

**Interface Manual**  
**USBLC8M32.DLL**  
**USBLC8M64.DLL**  
**(V1.30)**

**Coptonix GmbH**

Falkentaler Steig 9

D – 13467 Berlin

Phone: +49 – (0)30 – 61 74 12 48

Fax: +49 – (0)30 – 61 74 12 47

[www.coptonix.com](http://www.coptonix.com)

[support@coptonix.com](mailto:support@coptonix.com)

# Contents

<b>Dynamic Link Library</b> .....	<b>4</b>
<b>1 Functions</b> .....	<b>4</b>
1.1 USB Functions.....	4
1.1.1 ls_GetErrorString.....	4
1.1.2 ls_Initialize.....	4
1.1.3 ls_SetPacketLength.....	4
1.1.4 ls_GetPacketLength.....	4
1.1.5 ls_EnumDevices.....	4
1.1.6 ls_OpenDeviceByIndex.....	5
1.1.7 ls_OpenDeviceBySerial.....	5
1.1.8 ls_CloseDevice.....	5
1.1.9 ls_DeviceCount.....	5
1.1.10 ls_CurrentDeviceIndex.....	5
1.1.11 ls_GetFWVersion.....	5
1.1.12 ls_GetVendorName.....	5
1.1.13 ls_GetProductName.....	5
1.1.14 ls_GetSerialNumber.....	5
1.1.15 ls_SetEPTimeOut.....	5
1.1.16 ls_GetEPTimeOut.....	6
1.2 Data Functions.....	6
1.2.1 ls_WaitForPipe.....	6
1.2.2 ls_GetPipe.....	6
1.2.3 ls_GetFPS.....	6
1.3 Camera Functions.....	6
1.3.1 ls_GetSensorType.....	6
1.3.2 ls_GetMCUSensorType.....	6
1.3.3 ls_GetSensorName.....	6
1.3.4 ls_SetMode.....	7
1.3.5 ls_SetState.....	7
1.3.6 ls_SetIntTime.....	7
1.3.7 ls_SetExtDelay.....	7
1.3.8 ls_SetSoftTrigTime.....	7
1.3.9 ls_SetCFG1.....	8
1.3.10 ls_GetMode.....	8
1.3.11 ls_GetState.....	9
1.3.12 ls_GetIntTime.....	9
1.3.13 ls_GetExtDelay.....	9
1.3.14 ls_GetSoftTrigTime.....	9
1.3.15 ls_GetMinMaxTrigTime.....	9
1.3.16 ls_GetCFG1.....	9
1.3.17 ls_SetADCPGA1.....	10
1.3.18 ls_SetADCPGA2.....	10
1.3.19 ls_SetADCPGA3.....	10
1.3.20 ls_GetADCPGA1.....	10
1.3.21 ls_GetADCPGA2.....	10
1.3.22 ls_GetADCPGA3.....	10
1.3.23 ls_SetADCOFFSet1.....	11
1.3.24 ls_SetADCOFFSet2.....	11
1.3.25 ls_SetADCOFFSet3.....	11

---

1.3.26 ls_GetADCOffSet1.....	11
1.3.27 ls_GetADCOffSet2.....	11
1.3.28 ls_GetADCOffSet3.....	11
1.3.29 ls_SetADCConfig.....	12
1.3.30 ls_GetADCConfig.....	12
1.3.31 ls_SaveSettings.....	12

---

# Dynamic Link Library DLL

## 1 Functions

### 1.1 USB Functions

#### 1.1.1 `Is_GetErrorString`

**function `Is_geterrorstring(dwErr : DWORD) : PAnsiChar;`**

`Is_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 `Is_Initialize`

**function `Is_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 `Is_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`.

`Is_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 succeeds, 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 `Is_SetPacketLength`

**function `Is_setpacketlength(dwPacketLength : DWORD) : DWORD;`**

`Is_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 `Is_GetPacketLength`

**function `Is_getpacketlength(var dwPacketLength : DWORD) : DWORD;`**

`Is_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 `Is_EnumDevices`

**function `Is_enumdevices : Integer;`**

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

### 1.1.6 Is\_OpenDeviceByIndex

**function Is\_opendevicbyindex(index : Integer) : DWORD;**

Is\_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 Is\_OpenDeviceBySerial

**function Is\_opendevicbyserial(pcserialnum : PAnsiChar) : DWORD;**

Is\_OpenDeviceBySerial connect a USB device and starts reading thread (see Is\_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 Is\_CloseDevice

**function Is\_closedevice : DWORD;**

“Is\_CloseDevice” disconnects the current opened device. If the function fails, the return value (*dwErr*) is non zero.

### 1.1.9 Is\_DeviceCount

**function Is\_devicecount : Byte;**

Is\_DeviceCount returns the number of USB devices, which are currently connected to the system.

### 1.1.10 Is\_CurrentDeviceIndex

**function Is\_currentdeviceindex : Integer;**

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

### 1.1.11 Is\_GetFWVersion

**function Is\_getfwversion(index : Integer) : WORD;**

Is\_GetFWVersion returns the version of the firmware with index “*index*”.

### 1.1.12 Is\_GetVendorName

**function Is\_getvendornam(index : Integer) : PAnsiChar;**

Is\_GetVendorName returns the vendor’s name of the device with index “*index*”.

### 1.1.13 Is\_GetProductName

**function Is\_getproductnam(index : Integer) : PAnsiChar;**

Is\_GetProductName returns the product’s name of the device with index “*index*”.

### 1.1.14 Is\_GetSerialNumber

**function Is\_getserialnum(index : Integer) : PAnsiChar;**

Is\_GetSerialNumber the serial number of the device with index “*index*”.

### 1.1.15 Is\_SetEPTimeOut

**function Is\_seteptimeout(dwtimeout : DWORD) : DWORD;**

Is\_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 Is\_GetEPTimeOut

**function Is\_geteptimeout : DWORD;**

Is\_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 Is\_WaitForPipe

**function Is\_waitforpipe(dwTimeOut : DWORD) : DWORD;**

Is\_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 Is\_GetPipe

**function Is\_getpipe(lpBuffer : Pointer; dwToRead: DWORD;  
var dwRead: DWORD): DWORD;**

Is\_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 Is\_GetFPS

**function Is\_getfps : DWORD;**

Is\_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 Is\_GetSensorType

**function Is\_getsensortype(Var wSensorType, wPixelCount : Word) : DWORD;**

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

### 1.3.2 Is\_GetMCUSensorType

**function Is\_getmcusensortype(Var wSensorType : WORD) : DWORD;**

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

### 1.3.3 Is\_GetSensorName

**function Is\_getsensorname(wSensorType : WORD) : PAnsiChar;**

Is\_GetSensorName converts the sensor's type to a readable zero terminated string.

### 1.3.4 Is\_SetMode

**function Is\_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.
SOFT_TRIGGER	0x04	Acquisition is done on internal software trigger.

In EXT\_EXP\_CTRL the time between the negative edges determines the integration time.

Only firmware 1v20 and later supports SOFT\_TRIGGER mode.

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

### 1.3.5 Is\_SetState

**function Is\_setstate(ucState : Byte) : DWORD;**

Is\_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 Is\_SetIntTime

**function Is\_setinttime(dwIntTime : DWORD) : DWORD;**

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

### 1.3.7 Is\_SetExtDelay

**function Is\_setextdelay(dwExtDelay : DWORD) : DWORD;**

Is\_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 Is\_SetSoftTrigTime

**function Is\_setsofttrigtime(dwSftTrgTime : DWORD) : DWORD;**

Is\_SetSoftTrigTime sets the time period *T* [μs] of the internal software trigger. The line rate is reciprocal of time period ***f = 1/T [lines / second]***. Software trigger time of 500μs is equivalent to 2000 lines / second. The smallest time period / highest line rate is sensor depended. The time period must be greater than the integration/exposure time.

This function is supported by firmware version 1v20 and later.

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

### 1.3.9 Is\_SetCFG1

**function Is\_setcfg1(ucCFG1 : Byte) : DWORD;**

Is\_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 determines the value of <i>dwPacketLength</i> used in functions <i>Is_initialize</i> and <i>Is_setpacketlength</i> . 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.

### 1.3.10 Is\_GetMode

**function Is\_getmode(Var ucMode : Byte) : DWORD;**

Is\_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.
SOFT_TRIGGER	0x04	Acquisition is done on internal software trigger.

Only firmware 1v20 and later supports SOFT\_TRIGGER mode.

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



### 1.3.11 Is\_GetState

**function Is\_getstate(Var ucState : Byte) : DWORD;**

Is\_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.

### 1.3.12 Is\_GetIntTime

**function Is\_getinttime(Var dwIntTime : DWORD) : DWORD;**

Is\_GetIntTime returns the integration/exposure time “*dwIntTime*” in microseconds. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.13 Is\_GetExtDelay

**function Is\_getextdelay(var dwExtDelay : DWORD) : DWORD;**

Is\_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.

### 1.3.14 Is\_GetSoftTrigTime

**function Is\_getsofttrigtime(var dwSftTrgTime : DWORD) : DWORD;**

Is\_GetSoftTrigTime reads the software trigger time [μs]. For further information please refer to section 1.3.8. *This function is supported by firmware version 1v20 and later.*

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

### 1.3.15 Is\_GetMinMaxTrigTime

**function Is\_getminmaxtrigtime(var dwMinTime, dwMaxTime : DWORD) : DWORD;**

Is\_GetMinMaxTrigTime reads the lowest and highest possible software trigger time [μs] for the used sensor. For further information please refer to section 1.3.8. *This function is supported by firmware version 1v22 and later.*

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

### 1.3.16 Is\_GetCFG1

**function Is\_getcfg1(var ucCFG1 : Byte) : DWORD;**

Is\_GetCFG1 reads the configuration register 1. For further information please refer to section 1.3.9. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.17 Is\_SetADCPGA1

**function Is\_setadcpga1(wPGA : WORD) : DWORD;**

There are three PGA registers for individually programming the gain of all 3 channels. Is\_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.

### 1.3.18 Is\_SetADCPGA2

**function Is\_setadcpga2(wPGA : WORD) : DWORD;**

Is\_SetADCPGA2 sets PGA Gain register of second channel. For further information please refer to section 1.3.17. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.19 Is\_SetADCPGA3

**function Is\_setadcpga3(wPGA : WORD) : DWORD;**

Is\_SetADCPGA3 sets PGA Gain register of third channel. For further information please refer to section 1.3.17. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.20 Is\_GetADCPGA1

**function Is\_getadcpga1(var wPGA : WORD) : DWORD;**

Is\_GetADCPGA1 reads the value of the PGA Gain register of first channel. For further information please refer to section 1.3.17. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.21 Is\_GetADCPGA2

**function Is\_getadcpga2(var wPGA : WORD) : DWORD;**

Is\_GetADCPGA2 reads the value of the PGA Gain register of second channel. For further information please refer to section 1.3.17. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.22 Is\_GetADCPGA3

**function Is\_getadcpga3(var wPGA : WORD) : DWORD;**

Is\_GetADCPGA3 reads the value of the PGA Gain register of third channel. For further information please refer to section 1.3.17. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.23 Is\_SetADCOffSet1

**function Is\_setadcoffset1(wOffset : WORD) : DWORD;**

There are three Offset Registers for individually programming the offset of all 3 channels. Is\_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

### 1.3.24 Is\_SetADCOffSet2

**function Is\_setadcoffset2(wOffset : WORD) : DWORD;**

Is\_SetADCOffSet2 sets the Offset register of second channel. For further information please refer to section 1.3.23. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.25 Is\_SetADCOffSet3

**function Is\_setadcoffset3(wOffset : WORD) : DWORD;**

Is\_SetADCOffSet3 sets the Offset register of third channel. For further information please refer to section 1.3.23. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.26 Is\_GetADCOffSet1

**function Is\_getadcoffset1(var wOffset : WORD) : DWORD;**

Is\_GetADCOffSet1 reads the value of the Offset register of first channel. For further information please refer to section 1.3.23. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.27 Is\_GetADCOffSet2

**function Is\_getadcoffset2(var wOffset : WORD) : DWORD;**

Is\_GetADCOffSet2 reads the value of the Offset register of second channel. For further information please refer to section 1.3.23. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.28 Is\_GetADCOffSet3

**function Is\_getadcoffset3(var wOffset : WORD) : DWORD;**

Is\_GetADCOffSet3 reads the value of the Offset register of third channel. For further information please refer to section 1.3.23. If the function fails, the return value (*dwErr*) is non zero.

### 1.3.29 Is\_SetADCConfig

**function Is\_setadcconfig(wConfig : WORD) : DWORD;**

Is\_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.

### 1.3.30 Is\_GetADCConfig

**function Is\_getadcconfig(var wConfig : WORD) : DWORD;**

Is\_GetADCConfig reads the ADC configuration register.

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

### 1.3.31 Is\_SaveSettings

**function Is\_savesettings : DWORD;**

Is\_SaveSettings saves all parameters / settings into EEPROM. If the function fails, the return value (*dwErr*) is non zero.