'A2'	HG	ETH	The camera temperature has exceeded the normal operational limits
'A3'	HG	ETH	The camera is losing power and will be entering battery backup mode
'A4'	HG	ETH	The camera has changed state. The string contains the current state in bytes 7 and 8: 00: unknown. 01: standby. 02: live. 03: pre-trigger recording. 04: post-trigger recording. 05: record done (images in memory). 06: download. 11: offline (disconnected).
'A5'	HG	ETH	A root hub has not been detected in the sync/trigger bus.
'A6'	HG	ETH	The camera has detected an error/ fault condition
'A7'	HG	ETH	The camera has completed a configuration update
'A8'	HG	ETH	The camera has restored a disconnection

7.6. Appendix F – Data types

This appendix describes the data types defined in the **XStrmAPI.h** header file.

7.6.1. XS_CAM_MODEL

The XS_CAM_MODEL type enumerates the camera models.

- **XS_CM_UNKNOWN**: Unknown camera model
- **XS_CM_MP_X1**: MotionPro X-1 (was XS-3).
- **XS_CM_MP_X4**: MotionPro X-4 (was XS-4).
- **XS_CM_MP_X3**: MotionPro X-3 (was XS-5).
- **XS_CM_MP_X5**: MotionPro X-5 (was XS-6).
- **XS_CM_MP_X2**: MotionPro X-2 (was XS-7).
- **XS_CM_MP_M3**: MotionScope M-3 (CameraLink).
- **XS_CM_MP_M4**: MotionScope M-4 (not used).
- **XS_CM_MP_M5**: MotionScope M-5 (CameraLink).
- **XS_CM_MP_Y3**: MotionPro Y-3.
- **XS_CM_MP_Y4**: MotionPro Y-4.
- **XS_CM_MP_Y5**: MotionPro Y-5.
- **XS_CM_HG_100K**: MotionXtra HG-100K.
- **XS_CM_HG_LE**: MotionXtra HG-LE.
- **XS_CM_HG_TH**: MotionXtra HG-TH.
- **XS_CM_HG_2K**: Legacy MotionXtra HG-2000.
- **XS_CM_CR_2K**: Legacy MotionXtra CR-2000.
- **XS_CM_TX_2K**: Legacy MotionXtra TX-2000.
- **XS_CM_MP_N3**: MotionXtra N/NR-3.
- **XS_CM_MP_N4**: MotionXtra N/NR/NX-4.
- **XS_CM_MP_N5**: MotionXtra N/NR/NX-5.
- **XS_CM_MP_Y6**: MotionPro Y-6.
- **XS_CM_MP_Y7**: MotionPro Y-7.
- **XS_CM_MP_N7**: MotionXtra N-7.
- **XS_CM_MP_Y8**: MotionPro Y-8.
- **XS_CM_MP_N8**: MotionXtra N-8.
- **XS_CM_MP_Y10**: MotionPro Y-10.
- **XS_CM_MP_O9**: MotionXTra Os-3.

- **XS_CM_MP_O4**: MotionXTra Os-4.
- **XS_CM_MP_O5**: MotionXTra Os-5.
- **XS_CM_MP_O7**: MotionXTra Os-7.
- **XS_CM_MP_O8**: MotionXTra Os-8.
- **XS_CM_CC_1060**: Crash-Cam 1060.
- **XS_CM_CC_1520**: Crash-Cam 1520.
- **XS_CM_CC_1540**: Crash-Cam 1540.
- **XS_CM_CC_4010**: Crash-Cam 4010.
- **XS_CM_CC_M1510**: CrashCam Mini 1510.
- **XS_CM_CC_2020**: CrashCam 2020.
- **XS_CM_CC_M5K05**: CrashCam Mini 5K05.
- **XS_CM_MP_O3**: MotionXtra Os3.
- **XS_CM_CC_M1520**: CrashCam Mini 1520.
- **XS_CM_CC_M3510**: CrashCam Mini 3510.
- **XS_CM_MINI_HD**: CrashCam Mini HD/POV.
- **XS_CM_CC_M1540:** CrashCam Mini 1540.
- **XS_CM_CC_M3525:** CrashCam Mini 3525.
- **XS_CM_PCIE_X7**: PCIe XStream 720p.
- **XS_CM_PCIE_X14**: PCIe XStream 1440p.
- **XS_CM_XSM_1540**: XStream Mini 1540.
- **XS_CM_XSM_3520**: XStream Mini 3520.
- **XS_CM_XSM_STICK**: XStream Stick.
- **XS_CM_XSM_5K**: XStream Mini 5K.
- **XS_CM_XSM_4KV**: XStream Mini 4K-Veloce.
- **XS_CM_R_HD**: R-series HD.
- **XS_CM_CC_STICK**: CrashCam Stick.

7.6.2. XS_ENUM_FLT

The XS_ENUM_FLT enumerates the camera types.

- **XS_EF_USB_X**: MotionPro X on USB 2.0.
- **XS_EF_GE_X**: MotionPro X on Giga Ethernet.
- **XS_EF_HG**: MotionXtra HG on Giga Ethernet.
- **XS_EF_CL**: MotionScope M on Camera Link.
- **XS_EF_USB_Y**: MotionPro Y on USB 2.0.

- **XS_EF_GE_Y**: MotionPro Y on Giga Ethernet.
- **XS_EF_LG_RL**: MotionXtra Legacy on Giga Ethernet.
- **XS_EF_GE_N**: MotionXtra N/NR/NX on Giga Ethernet.
- **XS_EF_GE_NO**: MotionXtra N/NR/NX/O on Giga Ethernet.
- **XS_EF_PCI_X**: X-Stream PCIe on PCIe or Thunderbolt.
- **XS_EF_VCAM**: RAW virtual cameras (RAW files).
- **XS_EF_VSSD**: SSD virtual camera (O camera removable SSD).

7.6.3. XS_LINK_TYPE

The XS_LINK_TYPE type enumerates the links.

- **XS_LT_USB20**: USB 2.0 link.
- **XS_LT_GIGAETH**: Giga Ethernet (1 Mbps).
- **XS_LT_CAMLINK**: CameraLink (Frame Grabber).
- **XS_LT_SDCARD**: SD card or removable SSD.
- **XS_LT_RAWFILE**: Raw File (fast download).
- **XS_LT_WIFI**: Wi-Fi (Ethernet)
- **XS_LT_PCIE**: PCI Express.
- **XS_LT_TB**: Thunderbolt.
- **XS_LT_10GIGAETH**: 10 Gigabit Ethernet

7.6.4. XS_SNS_TYPE

The XS_SNS_TYPE type enumerates the sensor types.

- **XS_ST_MONOCHROME**: monochrome sensor.
- **XS_ST_COLOR**: color sensor.

7.6.5. XS_CFA_PATTERN

The XS_CFA_PATTERN type enumerates the color filter array patterns (color cameras).

- **XS_CFAP_GRBG**: GRBG pattern.
- **XS_CFAP_BGGR**: BGGR pattern.
- **XS_CFAP_RGGB**: RGGB pattern.
- **XS_CFAP_GBRG**: GBRG pattern.

7.6.6. XS_FG_TYPE

The XS_FG_TYPE type enumerates the camera link frame grabber types (M-series).

- **XS_FG_COR_X64CL**: Dalsa-Coreco X64 Xcelera-CL PX4.
- **XS_FG_NI_PCIE1429**: National Instruments PCIe-1429.
- **XS_FG_MATROX_H_S**: Matrox Helios/Solios.
- **XS_FG_EPIXCI_E4**: Epix PIXCI E4.
- **XS_FG_BF_KARBON**: Bitflow Karbon-CL.

7.6.7. XS_SNS_MODEL

The XS_SNS_MODEL type enumerates the sensor models.

- **XS_SM_UNKNOWN**: unknown sensor model.
- **XS_SM_MV13**: MV-13 for X-Stream XS3 cameras.
- **XS_SM_MV02**: MV-02 for HS4 and X4 cameras.
- **XS_SM_RL_LEGACY**: Redlake legacy for HG2000, CR, TX cameras.
- **XS_SM_MAKO**: MAKO for HG100K, HG-LE and HG-TH cameras
- **XS_SM_SYRIUS**: Sirius for HS3, X3, M3 and N3 cameras.
- **XS_SM_ORION**: Orion for X5, M5, Y5 and N5 cameras.
- **XS_SM_OTION_II**: Orion II for new Y5, M5 and N5 cameras.
- **XS_SM_NORTH_STAR**: North Star for Y4 cameras
- **XS_SM_NORTH_STAR_II**: North Star II for new Y4 and N4 cameras
- **XS_SM_NOZOMI**: Nozomi for Y6 cameras.
- **XS_SM_PEGASUS**: Pegasus for Os10 cameras.
- **XS_SM_SIRIUS_II**: Sirius II for Os7.
- **XS_SM_GEMINI**: Gemini for Os9.
- **XS_SM_LEO**: Leo for Os16.
- **XS_SM_STK**: Stick Sensor.
- **XS_SM_RCHD**: R-series HD sensor.
- **XS_SM_4KVEL**: XS-MINI 4K Veloce sensor.

7.6.8. XS_REVISION

The XS_REVISION type enumerates the camera revision numbers.

- **XS_REV_A**: revision A (original).
- **XS_REV_B, C, D**: revision B, C, D, etc.

7.6.9. XS_MISC_CAPS

The XS_MISC_CAPS type enumerates miscellaneous capabilities of the camera.

- **XS_CAP_NR**: the camera is an NR.
- **XS_CAP_NX**: the camera is an NX.
- **XS_CAP_NXT**: the camera is an NX-Tra.
- **XS_CAP_NXA**: the camera is an NX-Air.
- **XS_CAP_DNR2**: the camera supports DNR.
- **XS_CAP_HWBROC**: the camera supports Hardware BROC.
- **XS_CAP_JPEG**: the camera supports JPEG.
- **XS_CAP_1PPS**: the camera supports 1PPS input and output.
- **XS_CAP_BATSTS**: the camera supports Battery status read.
- **XS_CAP_FBCAM**: the camera is a FB model.
- **XS_CAP_PIV**: the camera has the PIV option.
- **XS_CAP_OS**: the camera is an Os (sealed).
- **XS_CAP_GPSMOD**: the camera has the internal GPS module.
- **XS_CAP_INX**: the camera is an industrial NX camera.
- **XS_CAP_JPLROC**: the camera is a JPL ROC model
- **XS_CAP_PTP**: the camera supports PTP.
- **XS_CAP_IS1024**: the camera is an N4/NR4/NX4 that supports the 1024x1024 resolution.
- **XS_CAP_OS3**: the camera is an Os version 3.
- **XS_CAP_OSA**: the camera is an Os Airborne.
- **XS_CAP_PLL**: the camera supports the Phase Lock Loop mode.
- **XS_CAP_IRIGMD**: the camera has the internal IRIG module.
- **XS_CAP_SDI_FW**: the camera is a CC mini HD/POV with a special firmware for SDI output.
- **XS_CAP_24GB**: the camera has 24GB onboard DDR.

7.6.10. XS_PRE_PARAM

The XS_PRE_PARAM type enumerates the pre-configuration parameters**.**

- **XSPP_IP_ADDRESS**: the camera IP address (GE models only).
- **XSPP_NET_AD_IP**: the network adapter IP address (GE models only)
- **XSPP_IP_ADD_EX**: the camera IP address (extended – GE models only)
- **XSPP_CAM_CMD_PORT**: the camera command port (extended – GE models only)

- **XSPP_NET_ADD_CMD_PORT**: the application command port (extended – GE models only)
- **XSPP_GET_IP_ADDRESS**: read the camera IP address and sub-net mask (extended – HG models only).
- **XSPP_DB_FOLDER**: set the path to the database folder (used to enumerate RAW files virtual cameras).
- **XSPP_CAM_DFL_GW**: camera default gateway (not active yet).
- **XSPP_DISABLE_1024**: disable 1024x1024 resolution on Y4, N4, NR4 and NX4.
- **XSPP_REBOOT_FW**: reboot camera firmware
- **XSPP_PCIX_DMASIZE**: size of DMA buffer for X-Stream PCIe camera.

7.6.11. XS_STATUS

The XS_STATUS enumerates the camera status:

- **XSST_UNKNOWN**: unknown status.
- **XSST_IDLE**: the camera is idle (ready to operate).
- **XSST_LIVE**: the camera is in live mode (HG-only).
- **XSST_REC_PRETRG**: the camera is recording pre-trigger frames.
- **XSST_REC_POSTRG**: the camera is recording post-trigger frames.
- **XSST_REC_DONE**: the camera has recorded (HG-only).
- **XSST_DOWNLOAD**: the camera is downloading images (HG-only).
- **XSST_DISCONNECT**: the camera is disconnected (X and Y only).
- **XSST_DWL_SD**: the camera is downloading data to the SD card in the IRIG-Flash module (Y only).
- **XSST_DWLUPL_SSD**: the camera is downloading/uploading data to/from the SSD.
- **XSST_VPB_ON_COFF**: Video (HDMI) playback is on but the HDMI is disconnected.
- **XSST_VPB_ON_CON**: Video (HDMI) playback is on and the HDMI is connected.
- **XSST_PLAYBACK**: the camera is playing back images to the video output (CC mini HD only with special firmware).

7.6.12. XS_EXP_MODE

The XS_EXP_MODE enumerates the camera exposure modes:

- **XS_EM_SINGLE_EXP**: single exposure.
- **XS_EM_DOUBLE_EXP**: double exposure.
- **XS_EM_EDR**: Extended Dynamic Range with single image (Y4, N4 only).
- **XS_EM_XDR**: eXtended Dynamic Range with double exposure (other cameras).

7.6.13. XS_REC_MODE

The XS_REC_MODE enumerates the camera record modes:

- **XS_RM_NORMAL**: normal acquisition mode.
- **XS_RM_CIRCULAR**: circular acquisition mode.
- **XS_RM_BROC**: burst record on command (HG-only).
- **XS_RM_ROC**: record on command (HG-only).
- **XS_RM_READY**: ready mode waiting for trigger (HG-only).

7.6.14. XS_SYNCIN_CFG

The XS_SYNCIN_CFG enumerates the configuration of the Sync In:

- **XS_SIC_INTERNAL**: internal frame rate acquisition.
- **XS_SIC_EXT_EDGE_HI**: external, exposure starts on edge, active High.
- **XS_SIC_ EXT_EDGE_LO**: external, exposure starts on edge, active Low.
- **XS_SIC_ EXT_PULSE_HI**: external, exposure integrated over pulse, active High.
- **XS_SIC_EXT_PULSE_LO**: external, exposure integrated over pulse, active Low
- **XS_SIC_IRIG_DTS_EXT**: IRIG/DTS mode with external generation of 1pps signal.
- **XS_SIC_IRIG_DTS_INT**: IRIG/DTS mode with internal generation of 1pps signal.
- **XS_SIC_1PPS**: the sync in signal is a 1PPS signal.
- **XS_SIC_PTP**: the sync in signal is PTP.
- **XS_SIC_EPPL_EDGE_HI**: phase lock loop (edge high).
- **XS_SIC_EPPL_EDGE_LO**: phase lock loop (edge low).
- **XS_SIC_EDYN_PULSE_HI**: external dynamic pulse high. Pulse width changes.
- **XS_SIC_EDYN_PULSE_LO**: external dynamic pulse low. Pulse width changes.

7.6.15. XS_SYNCOUT_CFG

The XS_SYNCOUT_CFG enumerates the configuration of the Sync Out for Y cameras:

- **XS_SOC_DFL**: default behavior.
- **XS_SOC_DFL_INV**: default behavior (inverted).
- **XS_SOC_CFGWID**: the sync out signal width is configurable.
- **XS_SOC_CFGWID_INV**: the sync out signal width is configurable and inverted.
- **XS_SOC_DISABLED**: the sync out signal is disabled.
- **XS_SOC_DBLEXP**: the sync out signal reproduces the double exposure pulses.
- **XS_SOC_1PPS**: the camera generates a 1PPS signal on the sync out.

7.6.16. XS_SYNCOUT_ALIGN

The XS_SYNCOUT_ALIGN enumerates the alignment of the Sync Out for Y cameras in auto-exposure mode:

- **XS_SOA_EXP**: the sync out is aligned with the exposure.
- **XS_SOA_SYNC_IN**: the sync out is aligned with the Sync In.

7.6.17. XS_TRIGIN_CFG

The XS_TRIGIN_CFG enumerates the configuration of the Event Trigger:

- **XS_TIC_EDGE_HI**: the trigger starts on edge, active High.
- **XS_TIC_ EDGE_LO**: the trigger starts on edge, active Low.
- **XS_TIC_SWC**: the trigger starts on Switch Closure.
- **XS_TIC_GATE_HI**: the trigger acts as a gate to the acquisition (high level).
- **XS_TIC_GATE_LO**: the trigger acts as a gate to the acquisition (low level).
- **XS_TIC_DISABLED**: the trigger is disabled.

7.6.18. XS_MTRIG_CFG

The XS_MTRIG_CFG enumerates the motion trigger modes:

- **XS_MT_DISABLED**: the motion trigger is disabled.
- **XS_MT_AVG_CHG**: average brightness changes.
- **XS_MT_AVG_INCR**: average brightness increases.
- **XS_MT_AVG_DECR**: average brightness decreases.
- **XS_MT_MOTION**: any motion is detected.

7.6.19. XS_IMG_FMT

The XS_IMG_FMT enumerates the image formats:

- **XS_IF_GRAY8**: gray 8 (8 bit).
- **XS_IF_BAYER8**: Bayer pattern 8 (8 bit).
- **XS_IF_GRAY16**: gray 16 (10 bit).
- **XS_IF_BAYER16**: Bayer pattern 16 (10 bit).
- **XS_IF_BGR24**: Windows 24 bit BGR.
- **XS_IF_BGRA32**: Windows 32 bit BGRA (byte A is not used).
- **XS_IF_ARGB**: MAC 32 bit ARGB (byte A is not used).
- **XS_IF_GRAY8X**: 8 bit grayscale obtained by RGB color image (color cameras only).
- **XS_IF_GRAY16X**: 16 bit grayscale obtained by RGB color image (color cameras only).

- **XS_IF_BGR48**: Windows 48 bit BGR.

7.6.20. XS_CI_MODE

The XS_CI_MODE enumerates the color interpolation algorithms:

- **XS_CIM_BILINEAR**: the bilinear algorithm.
- **XS_CIM_ADVANCED**: advanced algorithm.
- **XS_CIM_ADVANCED2**: advanced algorithm.

7.6.21. XS_SENSOR_GAIN

The XS_SENSOR_GAIN enumerates the sensor digital gains:

- **XS_SG_1_00**: no gain (default).
- **XS_SG_1_41**: gain 1.41 (square root of 2).
- **XS_SG_2_00**: gain 2.
- **XS_SG_2_82**: gain 2.81 (2 times the square root of 2).

7.6.22. XS_PIX_GAIN

The XS_PIX_GAIN enumerates the camera pixel gains:

- **XS_PG_1X**: gain 1x (bits 2 to 9).
- **XS_PG_2X**: gain 2x (bits 1 to 8).
- **XS_PG_4X**: gain 4x (bits 0 to 7).

7.6.23. XS_LUT

The XS_PIX_GAIN enumerates the camera lookup tables:

- **XS_LUT_OFF**: disabled (no LUT applied)
- **XS_LUT_USER**: user-defined lookup table.
- **XS_LUT_A**: lookup table A.
- **XS_LUT_B**: lookup table B.
- **XS_LUT_C**: lookup table C.
- **XS_LUT_D**: lookup table D.
- **XS_LUT_E**: lookup table E.
- **XS_LUT_G14**: Gamma 1.4 lookup table.
- **XS_LUT_G18**: Gamma 1.8 lookup table.
- **XS_LUT_G20**: Gamma 2.0 lookup table.
- **XS_LUT_REC709**: REC 709 lookup table.
- **XS_LUT_BT2020**: BT2020 lookup table.

7.6.24. XS_LUT_MASK

The XS_PIX_GAIN enumerates the lookup table channels mask:

- **XS_LUTMSK_OFF**: disabled (no LUT applied)
- **XS_LUTMSK_R**: enable red channel.
- **XS_LUTMSK_G**: enable green channel.
- **XS_LUTMSK_B**: enable blue channel.
- **XS_LUTMSK_ALL**: all channels are enabled.

7.6.25. XS_BINNING

The XS_BINNING enumerates the binning values.

- **XS_BIN_1X1**: binning 1x1 (default – no binning).
- **XS_BIN_2X2**: binning 2x2.
- **XS_BIN_3X3**: binning 3x3.
- **XS_BIN_4X4**: binning 4x4.

7.6.26. XS_HDMI_MODE

The XS_HDMI_MODE enumerates the HDMI output modes (Y cameras):

- **XS_HDMI_OFF**: the HDMI output is off.
- **XS_HDMI_ON**: the HDMI output is on.
- **XS_HDMI_TRANSFER**: the HDMI output is transferring the acquisition.
- **XS_HDMI_INDEPENDENT**: the HDMI output is independent from the computer.

7.6.27. XS_VIDEO_MODE

The XS_VIDEO_MODE enumerates the video output modes (X, HG and Y cameras):

- **XS_VM_X_PAL**: PAL output for X and HG cameras.
- **XS_VM_X_NTSC**: NTSC output for X and HG cameras.
- **XS_VM_Y_720P_60HZ**: 720p @ 60 Hz output.
- **XS_VM_Y_1080_60HZ**: 1080p @ 60 Hz output.
- **XS_VM_Y_1080_25HZ**: 1080p @ 25 Hz output.
- **XS_VM_Y_1080_24HZ**: 1080p @ 24 Hz output.
- **XS_VM_Y_1080_30HZ**: 1080p @ 30 Hz output.

7.6.28. XS_VIDEO_PB

The XS_VIDEO_PB enumerates the HDMI asynchronous playback modes (Y cameras):

- **XS_VPB_OFF**: the asynchronous playback is off.
- **XS_VPB_FWD**: the asynchronous playback is on and forward.
- **XS_VPB_REW**: the asynchronous playback is on and rewind.

7.6.29. XS_PREV_MODE

The XS_PREV_MODE enumerates the preview modes (Y cameras):

- **XS_PM_FULL_RES**: the preview is done at full resolution.
- **XS_PM_LOW_RES**: the preview is done at half the resolution (Chroma sub sampling).

7.6.30. XS_LIVE

The XS_LIVE enumerates the Live commands

- **XS_LIVE_STOP**: stop the fast live.
- **XS_LIVE_START**: start the fast live.

7.6.31. XS_CALLBACK_FLAGS

The XS_CALLBACK_FLAGS enumerates the Queue callback flags:

- **XS_CF_DONE**: callback is called only when the operation is completed.
- **XS_CF_FAIL**: callback is called only when the operation fails.
- **XS_CF_CBONLY**: install callback only and do not start acquisition.

7.6.32. XS_CALIB_OPCODE

The XS_CALIB_OPCODE enumerates the calibration operations:

- **XS_C_BKG_ALL**: acquire background in optimal conditions.
- **XS_C_FILE_RELOAD**: reload factory calibration coefficients.
- **XS_C_FILE_DOWNLOAD**: download calibration file from the camera.
- **XS_C_CURRENT_BKG**: calibrate background in current conditions.
- **XS_C_CURRENT_PSC**: computes PSC coefficients in current conditions.
- **XS_C_CURRENT_RESET**: reset current conditions coefficients and delete file.
- **XS_C_MEMORY**: perform memory calibration (CC-Mini only).
- **XS_C_ABORT**: abort any of the above operations.

7.6.33. XS_DGR_SIZE

The XS_DGR_SIZE enumerates the size of Ethernet packets (Jumbo packets):

- **XS_DGR_1488**: default size of regular Ethernet packets (no Jumbo).
- **XS_DGR_2888**: 2888 bytes.
- **XS_DGR_4328**: 4328 bytes.
- **XS_DGR_5768**: 5768 bytes.
- **XS_DGR_7208**: 7208 bytes.
- **XS_DGR_8640**: 8640 bytes (jumbo packets maximum size).

7.6.34. XS_PR_OP

The XS_PR_OP enumerates the operations that can be performed by the camera after each acquisition:

- **XS_PR_NOTHING**: no op.
- **XS_PR_DWL_SD**: download the images to the SD card in the Flash-IRIG module.
- **XS_PR_SSD_STREAM**: enable SSD streaming mode.
- **XS_PR_SSD_BACKUP**: enable SSD backup mode.

7.6.35. XS_MARKER_CFG

The XS_MARKER_CFG enumerates the configuration of marker input:

- **XS_MRK_OFF**: no marker.
- **XS_MRK_SYNCIN**: the marker will be detected form the Sync In input.
- **XS_MRK_TRIGIN**: the marker will be detected from the Trigger In input.

7.6.36. XS_CLOCK_SPEED

The XS_CLOCK_SPEED enumerates the camera internal clock speed:

- **XS_CLS_LOWER**: lower clock speed.
- **XS_CLS_MIDDLE**: middle clock speed.
- **XS_CLS_LARGER**: larger clock speed.

7.6.37. XS_HD_ROI

The XS_HD_ROI enumerates the pre-defined regions of interest for the HD cameras:

- **XSHD_RES_00**: first resolution (model-dependent).
- **XSHD_RES_01**: second resolution (model-dependent).

- …
- **XSHD_RES_04**: fourth resolution (model dependent).

7.6.38. XS_HD_ZOOM

The XS_HD_ZOOM enumerates the digital zoom values

- **XSHD_Z_100**: no zoom (default).
- **XSHD_Z_125**: 1.25X zoom.
- **XSHD_Z_150**: 1.5X zoom.
- **XSHD_Z_200**: 2X zoom.
- **XSHD_Z_300**: 3X zoom.
- **XSHD_Z_400**: 4X zoom.
- **XSHD_Z_500**: 5X zoom.
- **XSHD_Z_600**: 6X zoom.
- **XSHD_Z_700**: 7X zoom.
- **XSHD_Z_800**: 8X zoom.
- **XSHD_Z_900**: 9X zoom.
- **XSHD_Z_1000**: 10X zoom.
- **XSHD_Z_1200**: 12X zoom.
- **XSHD_Z_1400**: 14X zoom.
- **XSHD_Z_1600**: 16X zoom.

7.6.39. XS_ROT_ANGLE

The XS_ROT_ANGLE enumerates the available rotations:

- **XS_ROT_0**: no rotation (default).
- **XS_ROT_90**: 90 degrees rotation.
- **XS_ROT_180**: 180 degrees rotation.
- **XS_ROT_270**: 270 degrees rotation.

7.6.40. XS_FLIP

The XS_FLIP enumerates the available flips:

- **XS_FLIP_NONE**: no flip (default).
- **XS_FLIP_HORZ**: horizontal flip.
- **XS_FLIP_VERT**: vertical flip.
- **XS_FLIP_BOTH**: both sides flip.

7.6.41. XS_ATTRIBUTE

The XS_ATTRIBUTE enumerates the attribute types:

- **XS_ATTR_MIN**: minimum value of the parameter.
- **XS_ATTR_MAX**: maximum value of the parameter.
- **XS_ATTR_READONLY**: the parameter is read-only.
- **XS_ATTR_DEFAULT**: the default value of the parameter.

7.6.42. XS_JPEG

The XS_JPEG enumerates the jpeg compression type:

- **XS_JPEG_OFF**: disabled.
- **XS_JPEG_CVT**: if enabled, the data is converted in the driver and returned uncompressed.
- **XS_JPEG_RAW**: if enabled, the data is not converted and returned compressed.

7.6.43. XS_BATTERY

The XS_BATTERY enumerates the battery condition.

- **XS_BAT_LEVEL_MASK**: masks the battery level bits.
- **XS_BAT_STATE_MASK**: masks the battery status bits.
- **XS_BAT_STATE_DISCHARGING**: the power supply is not connected.
- **XS_BAT_STATE_CHARGING**: the power supply is connected and charging the battery.
- **XS_BAT_STATE_FULLY_CHARGED**: the battery is fully charged.
- **XS_BAT_DISABLED**: the battery is connected but it's disabled.
- **XS_BAT_UNKNOWN**: unknown state or not supported.

7.6.44. XS_LENS_INFO

The XS_LENS_INFO enumerates lens information.

- **XS_LENS_TYPE**: 4 bits returning lens type (0: none, 1:Canon 2: MFT, 4:IDT MFT).
- **XS_LENS_ZOOM**: lens has configurable zoom. If 0, the lens is prime.
- **XS_LENS_MOTOR_ZOOM**: lens has motorized zoom.

7.6.45. XS_LENS_CMD

The XS_LENS_CMD enumerates some lens commands.

- **XS_LCMD_POWEROFF**: cut lens power for safe removal.

- **XS_LCMD_RESET**: restores the lens after a power off (no need to unplug and plug the lens).

7.6.46. XS_ERROR

The XS_ERROR enumerates the return codes. See Appendix A.

7.6.47. XS_INFO

The XS_INFO enumerates the camera information index. See Appendix B.

7.6.48. XS_PARAM

The XS_PARAM enumerates the camera parameters. See Appendix C.

7.7. Appendix G – Structures

This appendix describes the structures defined in the **XStrmAPI.h** header file.

7.7.1. XS_SETTINGS

The XS_SETTINGS structure is an opaque structure that contains the all the camera parameters in compact format. The user may access the structure using the XsSetParameter and XsGetParameter routines.

```
typedef struct
{
      XSULONG32 cbSize;
      XSULONG32 nData[ 255 ];
} XS_SETTINGS, *PXS_SETTINGS;
```

Members

cbSize

It specifies the size of the structure. Must be set to sizeof (XS_SETTINGS), otherwise the related functions doesn't work.

nData

It specifies the opaque structure data, an array of 255 XSULONG32 values.

7.7.2. XS_ENUMITEM

The XS_ENUMITEM structure contains information about a camera. It must be used in the camera enumeration procedure with the XsEnumCameras routine.

```
typedef struct
{
      XSULONG32 nCameraId;
      XSULONG32 nCameraModel;
      XSULONG32 nSensorType;
      XSULONG32 nSensorModel;
      XSULONG32 nSerial;
      XSULONG32 nRevision;
      XSULONG32 nEFWVersion;
      XSULONG32 nCFWVersion;
      XSULONG32 bIsOpen;
      XSULONG32 nLinkType;
      XSULONG32 bIRIG;
      XSULONG32 bIntensified;
      XSULONG32 nIntClock;
      XSULONG32 nLicInfo;
      XSULONG32 nSubModel;
      XSULONG32 bEDR;
      XSULONG32 nMinExp;
      XSULONG32 nGeAdpMTU;
      XSULONG32 nMiscCaps;
      XSULONG32 bDgrSize;
      XSULONG32 bNewDesign;
      XSULONG32 bLight;
      XSULONG32 bPlus;
      XSULONG32 bGigaEth;
      XSULONG32 nUsbVID;
      XSULONG32 nUsbPID;
      XSULONG32 nUsbPort;
      XSULONG32 nGeCamMACAddLo;
      XSULONG32 nGeCamMACAddHi;
      XSULONG32 nGeCamIPAdd;
      XSULONG32 nGeCamNetMask;
      XSULONG32 nGeCamCmdPort;
      XSULONG32 nGeAdpMACAddLo;
      XSULONG32 nGeAdpMACAddHi;
      XSULONG32 nGeAdpIPAdd;
      XSULONG32 nGeAdpNetMask;
      char szCameraName[L_CAMNAME];
      XSULONG32 nCFW2Version;
      XSULONG32 nFGType;
      XSULONG32 nCFW3Version;
      XSULONG32 nCFAPattern;
      XSULONG32 nSSDSizeLo;
      XSULONG32 nSSDSizeHi;
      XSULONG32 nBatteryInfo;
      XSULONG32 nDDRSizeLo;
      XSULONG32 nDDRSizeHi;
      XSULONG32 anFill[10];
} XS_ENUMITEM, *PXS_ENUMITEM;
```

Members

nCameraId

It specifies the ID which identifies a camera among others. The user must use this camera id to open the camera with XsOpenCamera.

nCameraModel

It specifies the camera model (X3, X4, X5, etc.).

nSensorType

It specifies the sensor type (monochrome or color).

nSensorModel

It specifies the sensor model.

nSerial

It specifies the camera serial number (10 decimal digits value).

nRevision

It specifies the camera hardware revision number (A, B, C, etc.).

nEFWVersion

It specifies the EEPROM firmware version.

nCFWVersion

It specifies the Controller firmware version.

bIsOpen

It specifies whether the camera is currently open or not.

nLinkType

It specifies the camera link (USB 2.0 or Giga Ethernet).

bIRIG

It specifies if the camera supports IRIG

bIntensified

It specifies if the camera is intensified.

nIntClock

It specifies the internal clock frequency in Hertz.

nLicInfo

It specifies the license information (LSW: max count, MSW: current count).

nSubModel

It specifies the camera sub-model. If the value is 0 the sub-model is not specified.

bEDR

It specifies if the camera supports the EDR mode (Y4 and N4 cameras only).

nMinExp

It specifies the camera minimum exposure (1 µs or 100 ns).

nGeAdpMTU

It specifies the network adapter MTU value (Maximum Transmission Unit). It is useful to determine whether the jumbo packets may be enabled or not on the camera.

nMiscCaps

It specifies the camera miscellaneous capabilities (bit map).

nMinExp

It specifies the camera minimum exposure (1 µs or 100 ns).

bDgrSize

It specifies whether the camera supports configurable datagram size, i.e. jumbo packets (Ethernet).

bNewDesign

It specifies whether the camera is redesigned or not (image processing is on-board)

bLight

It specifies whether the camera is Light or not (Light cameras have reduced capabilities, like speed and resolution)

bPlus

It specifies if the camera has the Plus™ capability, i.e. it can acquire at double speed.

bGigaEth

It specifies if the camera has the Giga-Ethernet connector.

nUsbVID

It specifies the USB Vendor ID (USB 2.0 cameras only).

nUsbPID

It specifies the USB Product ID (USB 2.0 cameras only).

nUsbPort

It specifies the USB Port Number (USB 2.0 cameras only).

nGeCamMACAddLo

It specifies the low part of the camera MAC address (Gigabit Ethernet cameras only).

nGeCamMACAddHi

It specifies the high part of the camera MAC address (Gigabit Ethernet cameras only).

nGeCamIPAdd

It specifies the camera IP Address (Gigabit Ethernet cameras only).

nGeCamNetMask

It specifies the camera subnet mask (Gigabit Ethernet cameras only).

nGeCamDfltGw

It specifies the camera default gateway (Gigabit Ethernet cameras only). Not used yet.

nGeCamCmdPort

It specifies the camera Command UDP Port (GE cameras only). Do not change if not necessary.

nGeAdpMACAddLo

It specifies the low part of the Ethernet adapter MAC address (GE cameras only).

nGeAdpMACAddHi

It specifies the high part of the Ethernet adapter MAC address (GE cameras only).

nGeAdpIPAdd

It specifies the Ethernet adapter IP Address (GE cameras only).

nGeAdpNetMask

It specifies the Ethernet adapter subnet mask (GE cameras only).

szCameraName

It specifies if the camera name.

nFGType

It specifies the Camera Link Frame Grabber model (M-Series cameras only).

nCFW3Version

It specifies the controller firmware version 3 (build number).

nCFAPattern

It specifies the CFA Bayer pattern (see XS_CFA_PATTERN).

nSSDSizeLo, nSSDSizeHi

It specifies the size of the on-board SSD.

nBatteryInfo

It specifies the battery info. The LS word includes the manufacturing date (day: 5 bits, month: 4 bits, year after 1980: 7 bits), the MS word includes the battery serial number.

nDDRSizeLo, nDDRSizeHi

It specifies the size of onboard DDR.

anFill[10]

It specifies an array of unused parameters.

7.7.3. XS_FRAME

The XS_FRAME structure contains information about the image frame to be grabbed. It is used to acquire images in the XsSynchGrab and XsQueueOneFrame routines.

```
typedef struct
{
      void      *pBuffer;
      XSULONG32 nBufSize;
      XSULONG32 nImages;
      XSULONG32 nFormat;
      XSULONG32 nWidth;
      XSULONG32 nHeight;
      XSULONG32 nPixDepth;
      XSULONG32 nErrorCode;
} XS_FRAME, *PXS_FRAME;
```

Members

pBuffer

It specifies the pointer to the data. This field must be filled before calling any related routine.

nBufSize

It specifies the data buffer size, in bytes. This field must be filled before calling any related routine.

nImages

It specifies the number of images to acquire (1 in single exposure mode, 2 in double exposure mode); the buffer size must be enough to contain the specified number of images. This field must be filled before calling any related routine.

nFormat

It specifies the image format; this field is filled when the related routine returns.

nWidth

It specifies the image width; this field is filled when the related routine returns.

nHeight

It specifies the image height; this field is filled when the related routine returns.

nPixDepth

It specifies the image pixel depth; this field is filled when the related routine returns.

nErrorCode

It specifies the result code of the Grab operation.

7.7.4. XS_BROC_SECTION

The XS_BROC_SECTION contains information about a specific BROC section (address, firs frame index and time from trigger).

```
typedef struct _XS_BROC_SECTION
{
      XSULONG32 nStartAddrLo;
      XSULONG32 nStartAddrHi;
      XSULONG32 n1stFrmIdx;
      XSULONG32 nTrgTime;
} XS_XS_BROC_SECTION, *PXS_BROC_SECTION;
```

Members

nStartAddrLo

It specifies the low part of the segment starting address.

nStartAddrHi

It specifies the hight part of the segment starting address.

n1stFrmIdx

It specifies the index of the first frame of the BROC section.

nTrgTime

It specifies the time from trigger.

7.7.5. XS_BROC

The XS_BROC structure contains an array of 256 BROC sections (see above).

```
typedef struct _XS_BROC
{
     XS_XS_BROC_SECTION sect[256];
} XS_XS_BROC, *PXS_BROC;
```

Members

sect

It specifies the array of 256 BROC sections

7.7.6. XS_GPSTIMING

The XS_GPSTIMING contains timing information about the IRIG/GPS data.

```
typedef struct _XS_GPSTIMING
{
      XSULONG32 nSignalPresent;
      XSULONG32 nYear;
      XSULONG32 nDayOfYear;
      XSULONG32 nHours;
      XSULONG32 nMinutes;
      XSULONG32 nSeconds;
      XSULONG32 nMicroSeconds;
      XSULONG32 nFlags;
} XS_GPSTIMING, *PXS_GPSTIMING;
```

Members

nSignalPresent

It specifies if the IRIG/GPS signal is currently locked.

nYear

It specifies the year.

nDayOfYear

It specifies the day of the year (1 to 365).

nHours

It specifies the hours.

nMinnutes

It specifies the minutes.

nSeconds

It specifies the seconds.

nMicroseconds

It specifies the microseconds.

nFlags

It specifies some flags depending on the format (for ex. CF on X cameras).

7.7.7. XS_W2DCFG

The XS_W2DCFG contains configuration parameters for streaming to local disk.

```
typedef struct _XS_W2DCFG
{
      XSULONG32 nDDRBufSize;
      XSULONG32 nOpt;
      XSULONG32 nDrives;
      char szVolume1[64];
      char szVolume2[64];
      char szVolume3[64];
      char szVolume4[64];
      char szDirectory[512];
} XS_GPSTIMING, *PXS_GPSTIMING;
```

Members

nDDRBufSize

It specifies the size of the DDR buffer (M cameras only)

nOpt

It specifies the write to disk options (not used).

nDrives

It specifies the number of drives (1, 2 or 4).

szVolume1, 2, 3, 4

It specifies the disk volumes

szDirectory

It specifies the directory name to be added to volumes.

7.7.8. XS_AsyncCallback

The XS_AsyncCallback is the prototype of the callback function passed to the XsQueueOneFrame, XsQueueCameraSettings and XsMemoryStartGrab routines. The callback is called by the driver when the operation is completed.

```
typedef void (XSTREAMAPI *XS_AsyncCallback)
(
     void     *pUserData,
     XS_ERROR nErrCode,
     XSULONG32 nFlags
);
```

Members

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

nErrCode

It specifies the operation return code.

nFlags

It specifies a combination of the XS_CALLBACK_FLAGS values.

7.7.9. XS_ProgressCallback

The XS_ProgressCallback is the prototype of the callback function passed to the XsCalibrateNoiseReduction routine. The callback is called by the driver during the calibration and allows the user to show the calibration progress.

```
typedef void (XSTREAMAPI *XS_ProgressCallback)
(
     void     *pUserData,
     XSULONG32 nProgress,
     XSULONG32 nCount
);
```

Members

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

nProgress

It specifies the current progress index.

nCount

It specifies the total progress count.

7.7.10. XS_AnnouncementCallback

The XS_AnnouncementCallback is the prototype of the callback function passed to the XsSetAnnouncementCallback routine.

```
typedef void (XSTREAMAPI *XS_AnnouncementCallback)
(
     void *pUserData,
     char *pszAnnouncement
);
```

Members

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

pszAnnouncement

It specifies the announcement message zero-terminated string.

7.7.11. XS_StreamingCallback

XS_StremingCallback is the format of the callback function passed to XsConfigureWriteToDisk routine. The callback is called when write to disk is enabled and raw data is saved to the local disk. It also returns the amount of available DDR. The process should be stopped when this amount is too low and override may occur.

```
typedef void (XSTREAMAPI *XS_StreamingCallback)
(
      void     *pUserData,
      XSULONG32 nParam1,
      XSULONG32 nParam2,
      XSULONG32 nErrCode
);
```

Members

pUserData

Specifies a parameter passed to the callback routine, it may be a pointer to user data.

nParam1

It specifies the total number of frame saved.

nParam2

It specifies the percentage of available DDR buffer (0 to 1000). If this value goes below 200 override may occur.

nErrCode

It specifies the error code when something wrong happens. It returns XS_E_W2D_OVERRUN if DDR buffer overrun happens, otherwise 0.