Interface Manual
LibUSBLC8M64.so
LibUSBLC8MPI.so

(V1.03)

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 Library

In order to use this dynamic library and to access the USB Line Camera you need to install libusb-1.0. To install libusb directly from the repository run the below command from the terminal:

sudo apt-get install libusb-1.0-0-dev

Usually running Linux libusb applications need root privilege. To run Linux libusb applications **without** root privilege you need to use *udev rules* (For further information about *udev rules* please visit: https://wiki.debian.org/udev). The *.rules file should contain the rules below:

dSUBSYSTEM !="usb_device", ACTION !="add", GOTO="usblc_rules_end"
SYSFS{idVendor} =="19d1", SYSFS{idProduct} =="000f"
MODE="0666", OWNER="USER_NAME", GROUP="root"
LABEL="usblc_rules_end"

1 Functions

1.1 USB Functions

1.1.1 ls_GetErrorString

const char* ls_geterrorstring(int32 ierr);

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

1.1.2 ls_Initialize

void ls_initialize(int32 pipesize, int32 packetlength);

When starting the application, this function is called when the default values are not sufficient. The argument *pipesize* defines the size of a ring buffer (pipe). If *pipesize* is equal to 512, it means 512 bytes buffer and 67108864 is equal to 64 Mbytes. The default value is 4MB. *packetLength* is the number of bytes to be read per read request from the hardware FIFO. The value of *packetlength* 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 *packetLength* to multiple of this value.

1.1.3 ls_SetPacketLength

int32 ls_setpacketlength(int32 packetlength);

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

1.1.4 ls_GetPacketLength

int32 ls_getpacketlength(int32 &packetlength, uint32 timeout);

ls_GetPacketLength reads the recommended value for *PacketLength* from the line sensor controller. *PacketLength* is described in section 1.1.2. 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 (*ierr*) is non zero.

1.1.5 ls_EnumDevices
int32 ls_enumdevices();
ls_EnumDevices enumerates and creates a list of all connected devices and then returns the number of connected devices.

1.1.6 ls_OpenDeviceByIndex
int32 ls_opendevicebyindex(int32 index);
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 (*ierr*) is non zero.

1.1.7 ls_OpenDeviceBySerial
int32 ls_opendevicebyserial(char* pcserialnum);
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 (*ierr*) is non zero.

1.1.8 ls_CloseDevice
int32 ls_closedevice();
"ls_CloseDevice" disconnects the current opened device. If the function fails, the return value (*ierr*) is non zero.

1.1.9 ls_DeviceCount
uint8 ls_devicecount();
ls_DeviceCount returns the number of USB devices, which are currently connected to the system.

1.1.10 ls_CurrentDeviceIndex
int32 ls_currentdeviceindex();
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
uint32 ls_getfwversion(int32 index);
ls_GetMCU1Version returns the version of the firmware with index "***index***".

1.1.12 ls_GetVendorName
const char* ls_getvendorname(int32 index);
ls_GetVendorName returns the vendor's name of the device with index "***index***".

1.1.13 ls_GetProductName
const char* ls_getproductname(int32 index);
ls_GetProductName returns the product's name of the device with index "***index***".

1.1.14 ls_GetSerialNumber
const char* ls_getserialnumber(int32 index);
ls_GetSerialNumber the serial number of the device with index "***index***".

1.2 Data Functions

1.2.1 ls_WaitForPipe

void ls_waitforpipe(uint32 timeout) ;

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. *timeout* is the time out interval. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s.

1.2.2 ls_WaitForPipeCount

void ls_waitforpipecount(int32 numberofbytes, uint32 timeout) ;

ls_WaitForPipeCount checks whether the specified number of bytes are available for reading. If less or no data are available, the calling thread enters the wait state until data is received or the time-out interval elapses. *timeout* is the time out interval. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s.

1.2.3 ls_GetPipe

int32 ls_getpipe(void* lpbuffer, int32 numberbytes);

ls_GetPipe reads data from the pipe (ring buffer). The argument *lpbuffer* points to the buffer, which has to include the data. *numberbytes* specifies the length of the data which must be read. The function returns the actual number of bytes read. If *numberbytes* is specified with 0, then the function returns the actual number of bytes available without reading data.

1.2.4 ls_GetFPS

uint32 ls_getfps();

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

int32 ls_getsensortype(uint16 &sensortype, uint16 &pixelcount, uint32 timeout);

ls_GetSensorType reads the type of the sensor and the sensor's number of pixels from the sensor circuit board. See also "ls_GetMCUSensorType". *timeout* 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 (*ierr*) is non zero.

1.3.2 ls_GetMCUSensorType

int32 ls_getmcusensortype(uint16 &sensortype, uint32 timeout) ;

ls_GetMCUSensorType reads the type of the sensor supported by the main circuit board. *timeout* is the time out interval. The time-out value will be expected in 1 ms units. A value of 1000 corresponds to 1 s. See also "ls_GetSensorType". If the function fails, the return value (*ierr*) is non zero.

1.3.3 ls_GetSensorName

const char* ls_getsensorname(uint16 sensortype);

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

1.3.4 ls_SetMode

int32 ls_setmode(uint8 ucmode, uint32 timeout);

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.

timeout 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 (*ierr*) is non zero.

1.3.5 ls_SetState

int32 ls_setstate(uint8 ucstate, uint32 timeout);

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. *timeout* 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 (*ierr*) is non zero.

1.3.6 ls_SetIntTime

int32 ls_setinttime(uint32 inttime, uint32 timeout);

ls_SetIntTime sets the integration/exposure time *inttime* in microseconds. *timeout* 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 (*ierr*) is non zero.

1.3.7 ls_SetExtDelay

int32 ls_setextdelay(uint32 extdelay, uint32 timeout);

ls_SetExtDelay sets a time delay between the external trigger and the start of integration. The delay time is equal to ***extdelay* 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.
timeout 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 (*ierr*) is non zero.

1.3.8 ls_SetCFG1

int32 ls_setcfg1(uint8 uccfg1, uint32 timeout);

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.

timeout 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 (*ierr*) is non zero.

1.3.9 ls_GetMode

int32 ls_getmode(uint8 &ucmode, uint32 timeout);

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.

timeout 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 (*ierr*) is non zero.

1.3.10 ls_GetState

int32 ls_getstate(uint8 &ucstate, uint32 timeout);

ls_GetState returns the current state "*ucstate*":

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

timeout 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 (*ierr*) is non zero.

1.3.11 ls_GetIntTime

int32 ls_getinttime(uint32 &inttime, uint32 timeout);

ls_GetIntTime returns the integration/exposure time "*inttime*" in microseconds. *timeout* 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 (*ierr*) is non zero.

1.3.12 ls_GetExtDelay

int32 ls_getextdelay(uint32 extdelay, uint32 timeout);

ls_GetExtDelay reads the delay. For further information please refer to section 1.3.7.
timeout 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 (*ierr*) is non zero.

1.3.13 ls_GetCFG1

int32 ls_getcfg1(uint8 uccfg1, uint32 timeout);

ls_GetCFG1 reads the configuration register 1. For further information please refer to section 1.3.8. *timeout* 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 (*ierr*) is non zero.

1.3.14 ls_SetADCPGA1

int32 ls_setadcpga1(uint16 wPGA, uint32 timeout);

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).

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 (*ierr*) is non zero.

1.3.15 ls_SetADCPGA2

int32 ls_setadcpga2(uint16 wPGA, uint32 timeout);

ls_SetADCPGA2 sets PGA Gain register of second channel. For further information please refer to section 1.3.14. 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 (*ierr*) is non zero.

1.3.16 ls_SetADCPGA3

int32 ls_setadcpga3(uint16 wPGA, uint32 timeout);

ls_SetADCPGA3 sets PGA Gain register of third channel. For further information please refer to section 1.3.14. 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 (*ierr*) is non zero.

1.3.17 ls_GetADCPGA1

int32 ls_getadcpga1(uint16 &wPGA, uint32 timeout);

ls_GetADCPGA1 reads the value of the PGA Gain register of first channel. For further information please refer to section 1.3.12. 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 (*ierr*) is non zero.

1.3.18 ls_GetADCPGA2

int32 ls_getadcpga2(uint16 &wPGA, uint32 timeout);

ls_GetADCPGA2 reads the value of the PGA Gain register of second channel. For further information please refer to section 1.3.14. 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 (*ierr*) is non zero.

1.3.19 ls_GetADCPGA3

int32 ls_getadcpga3(uint16 &wPGA, uint32 timeout);

ls_GetADCPGA3 reads the value of the PGA Gain register of third channel. For further information please refer to section 1.3.14. 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 (*iErr*) is non zero.

1.3.20 ls_SetADCOffSet1

int32 ls_setadcoffset1(uint16 wOffSet, uint32 timeout);

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 (*ierr*) 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

int32 ls_setadcoffset2(uint16 wOffSet, uint32 timeout);

ls_SetADCOffSet2 sets the Offset register of second channel. For further information please refer to section 1.3.19. 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 (*ierr*) is non zero.

1.3.22 ls_SetADCOffSet3

int32 ls_setadcoffset3(uint16 wOffSet, uint32 timeout);

ls_SetADCOffSet3 sets the Offset register of third channel. For further information please refer to section 1.3.19. 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 (*ierr*) is non zero.

1.3.23 ls_GetADCOffSet1

int32 ls_getadcoffset1(uint16 &wOffSet, uint32 timeout);

ls_GetADCOffSet1 reads the value of the Offset register of first channel. For further information please refer to section 1.3.19. 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 (*ierr*) is non zero.

1.3.24 ls_GetADCOffSet2

int32 ls_getadcoffset2(uint16 &wOffSet, uint32 timeout);

ls_GetADCOffSet2 reads the value of the Offset register of second channel. For further information please refer to section 1.3.19. 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 (*ierr*) is non zero.

1.3.25 ls_GetADCOffSet3

int32 ls_getadcoffset3(uint16 &wOffSet, uint32 timeout);

ls_GetADCOffSet3 reads the value of the Offset register of third channel. For further information please refer to section 1.3.19. 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 (*ierr*) is non zero.

1.3.26 ls_SetADCConfig

int32 ls_setadcconfig(uint16 wConfig, uint32 timeout);

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 (*ierr*) is non zero.

1.3.27 ls_GetADCConfig

int32 ls_getadcconfig(uint16 &wConfig, uint32 timeout);

ls_GetADCConfig reads the ADC configuration register.
If the function fails, the return value (*ierr*) is non zero.

1.3.28 ls_SaveSettings

int32 ls_savesettings(uint32 timeout);

ls_SaveSettings saves all parameters / settings into EEPROM. 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 (*ierr*) is non zero.