



***Veritas LED***  
**SDK Reference Manual**  
(Software Development Kit, 32 and 64 bit)

### **Software Release**

1.00.07

### **Document Revision**

December 2018

### **Products Information**

<http://www.idtvision.com>

#### **North America**

1202 E Park Ave  
TALLAHASSE FL 32301  
United States of America  
P: (+1) (850) 222-5939  
F: (+1) (850) 222-4591  
[llourenco@idtvision.com](mailto:llourenco@idtvision.com)

#### **Europe**

via Pennella, 94  
I-38057 - Pergine Valsugana (TN)  
Italy  
P: (+39) 0461- 520511  
[pgallorosso@idtvision.com](mailto:pgallorosso@idtvision.com)

Eekhoornstraat, 22  
B-3920 - Lommel  
Belgium  
P: (+32) 11- 551065  
F: (+32) 11- 554766  
[amarinelli@idtvision.com](mailto:amarinelli@idtvision.com)

### **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

|   |           |
|---|-----------|
| <b>1. OVERVIEW</b> .....                        | <b>4</b>  |
| 1.1. Directories structure.....                 | 5         |
| 1.2. Supported devices.....                     | 6         |
| <b>2. USING THE SDK</b> .....                   | <b>7</b>  |
| 2.1. Overview.....                              | 7         |
| 2.1.1. Programming Languages.....               | 7         |
| 2.1.2. Example.....                             | 8         |
| 2.2. Detect a device and open it.....           | 9         |
| 2.2.1. Initialize the network environment.....  | 9         |
| 2.2.2. Enumerate/Open a device.....             | 9         |
| 2.3. Device timing configuration.....           | 10        |
| 2.3.1. Read the parameters limits.....          | 10        |
| 2.3.2. Read/Write the device configuration..... | 11        |
| 2.3.3. Operating mode.....                      | 12        |
| 2.3.4. Period, pulse width and delays.....      | 13        |
| 2.3.5. Dimming levels.....                      | 14        |
| 2.3.6. Dimming presets.....                     | 15        |
| 2.3.7. Turn off the lights.....                 | 15        |
| <b>3. SDK REFERENCE</b> .....                   | <b>16</b> |
| 3.1. Initialization functions.....              | 16        |
| 3.1.1. Overview.....                            | 16        |
| 3.1.2. LedInit.....                             | 17        |
| 3.1.3. LedEnumerate.....                        | 18        |
| 3.1.4. LedOpen.....                             | 19        |
| 3.1.5. LedClose.....                            | 20        |
| 3.2. Configuration Functions.....               | 21        |
| 3.2.1. Overview.....                            | 21        |
| 3.2.2. LedGetTimingCaps.....                    | 22        |
| 3.2.3. LedGetConfiguration.....                 | 23        |
| 3.2.4. LedSetConfiguration.....                 | 24        |
| 3.2.5. LedLoadPreset.....                       | 25        |
| 3.2.6. LedSavePreset.....                       | 26        |
| <b>4. APPENDIX</b> .....                        | <b>27</b> |
| 4.1. Appendix A - Return Codes.....             | 27        |
| 4.2. Appendix B – Data types.....               | 28        |
| 4.2.1. LED_MODEL.....                           | 28        |
| 4.2.2. LED_LINK.....                            | 28        |
| 4.2.3. LED_MODE.....                            | 28        |
| 4.2.4. LED_RETURN.....                          | 28        |
| 4.3. Appendix C – Structures.....               | 29        |
| 4.3.1. LED_ENUMITEM.....                        | 29        |
| 4.3.2. LED_CFG.....                             | 31        |
| 4.3.3. LED_CAPS.....                            | 32        |

# 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.

## **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/VeritasSDK V1.V2.V3”**

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

**BIN32:** it contains the 32-bit files (DLLs) that can be re-distributed with the device and your 32-bit application.

**BIN64:** it contains the 64-bit files (DLLs) that can be re-distributed with the device and your 64-bit application.

**DOCS:** it contains the SDK documentation.

**INCLUDE:** it contains the SDK header file (H).

**Lib32:** it contains the SDK lib files (32-bit).

**Lib64:** it contains the SDK lib files (64-bit).

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

## **1.2. Supported devices**

The Software development Kit supports the following Veritas models:

- 1<sup>st</sup> and 2<sup>nd</sup> generation 4-Port controllers.
- Constellation Access Point (CAP).
- Constellation 500, 500F and 500E.
- Constellation 2400H and 2400B.

## 2. Using the SDK

### 2.1. Overview

#### 2.1.1. Programming Languages

A C/C++ header file is included in the SDK (**LED\_API.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.

The Windows driver is a DLL (LEDControl.dll) that resides in Bin32 directory. The 64 bit version may be found in the Bin64 directory.

**MS Visual C++™**: A Visual C++ 6.0 stub COFF library is provided (**LEDControl.lib** in the Lib32 and Lib64 sub-directories); if you are programming with Visual C++, link your application to LEDControl.lib. The DLL uses Windows standard calling conventions (\_stdcall).

**Borland C++ Builder™**: the LEDControl.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 LEDControl.lib LEDControl.dll".

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

## 2.1.2. Example

The routines and the parameters used in this example are described in the following topics.

```
#include "LED_API.h"
#include <stdio.h>

int main(int argc, char* argv[])
{
    LED_ENUMITEM led[8];
    int nListLen = sizeof(led)/sizeof(LED_ENUMITEM);
    int nFrq = 50, nT;
    LED_HANDLE hDev;
    LED_CAPS caps;
    LED_CFG cfg;

    // Init the IP address (broadcast)
    LedInit(0xFFFFFFFF);

    // nListLen is the length of LED_ENUMITEM array
    nListLen = LedEnumerate( &led[0], nListLen );
    // nListLen is now the number of available devices.
    if (nListLen==0) return 0;

    // Open the first device in the list.
    LedOpen( &led[0], &hDev );

    // read the capabilities
    LedGetTimingCaps( hDev, &caps );

    // read the configuration
    LedGetConfiguration( hDev, &cfg );

    nT = (int)(1000000000./((double)nFrq+0.5));

    // set the parameters
    cfg.nMode = LED_MD_INT;
    cfg.nPeriod = nT;
    cfg.nPulseWid0 = 20000; // 20 us
    cfg.nPulseWid1 = 40000; // 40 us
    cfg.nPulseWid2 = cfg.nPulseWid3 = 0;
    cfg.nDelay1 = nT/4; // delay of 1/4 of the period
    cfg.nDelay0 = cfg.nDelay2 = cfg.nDelay3 = 0;

    // Send settings to the device
    LedSetConfiguration( hDev, &cfg );

    // wait
    getchar();

    // Send reset to the device
    cfg.nPulseWid0 = cfg.nPulseWid1 = 0; // reset
    LedSetConfiguration( hDev, &cfg );

    // Close the device
    LedClose(hDev);

    return 0;
}
```



## 2.2. Detect a device and open it

### 2.2.1. Initialize the network environment

The first call into the driver may be **LedInit**. Call **LedInit** to set the IP address of the network card connected to your device. If you don't know the IP address or your devices are connected to different network adapters, set the value to 0xFFFFFFFF (or do not do anything). If the IP address is undefined, the driver will try to enumerate the devices from all the network adapters installed on your computer.

### 2.2.2. Enumerate/Open a device

To get the list of available devices, call **LedEnumerate**. Use the returned **LED\_ENUMITEM** structure to open the device with **LedOpen**. Below a simple example of opening the first available Veritas device.

```
LED_ENUMITEM xsl[10];
int nLen = sizeof(xsl)/sizeof(LED_ENUMITEM);

// nListLen is the length of your LED_ENUMITEM array
nLen = LedEnumerate( &xsl[0], nLen );

// nListLen is now the number of cameras available.
// It may be larger than your XS_ENUMITEM array length!
if ( nLen>0 )
{
    LED_HANDLE hDev;
    // Open the first device in the list.
    LedOpen( &xsl[0], &hDev );
    // Do something...
    if ( hDev )
    {
        ...
        // Close the device.
        LedClose( hDev );
    }
}
```

The list is filled with the devices information. The info will be used to open the device and retrieve the handle. Then the handle must be used to call any other routine of the SDK.

## 2.3. Device timing configuration

The device configuration is stored in the opaque **LED\_CFG** structure. The structure is used to read and write parameters from/to the device.

### 2.3.1. Read the parameters limits

Before any other operation, the user should read the LED\_CAPS structure from the device.

The LED\_CAPS structure returns the limits of the timing parameters that can be configured in the device. See below:

- Range of operating modes (minimum and maximum value)
- Range of period (minimum and maximum in nanoseconds).
- Range of dimming values (minimum and maximum).
- Max number of dimming presets.

The example below shows how to read the LED\_CAPS structure

```
LED_HANDLE hDev;
LED_CAPS caps;
// Open the device
LedOpen( &xsl[0], &hDev );
// Do something...
if ( hDev)
{
    LedGetTimingCaps(hDev, &caps);
    // use the caps to set the limits
}
// Close the device.
LedClose( hDev );
```

### 2.3.2. Read/Write the device configuration

Once the parameter limits are known the user may start operating the device. To do so, read the LED\_CFG structure from the device, modify the parameters with valid data and send the configuration back to the device.

```
LED_CFG cfg;

// Read the cfg from the device.
LedGetConfiguration( hDev, &cfg );

// Change configuration
cfg.nMode = LED_MD_SYNCIN;

// Send the configuration to the device
LedSetConfiguration( hDev, &cfg );
```

### 2.3.3. Operating mode

Each Veritas device has multiple operation modes. The minimum and maximum allowed values for this parameter can be retrieved from the LED\_CAPS structure.

The table below shows the available modes.

| Mode           | Description  |
|----------------|--|
| LED_MD_INT     | The timing signals are generated by the internal clock. The frequency, pulse width and delays can be configured in the LED_CFG structure         |
| LED_MD_GPS     | The reference signal is externally generated by GPS. Frequency, width and delays can be configured.  |
| LED_MD_1PPS    | The reference signal is externally generated by 1PPS. Frequency, width and delays can be configured.   |
| LED_MD_SYNC_IN | The timing signals follow an external signal connected to the sync in connector. Frequency, width and delay of the signals cannot be configured. |
| LED_MD_CONT_D  | The output is continuous and not pulsed, without dimming.  |
| LED_MD_CONT_ND | The output is continuous and not pulsed, with dimming. Dimming levels can be configured.   |
| LED_MD_IRIG    | The reference signal is externally generated by IRIG. Frequency, width and delays can be configured.   |
| LED_MD_PTP     | The reference signal is externally generated by PTP (IEEE1548). Frequency, width and delays can be configured.                                   |

The formulas for converting the frequency into the period is shown below

$$nFrq = (\text{int})(1000000000./(\text{double})nPeriodNS + 0.5);$$

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

$$nPeriodNS = (\text{int})(1000000000./(\text{double})nFrq + 0.5);$$

### 2.3.4. Period, pulse width and delays

If the device is configured in internal, GPS, 1PPS, IRIG or PTP mode, each device channel (1 channel for the single LED models and 4 channels for the controllers) can be independently configured (pulse width and delay).

All those parameters are in nanoseconds.

**A channel is off if the corresponding pulse width is set to 0.**

The example below shows how to set a frequency of 100 Hz on channel 0 and turn other channels off.

```
LED_CFG cfg;
LED_CAPS caps;
int nFrq = 100
int nT = (int)(1000000000./ (double)nFrq+0.5);

// Read the capabilities and the configuration
LedGetTimingCaps(hDev, &caps);
LedGetConfiguration(hDev, &cfg);

// set the parameters
cfg.nMode = LED_MD_INT;
cfg.nPeriod = nT;
cfg.nPulseWid0 = nT/2; // 50% of duty cycle;
cfg.nPulseWid1 = nT/2;
cfg.nPulseWid2 = cfg.nPulseWid3 = 0;
cfg.nDelay1 = nT/4; // delay of 1/4 of the period
cfg.nDelay0 = cfg.nDelay2 = cfg.nDelay3 = 0;

// send the configuration
LedSetConfiguration(hDev, &cfg);
```

### 2.3.5. Dimming levels

If the device is configured in dimming mode, each channel dimming level can be independently configured. The values of minimum and maximum dimming are retrieved from the LED\_CAPS structure. The minimum value corresponds to turn the channel off, while the maximum corresponds to maximum intensity.

**A channel is off if the corresponding dimming level is set to the minimum.**

The example below shows how to set channel 0 and 2 to half of the maximum intensity and turn off channels 1 and 3. Once the configuration is sent to the device, the lights turn on immediately.

```
LED_CFG cfg;
LED_CAPS caps;
int nRange;

// Read the capabilities and the configuration
LedGetTimingCaps(hDev, &caps);
LedGetConfiguration(hDev, &cfg);

// set the parameters
nRange = caps.nMaxDimLevel - caps.nMinDimLevel;
cfg.nMode = LED_MD_CONT_D;
cfg.nDimLevel0 = cfg.nDimLevel2 = caps.nMinDimLevel;
cfg.nDimLevel1 = cfg.nDimLevel3 = caps.nMinDimLevel + nRange/2;

// send the configuration
LedSetConfiguration(hDev, &cfg);
```

### 2.3.6. Dimming presets

Veritas devices have the capability to store up to 4 dimming presets to the flash memory. The user can load a preset to the device or store a set of dimming values to one of the presets. The preset cannot store other timing values, such as frequency, pulse width or delay.

The example below shows how to store a set of dimming values to preset 0. It sets the levels to ascending values from minimum to maximum, then it sends those values to the device and saves them to preset 0.

```
LED_CFG cfg;
LED_CAPS caps;
int nRange;

// Read the capabilities and the configuration
LedGetTimingCaps(hDev, &caps);
LedGetConfiguration(hDev, &cfg);

// set the parameters
nRange = caps.nMaxDimLevel - caps.nMinDimLevel;
cfg.nMode = LED_MD_CONT_D;
cfg.nDimLevel0 = caps.nMinDimLevel + nRange/4;
cfg.nDimLevel1 = caps.nMinDimLevel + nRange/2;
cfg.nDimLevel2 = caps.nMinDimLevel + (3*nRange)/4;
cfg.nDimLevel3 = caps.nMaxDimLevel;

// send the configuration
LedSetConfiguration(hDev, &cfg);

// save current set to preset 0
LedSavePreset(hDev, 0);
```

### 2.3.7. Turn off the lights

When the user calls the LedClose function, the handle becomes no longer valid but the light do not turn off. The device keeps the latest configuration and does not stop the emission. To stop the light emission, the user must set to 0 the pulse widths (if the device is in a non dimming mode) or the dimming levels (if the device is in dimming mode).

## 3. SDK Reference

### 3.1. Initialization functions

#### 3.1.1. Overview

The initialization functions allow the user to detect, open and close the devices.

**LedInit** initializes the IP address of the network adapter connected to the devices.

**LedEnumerate** detects and enumerates the Veritas devices.

**LedOpen** opens a Veritas device.

**LedClose** closes a Veritas device previously open.



### 3.1.2. LedInit

**int LedInit (unsigned int *nNetAdpIP*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_GENERIC if any error occurs during the initialization.

#### **Parameters**

*nNetAdpIP*

The parameter specifies the IP address of the network adapter connected to the devices.

#### **Remarks**

The routine sets the network adapter IP address. It should be called before any other routine. The *nNetAdpIP* parameter specifies the IP address of the network adapter connected to the Veritas devices. If the value is not specified or it's configured with the value 0xFFFFFFFF, the driver search fro the devices on

See also:

### 3.1.3. LedEnumerate

**int LedEnumerate (PLED\_ENUMITEM *pItemList*, int *nListLen*)**

#### **Return values**

the number of detected Veritas devices

#### **Parameters**

*pItemList*

Specifies the pointer to an array of LED\_ENUMITEM structures

*nListLen*

Specifies the number of LED\_ENUMITEM items contained in the list

#### **Remarks**

The routine enumerates the active devices and fills the **LED\_ENUMITEM** structures with information about them. The *nListLen* variable specifies the number of structures in the *pItemList* array. The routine returns the number of detected Veritas devices or zero.

See also: **LedOpen**

### 3.1.4. LedOpen

**int LedOpen (XSULED\_ENUMITEM \*pItem, LED\_HANDLE\* pHandle)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

L\_E\_OPEN, if the device cannot be open.

#### **Parameters**

*pItem*

Specifies the pointer to a valid LED\_ENUMITEM structure

*pHandle*

Specifies the pointer to the variable that receives the device handle

#### **Remarks**

The routine opens the device with the *pItem* structure. The structure is retrieved by calling the **LedEnumerate** routine (see the LED\_ENUMITEM structure). If any error occurs during the procedure, the routine returns an error code.

See also: **LedClose**

### 3.1.5. LedClose

**int LedClose (LED\_HANDLE *hDev*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HANDLE, if the handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

#### **Remarks**

Closes an open device

See also: **LedOpen**

## 3.2. Configuration Functions

### 3.2.1. Overview

The configuration functions allow the user to control the parameters of the device.

**LedGetTimigCaps** gets information about the limits of the timing parameters (minimum and maximum values).

**LedGetConfiguration** reads current timing settings from the device and fills the LED\_CFG structure.

**LedSetConfiguration** sends an updated LED\_CFG structure to the device.

**LedLoadPreset** loads a dimming preset from the device flash memory.

**LedSavePreset** saves the current dimming status to a preset in the device flash memory.

### 3.2.2. LedGetTimingCaps

**Int LedGetTimingCaps (LED\_HANDLE *hDev*, PLED\_CAPS *pCaps*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HANDLE, if the device handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

*pCaps*

Specifies the pointer to a valid LED\_CAPS structure

#### **Remarks**

This function fills a LED\_CAPS structure with information about the limits of timing parameters.

**See also:**

### 3.2.3. LedGetConfiguration

**int LedGetConfiguration (LED\_HANDLE *hDev*, PLED\_CFG *pCfg*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HND, if the device handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

*pCfg*

Specifies the pointer to the structure to be filled with the device configuration

#### **Remarks**

This function reads the current configuration from the device and fills the structure with the parameter values.

**See also:** [LedSetConfiguration](#)

### 3.2.4. LedSetConfiguration

**int LedSetConfiguration (LED\_HANDLE *hDev*, PLED\_CFG *pCfg*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HND, if the device handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

*pCfg*

Specifies the pointer to the structure to be sent to the device

#### **Remarks**

This routine sends the configuration to the device.

**See also:** **LedGetConfiguration**



### 3.2.5. LedLoadPreset

**int LedLoadPreset (LED\_HANDLE *hDev*, int *nPresetIdx*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HND, if the device handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

*nPresetIdx*

Specifies the index of the preset from where the values are loaded.

#### **Remarks**

The routine loads the dimming values from the preset specified by the *nPresetIdx* index. The index must be a value between 0 and *nMaxPreset* - 1, when *nMaxPreset* is the maximum preset index value read from the LED\_CAPS structure.

**See also:** LedSavePreset

### 3.2.6. LedSavePreset

**int LedSavePreset (LED\_HANDLE *hDev*, int *nPresetIdx*)**

#### **Return values**

LD\_SUCCESS if successful, otherwise

LD\_E\_INVALID\_HND, if the device handle is not valid.

#### **Parameters**

*hDev*

Specifies the handle to an open device

*nPresetIdx*

Specifies the index of the preset where the values are saved

#### **Remarks**

The routine saves the current dimming values to a preset specified by the *nPresetIdx* index. The index must be a value between 0 and *nMaxPreset* – 1, when *nMaxPreset* is the maximum preset index value read from the LED\_CAPS structure.

**See also:** LedLoadPreset

## 4. Appendix

### 4.1. 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 **LED\_API.h** header file in the **Include** sub directory.

| Code             | Value | Notes   |
|------------------|-------|---|
| LD_SUCCESS       | 0     | OK – No errors                                |
| LD_E_GENERIC     | -1    | Generic Error                                 |
| XS_E_UNSUPPORTED | -2    | The function is not supported for this device |
| XS_E_INVALID_HND | -3    | Invalid LED_HANDLE device handle              |
| XS_E_OPEN        | -4    | The device cannot be open                     |

## 4.2. Appendix B – Data types

This appendix describes the data types defined in the **XStrmAPI.h** header file.

### 4.2.1. LED\_MODEL

The LED\_MODEL type enumerates the device.

- **LM\_4LED**: 1<sup>st</sup> and 2<sup>nd</sup> generation legacy LED controllers.
- **LM\_CAP**: Constellation Access Point.
- **LM\_4XLED**: 4X 120W LED Controller.
- **LM\_C500**: Constellation 500/500F.
- **LM\_C500E**: Constellation 500E.
- **LM\_C2400H**: Constellation 2400H.
- **LM\_C2400B**: Constellation 2400B.

### 4.2.2. LED\_LINK

The LED\_LINK type enumerates the links.

- **LD\_LNK\_USB**: USB 2.0 link.
- **LD\_LNK\_ETH**: Ethernet (10/100 BaseT).
- **LD\_LNK\_WIFI**: Wi-Fi.

### 4.2.3. LED\_MODE

The LED\_MODE type enumerates the operating modes.

- **LED\_MD\_INT**: internal synchronization source.
- **LED\_MD\_GPS**: GPS sync source.
- **LED\_MD\_1PPS**: 1PPS sync source.
- **LED\_MD\_SYNCIN**: external sync in.
- **LED\_MD\_CONT\_D**: continuous with dimming.
- **LED\_MD\_CONT\_ND**: continuous without dimming.
- **LED\_MD\_IRIG**: IRIG sync source.
- **LED\_MD\_PTP**: PTP sync source.

### 4.2.4. LED\_RETURN

The LED\_RETURN type enumerates the return codes (see appendix A).

## 4.3. Appendix C – Structures

This appendix describes the structures defined in the **LED\_API.h** header file.

### 4.3.1. LED\_ENUMITEM

The LED\_ENUMITEM structure contains information about a device. It's retrieved by calling the LedEnumerate routine.

```
typedef struct
{
    int cbSize;
    int nDeviceID;
    int nDeviceModel;
    int nSerialNoLo;
    int nSerialNoHi;
    int nFWVersion;
    int nFWCfgVer;
    char szName[128];
    int bIsOpen;
    int nLinkType;
    int nUsbPort;
    int nUsbVID;
    int nUsbPID;
    int nEthDevAdd;
    int nEthDevSNM;
    int nEthAdpAdd;
    int nEthAdpSNM;
} XS_ENUMITEM, *PXS_ENUMITEM;
```

#### Members

##### *nCbSize*

It specifies the size of the structure.

##### *nDeviceID*

It specifies the ID which identifies a device.

##### *nDeviceModel*

It specifies the device model.

##### *nSerialLo*

It specifies the LS part of the device serial number.

##### *nSerialHi*

It specifies the MS part of the device serial number.

##### *nFWVersion*

It specifies the firmware version.

##### *nFWCfgVersion*

It specifies the firmware configuration.

*szName*

It specifies the device name (128 characters zero-terminated).

*bIsOpen*

It specifies whether the device is currently open or not.

*nLinkType*

It specifies the communication link (USB 2.0, Ethernet or Wi-Fi).

*nUsbPort*

It specifies the USB port ID (USB only).

*nUsbVID*

It specifies the USB Vendor ID (USB only).

*nUsbPID*

*It specifies the USB Product ID (USB only).*

*nEthDevAdd*

It specifies the device IP Address (Ethernet or Wi-Fi).

*nEthDevSNM*

It specifies the device sub-net mask (Ethernet or Wi-Fi).

*nEthAdpIPAdd*

It specifies the Ethernet adapter IP Address (GE cameras only).

*nEthAdpNetMask*

It specifies the Ethernet adapter sub-net mask (GE cameras only).

### 4.3.2. LED\_CFG

The LED\_CFG structure contains the device parameters. The user may modify the structure fields.

```
typedef struct
{
    unsigned int nMode;
    unsigned int nPeriod;
    unsigned int nPulseWid0;
    unsigned int nDelay0;
    unsigned int nPulseWid1;
    unsigned int nDelay1;
    unsigned int nPulseWid2;
    unsigned int nDelay2;
    unsigned int nPulseWid3;
    unsigned int nDelay3;
    unsigned int nDimLevel0;
    unsigned int nDimLevel1;
    unsigned int nDimLevel2;
    unsigned int nDimLevel3;
} LED_CFG, *PLED_CFG;
```

#### Members

##### *nMode*

It specifies the timing mode (see LED\_MODE enumeration).

##### *nPeriod*

The period (inverse of frequency) of the output timing signals. It's defined in nanoseconds.

##### *nPulseWid0, 1, 2, 3*

It specifies the pulse width of the channels. It's defined in nanoseconds.

##### *nDelay0, 1, 2, 3*

It specifies the delay of the channels. It's defined in nanoseconds.

##### *nDimLevel0, 1, 2, 3*

It specifies the dimming level of the channels when the mode is set to "dimming". It's defined in dimming units. The limits (min, max) can be retrieved from the LED\_CAPS structure.

### 4.3.3. LED\_CAPS

The LED\_CAPS structure contains information about the limits of the timing parameters (minimum and maximum). The user should take care of these values before setting the timing parameters.

```
typedef struct
{
    unsigned int nMinMode;
    unsigned int nMaxMode;
    unsigned int nMinPeriod;
    unsigned int nMaxPeriod;
    unsigned int nMinDimLevel;
    unsigned int nMaxDimLevel;
    unsigned int nMaxPreset;
} XS_FRAME, *PXS_FRAME;
```

#### Members

##### *nMinMode*

It specifies the minimum value for the mode parameter.

##### *nMaxMode*

It specifies the maximum value for the mode parameter.

##### *nMinPeriod*

It specifies the minimum value for the period parameter. The minimum value for width and delay is 0.

##### *nMaxPeriod*

It specifies the maximum value for the period parameter. The maximum value for width and delay is the period itself.

##### *nMinDimLevel*

It specifies the minimum value for the dimming parameter.

##### *nMaxDimLevel*

It specifies the maximum value for the dimming parameter.

##### *nMaxPreset*

It specifies the number of preset that can be configured (usually 4).