Interface Manual
USBLC3301x64.so
USBLC3301PI.so

(V1.00)

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
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"
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.
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. 160000) 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_GetPixelCount

uint16 ls_getpixelcount();

ls_GetPixelCount returns the number of pixels of the sensor (102 pixels).

1.3.2 ls_GetSensorName

const char* ls_getsensorname();

ls_GetSensorName returns the name of the sensor “TSL3301”.

1.3.3 ls_SetMode

int32 ls_setmode(uint8 ucmode, uint32 timeout);

There are 3 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.

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.

1.3.4 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.

1.3.5 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.

1.3.6 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[0:3]		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*.
	0000 = 0	1 image / USB transfer.
	0001 = 1	2 images / USB transfer
	….	
	1110 = 14	15 images / USB transfer
	1111 = 15	16 images / USB transfer
Bit[4:7]		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.

1.3.7 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.

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_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.

1.3.9 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.

1.3.10 ls_GetCFG1

int32 ls_getcfg1(uint8 uccfg1, uint32 timeout);

ls_GetCFG1 reads the configuration register 1. For further information please refer to section 1.3.6. *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_SetGain

int32 ls_setgain(uint8 ucleft, uint8 uccenter, uint8 ucright, uint32 timeout);

The sensor is divided into three 34-pixel zones (*ucleft, uccenter, ucright*), with each zone having programmable gain correction. The table below lists the gain settings and the corresponding pixel values. 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.

Gain Code	Rel. Gain	% Increase	Gain Code	Rel. Gain	% Increase
0	1		**16**	1.52	3.23
1	1.02	2.17	**17**	1.57	3.33
2	1.05	2.22	**18**	1.62	3.45
3	1.07	2.27	**19**	1.68	3.57
4	1.09	2.33	**20**	1.74	3.70
5	1.12	2.38	**21**	1.81	3.85
6	1.15	2.44	**22**	1.88	4.00
7	1.18	2.50	**23**	1.96	4.17
8	1.21	2.56	**24**	2.05	4.35
9	1.24	2.63	**25**	2.14	4.55
10	1.27	2.70	**26**	2.24	4.76
11	1.31	2.78	**27**	2.35	5.00
12	1.34	2.86	**28**	2.48	5.26
13	1.38	2.94	**29**	2.61	5.56
14	1.43	3.03	**30**	2.77	5.88
15	1.47	3.13	**31**	2.94	6.25

1.3.12 ls_GetGain

int32 ls_getgain(uint8 &ucleft, uint8 &uccenter, uint8 &ucright, uint32 timeout);

ls_GetGain reads the values of all 3 gain registers of the sensor. For further information please refer to section 1.3.11. 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_SetOffSet

int32 ls_setoffset(uint8 ucleft, uint8 uccenter, uint8 ucright, uint32 timeout));

The sensor is divided into three 34-pixel zones (*ucleft, uccenter, ucright*), with each zone having programmable gain correction. Codes 0h to 7Fh corresponds to positive offset values and codes 80h to FFh corresponds to negative offset values. Offset is affected by the gain settings and may have to be adjusted after gain changes are made. The offset correction is proportional to the gain setting. At minimal gain, one LSB of the offset corresponds to approximately 1/3 LSB of the device output, and at maximum gain, to about 1 LSB of the device output. 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_GetOffSet

int32 ls_getoffset(uint8 &ucleft, uint8 &uccenter, uint8 &ucright, uint32 timeout);

ls_GetOffSet reads the values of the Offset registers (*ucleft, uccenter, ucright*). For further information please refer to section 1.3.13. 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_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.