Interface Manual
USBLC8M32.DLL
USBLC8M64.DLL
(V1.04)

Coptonix GmbH
Luxemburger Str. 31
D – 13353 Berlin
Phone: +49 – (0)30 – 61 74 12 48
Fax: +49 – (0)30 – 61 74 12 47
www.coptonix.com
support@coptonix.com

Contents

Dynamic Link Library DLL

1 Functions

1.1 USB Functions

1.1.1 ls_GetErrorString

function ls_geterrorstring(dwErr : DWORD) : PAnsiChar;

ls_GetErrorString converts the error code *dwErr* to a readable zero terminated string. If not specified, *dwErr* is the return value of most functions below.

1.1.2 ls_Initialize

function ls_initialize(dwPipeSize, dwPacketLength, dwThreadClass : DWORD;
iThreadPrio : Integer; pcMsgID : PAnsiChar): DWORD;

When starting the application, this function is called when the default values are not sufficient. The argument *dwPipeSize* defines the size of a ring buffer (pipe). If *dwPipeSize* is equal to 512, it means 512 bytes buffer and 67108864 is equal to 64 Mbytes. The default value is 4MB. *dwPacketLength* is the number of bytes to be read per read request from the hardware FIFO. The value of *dwPacketlength* must be equal to or multiple of (number of pixels x 2) when operating in 16Bit mode, and equal to or multiple of number of pixels when operating in 8Bit mode.
Use the function *ls_getpacketlength* (1.1.4) to read the number of bytes the device transfers per USB transaction, and set *dwPacketLength* to multiple of this value. The argument *dwThreadClass* and *iThreadPrio* gives the possibility to adapt the priority of the reading thread. dwThreadClass is the thread class and the default value is NORMAL_PRIORITY_CLASS. iThreadPrio is the priority of the thread. It's default value is THREAD_PRIORITY_NORMAL.

ls_Initialize defines a new window message that is guaranteed to be unique throughout the system. The argument ***pcMsgID*** (as PAnsiChar e.g. "My_USBLS_App" should be unique). If more than one application use the same ***pcMsgID***, they will share the same window message ID. If the function successes, it returns a message identifier in the range 0xC000 through 0xFFFF. If the function fails, the return value is zero. The returned value must be saved in a global valid variable in order to use it later in processing messages. For the registration of the new window message the window API function "RegisterWindowMessage" is used.

1.1.3 ls_SetPacketLength

function ls_setpacketlength(dwPacketLength : DWORD) : DWORD;

ls_SetPacketLength sets the value of *dwPacketLength*. *dwPacketLength* is described in section 1.1.2. Before calling this function, the device must be closed. If the function fails, the return value (*dwErr*) is non zero.

1.1.4 ls_GetPacketLength

function ls_getpacketlength(var dwPacketLength : DWORD) : DWORD;

ls_GetPacketLength reads the recommended value for *dwPacketLength* from the line sensor controller. *dwPacketLength* is described in section 1.1.2. If the function fails, the return value (*dwErr*) is non zero.

1.1.5 ls_EnumDevices

function ls_enumdevices : Integer;

ls_EnumDevices enumerates and creates a list of all connected devices and then returns the number of connected devices.

1.1.6 ls_OpenDeviceByIndex

function ls_opendevicebyindex(index : Integer) : DWORD;

ls_OpenDeviceByIndex connects a USB device and starts the reading thread. The argument *index* is 0-based. This means that the first device was connected has the index zero (0), the second one has the index 1, and so on. If the function fails, the return value (*dwErr*) is non zero.

1.1.7 ls_OpenDeviceBySerial

function ls_opendevicebyserial(pcserialnum : PAnsiChar) : DWORD;

ls_OpenDeviceBySerial connect a USB device and starts reading thread (see ls_OpenDeviceByIndex). The argument ***pcserialnum*** is the serial number (e.g. 1500000) of a device. If the function fails, the return value (*dwErr*) is non zero.

1.1.8 ls_CloseDevice

function ls_closedevice : DWORD;

"ls_CloseDevice" disconnects the current opened device. If the function fails, the return value (*dwErr*) is non zero.

1.1.9 ls_DeviceCount

function ls_devicecount : Byte;

ls_DeviceCount returns the number of USB devices, which are currently connected to the system.

1.1.10 ls_CurrentDeviceIndex

function ls_currentdeviceindex : Integer;

ls_CurrentDeviceIndex returns the index of the opened USB device. The return value is -1 if no USB device is opened.

1.1.11 ls_GetFWVersion

function ls_getfwversion(index : Integer) : WORD;

ls_GetFWVersion returns the version of the firmware with index "***index***".

1.1.12 ls_GetVendorName

function ls_getvendorname(index : Integer) : PAnsiChar;

ls_GetVendorName returns the vendor's name of the device with index "***index***".

1.1.13 ls_GetProductName

function ls_getproductname(index : Integer) : PAnsiChar;

ls_GetProductName returns the product's name of the device with index "***index***".

1.1.14 ls_GetSerialNumber

function ls_getserialnumber(index : Integer) : PAnsiChar;

ls_GetSerialNumber the serial number of the device with index "***index***".

1.1.15 ls_SetEPTimeOut

function ls_seteptimeout(dwtimeout : DWORD) : DWORD;

ls_SetEPTimeOut sets the timeout value of the USB IN End Point. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s. Before calling this function, the device must be closed. If the function fails, the return value (*dwErr*) is non zero.

1.1.16 ls_GetEPTimeOut

function ls_geteptimeout : DWORD;

ls_GetEPTimeOut reads the current timeout value of the USB IN End Point. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s. If the function fails, the return value (*dwErr*) is non zero.

1.2 Data Functions

1.2.1 ls_WaitForPipe

function ls_waitforpipe(dwTimeOut : DWORD) : DWORD;

ls_WaitForPipe checks whether the pipe contains data for reading. If no data are available, the calling thread enters the wait state until data is received or the time-out interval elapses. *dwTimeOut* is the time out interval. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s. If the function fails, the return value (*dwErr*) is non zero.

1.2.2 ls_GetPipe

function ls_getpipe(lpBuffer : Pointer; dwToRead: DWORD;
var dwRead: DWORD): DWORD;

ls_GetPipe reads data from the pipe (ring buffer). The argument *lpBuffer* points to the buffer, which has to include the data. *dwToRead* specifies the length of the data which must be read, and *dwRead* returns the actual number of bytes read. If *dwToRead* is specified with 0, then *dwRead* returns actual number of bytes available without reading data. If the function fails, the return value (*dwErr*) is non zero.

1.2.3 ls_GetFPS

function ls_getfps : DWORD;

ls_GetFPS reads the number of bytes transferred per second. To determine the speed (number of frames per second) the return value must be divided by the number of pixels.

1.3 Camera Functions

1.3.1 ls_GetSensorType

function ls_getsensortype(Var wSensorType, wPixelCount : Word) : DWORD;

ls_GetSensorType reads the type of the sensor and the sensor's number of pixels from the sensor circuit board. See also "ls_GetMCUSensorType".
If the function fails, the return value (*dwErr*) is non zero.

1.3.2 ls_GetMCUSensorType

function ls_getmcusensortype(Var wSensorType : WORD) : DWORD;

ls_GetMCUSensorType reads the type of the sensor supported by the main circuit board. See also "ls_GetSensorType". If the function fails, the return value (*dwErr*) is non zero.

1.3.3 ls_GetSensorName

function ls_getsensorname(wSensorType : WORD) : PAnsiChar;

ls_GetSensorName converts the sensor's type to a readable zero terminated string.

1.3.4 ls_SetMode

function ls_setmode(ucMode : Byte) : DWORD;

There are 4 operation modes available. The value for *ucMode* must be

ONE_SHOT	0x00	Acquisition is software triggered.
EXT_TRIGGER	0x01	Acquisition is done on external trigger.
FREE_RUNNING	0x02	Acquisition is done continuously.
EXT_EXP_CTRL	0x03	Acquisition is done on external trigger.

In EXT_EXP_CTRL the time between the negative edges determines the integration time.
If the function fails, the return value (*dwErr*) is non zero.

1.3.5 ls_SetState

function ls_setstate(ucState : Byte) : DWORD;

ls_SetState starts or stops data acquisition. If value passed to *ucState* is 0x01, acquisition starts. If value passed for *ucState* is 0x00, acquisition stops. If the function fails, the return value (*dwErr*) is non zero.

1.3.6 ls_SetIntTime

function ls_setinttime(dwIntTime : DWORD) : DWORD;

ls_SetIntTime sets the integration/exposure time *dwIntTime* in microseconds. If the function fails, the return value (*dwErr*) is non zero.

1.3.7 ls_SetExtDelay

function ls_setextdelay(dwExtDelay : DWORD) : DWORD;

ls_SetExtDelay sets a time delay between the external trigger and the start of integration. The delay time is equal to ***dwExtDelay* x 1/f**, where f is the clock frequency. The frequency depends on the sensor used. e.g. 1/f = 120ns for the Hamamatsu sensor S11639-01.
If the function fails, the return value (*dwErr*) is non zero.

1.3.8 ls_SetCFG1

function ls_setcfg1(ucCFG1 : Byte) : DWORD;

ls_SetCFG1 sets the configuration register 1.
Note: Changes take effect after power off/on.

Bit number	Value	Description
Bit 0	0	The image number is not used.
	1	A 4 byte image number is appended at the end of an image data. In 16Bit mode the image number replaces the last 2 pixel values. In 8Bit mode the image number replaces the last 4 pixel values.
Bit 1	0	8Bit mode.
	1	16Bit mode.
Bit[2:6]		Number of images to be buffered before transferring to the host. This value deteremines the value of *dwPacketLength* used in functions *ls_initialize* and *ls_setpacketlength*. The max. number of images depends on the number of pixels. e.g. for 2048 pixel the max. number of images is: 16Bit mode: 4 images 8Bit mode: 8 images.
	00000 = 0	1 image / USB transfer.
	00001 = 1	2 images / USB transfer
	….	
	11110 = 30	31 images / USB transfer
	11111 = 31	32 images / USB transfer
Bit 7		Not used. Do not care.

If the function fails, the return value (*dwErr*) is non zero.

1.3.9 ls_GetMode

function ls_getmode(Var ucMode : Byte) : DWORD;

ls_GetMode returns the current mode "*ucMode*":

ONE_SHOT	0x00	Acquisition is software triggered.
EXT_TRIGGER	0x01	Acquisition is done on external trigger.
FREE_RUNNING	0x02	Acquisition is done continuously.
EXT_EXP_CTRL	0x03	Acquisition is done on external trigger.

If the function fails, the return value (*dwErr*) is non zero.

1.3.10 ls_GetState

function ls_getstate(Var ucState : Byte) : DWORD;

ls_GetState returns the current state "*ucState*":

- 0x00 Acquisition is stopped
- 0x01 Acquisition is running

If the function fails, the return value (*dwErr*) is non zero.

1.3.11 ls_GetIntTime

function ls_getinttime(Var dwIntTime : DWORD) : DWORD;

ls_GetIntTime returns the integration/exposure time "*dwIntTime*" in microseconds. If the function fails, the return value (*dwErr*) is non zero.

1.3.12 ls_GetExtDelay

function ls_getextdelay(var dwExtDelay : DWORD) : DWORD;

ls_GetExtDelay reads the delay. For further information please refer to section 1.3.7.
If the function fails, the return value (*dwErr*) is non zero.

1.3.13 ls_GetCFG1

function ls_getcfg1(var ucCFG1 : Byte) : DWORD;

ls_GetCFG1 reads the configuration register 1. For further information please refer to section 1.3.8. If the function fails, the return value (*dwErr*) is non zero.

1.3.14 ls_SetADCPGA1

function ls_setadcpga1(wPGA : WORD) : DWORD;

There are three PGA registers for individually programming the gain of all 3 channels. ls_SetADCPGA1 sets PGA Gain register of first channel. Bits D8, D7, and D6 in each register must be set to zero, and Bits D5 through D0 control the gain range from 1× to 6× in 64 increments. The coding for the PGA registers is straight binary, with an all "zeros" word corresponding to the minimum gain setting (1×) and an all "ones" word corresponding to the maximum gain setting (6×).

D8	D7	D6	D5	D4	D3	D2	D1	D0	Gain (V/V)	Gain (dB)
0	0	MSB						LSB		
0	0	0	0	0	0	0	0	0	1.0	0.0
0	0	0	0	0	0	0	0	1	1.013	0.12
				•					•	•
				•					•	•
				•					•	•
0	0	0	1	1	1	1	1	0	5.56	14.9
0	0	0	1	1	1	1	1	1	6.0	15.56

The PGA Gain is approximately "linear in DB" and follows the equation:

$$Gain = \frac{6.0}{1 + 5.0\left[\frac{63 - G}{63}\right]}$$ where G is the register value (0 – 63).

If the function fails, the return value (*dwErr*) is non zero.

1.3.15 ls_SetADCPGA2

function ls_setadcpga2(wPGA : WORD) : DWORD;

ls_SetADCPGA2 sets PGA Gain register of second channel. For further information please refer to section 1.3.14. If the function fails, the return value (*dwErr*) is non zero.

1.3.16 ls_SetADCPGA3

function ls_setadcpga3(wPGA : WORD) : DWORD;

ls_SetADCPGA3 sets PGA Gain register of third channel. For further information please refer to section 1.3.14. If the function fails, the return value (*dwErr*) is non zero.

1.3.17 ls_GetADCPGA1

function ls_getadcpga1(var wPGA : WORD) : DWORD;

ls_GetADCPGA1 reads the value of the PGA Gain register of first channel. For further information please refer to section 1.3.14. If the function fails, the return value (*dwErr*) is non zero.

1.3.18 ls_GetADCPGA2

function ls_getadcpga2(var wPGA : WORD) : DWORD;

ls_GetADCPGA2 reads the value of the PGA Gain register of second channel. For further information please refer to section 1.3.14. If the function fails, the return value (*dwErr*) is non zero.

1.3.19 ls_GetADCPGA3

function ls_getadcpga3(var wPGA : WORD) : DWORD;

ls_GetADCPGA3 reads the value of the PGA Gain register of third channel. For further information please refer to section 1.3.14. If the function fails, the return value (*dwErr*) is non zero.

1.3.20 ls_SetADCOffSet1

function ls_setadcoffset1(wOffSet : WORD) : DWORD;

There are three Offset Registers for individually programming the offset of all 3 channels. ls_SetADCOffSet1 sets the Offset register of first channel. Bits D8 through D0 control the offset range from –300 mV to +300 mV in 512 increments. The coding for the Offset Registers is Sign Magnitude, with D8 as the sign bit. If the function fails, the return value (*dwErr*) is non zero.

D8	D7	D6	D5	D4	D3	D2	D1	D0	Offset (mV)
MSB								LSB	
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	+1.2
				•					•
				•					•
				•					•
0	1	1	1	1	1	1	1	1	+300
1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	-1.2
				•					•
				•					•
				•					•
1	1	1	1	1	1	1	1	1	-300

1.3.21 ls_SetADCOffSet2
function ls_setadcoffset2(wOffSet : WORD) : DWORD;
ls_SetADCOffSet2 sets the Offset register of second channel. For further information please refer to section 1.3.20. If the function fails, the return value (*dwErr*) is non zero.

1.3.22 ls_SetADCOffSet3
function ls_setadcoffset3(wOffSet : WORD) : DWORD;
ls_SetADCOffSet3 sets the Offset register of third channel. For further information please refer to section 1.3.20. If the function fails, the return value (*dwErr*) is non zero.

1.3.23 ls_GetADCOffSet1
function ls_getadcoffset1(var wOffSet : WORD) : DWORD;
ls_GetADCOffSet1 reads the value of the Offset register of first channel. For further information please refer to section 1.3.20. If the function fails, the return value (*dwErr*) is non zero.

1.3.24 ls_GetADCOffSet2
function ls_getadcoffset2(var wOffSet : WORD) : DWORD;
ls_GetADCOffSet2 reads the value of the Offset register of second channel. For further information please refer to section 1.3.20. If the function fails, the return value (*dwErr*) is non zero.

1.3.25 ls_GetADCOffSet3
function ls_getadcoffset3(var wOffSet : WORD) : DWORD;
ls_GetADCOffSet3 reads the value of the Offset register of third channel. For further information please refer to section 1.3.20. If the function fails, the return value (*dwErr*) is non zero.

1.3.26 ls_SetADCConfig
function ls_setadcconfig(wConfig : WORD) : DWORD;
ls_SetADCConfig sets the ADC configuration register.
A value of 0x0080 sets the input range of ADC to 4V, and a value of 0x0000 to 2V.
If the function fails, the return value (*dwErr*) is non zero.

1.3.27 ls_GetADCConfig
function ls_getadcconfig(var wConfig : WORD) : DWORD;
ls_GetADCConfig reads the ADC configuration register.
If the function fails, the return value (*dwErr*) is non zero.

1.3.28 ls_SaveSettings
function ls_savesettings : DWORD;
ls_SaveSettings saves all parameters / settings into EEPROM. If the function fails, the return value (*dwErr*) is non zero.