Interface Manual
USBLC32.DLL

(V1.09)

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

Inhaltsverzeichnis

Dynamic Link Library DLL

1 Functions

1.1 USB Functions

1.1.1 ls_GetErrorString

function ls_geterrorstring(dwErr : DWORD) : PChar;

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 : PChar): 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 = 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 (number of pixels x 2), e.g. 4096 for a sensor with 2048 pixels. The default value is 4096 Bytes for a 2048 pixel sensor. 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 PChar 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_EnumDevices

function ls_enumdevices : Integer;

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

1.1.5 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.

1.1.6 ls_OpenDeviceBySerial

function ls_opendevicebyserial(pcserialnum : PChar) : DWORD;

ls_OpenDeviceBySerial connect a USB device and starts reading thread (see ls_OpenDeviceByIndex). The argument ***pcserialnum*** is the serial number (e.g. 1400000) of a device. If the function fails, the return value (*dwErr*) is non zero.

1.1.7 ls_CloseDevice

function ls_closedevice : DWORD;

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

1.1.8 ls_DeviceCount

function ls_devicecount : Byte;

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

1.1.9 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.

1.1.10 ls_GetMCU1Version

function ls_getmcu1version(index : Integer) : WORD;

ls_GetMCU1Version returns the version of the device (USB controller) with index "***index***".

1.1.11 ls_GetVendorName

function ls_getvendorname(index : Integer) : PChar;

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

1.1.12 ls_GetProductName

function ls_getproductname(index : Integer) : PChar;

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

1.1.13 ls_GetSerialNumber

function ls_getserialnumber(index : Integer) : PChar;

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

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_ResetFiFo

function ls_resetfifo : DWORD;

ls_ResetFiFo resets the hardware FIFO without reading it.
If the function fails, the return value (*dwErr*) is non zero.

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_GetMCU2SensorType".
If the function fails, the return value (*dwErr*) is non zero.

1.3.2 ls_GetSensorName

function ls_getsensorname(wSensorType : WORD) : PChar;

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

1.3.3 ls_SetMode

function ls_setmode(ucMode, ucTimeOut : Byte) : DWORD;

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.

The time-out value will be expected in 100 ms units. A value of 10 corresponds to 1 sec. The maximal programmable time-out interval is 25 seconds, because an 8 Bit variable (*ucTimeOut*) is used.

e.g.

```
const _100ms = 1;                 // 100 ms time-out interval
const _1s = _100ms * 10;          // 1 sec. time-out interval
const ucTimeOut = _1s * 2;        // 2 sec. time-out interval
```

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

1.3.4 ls_SetState

function ls_setstate(ucState, ucTimeOut : 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.

1.3.5 ls_SetIntTime

function ls_setinttime(dwIntTime : DWORD; ucTimeOut : Byte) : DWORD;

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

1.3.6 ls_GetMCU2Version

function ls_getmcu2version(var wVersion : WORD; ucTimeOut : Byte): WORD;

ls_GetMCU2Version reads the version "*wVersion*"of the line sensor controller.

1.3.7 ls_GetMCU2SensorType

function ls_getmcu2sensortype(Var wSensorType : WORD;
ucTimeOut : Byte) : DWORD;

ls_GetMCU2SensorType 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.

1.3.8 ls_GetMode

function ls_getmode(Var ucMode : Byte; ucTimeOut : 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.

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

1.3.9 ls_GetState

function ls_getstate(Var ucState : Byte; ucTimeOut : 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.

1.3.10 ls_GetIntTime

function ls_getinttime(Var dwIntTime : DWORD; ucTimeOut : Byte) : DWORD;

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

1.3.11 ls_GetPacketLength

function ls_setpacketlength(var dwPacketLength : DWORD;
ucTimeOut : Byte) : 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.

1.3.12 ls_SetADCPGA

function ls_setadcpga(wRPGA, wGPGA, wBPGA : WORD) : DWORD;

There are three PGA registers for individually programming the gain of all 3 channels. *wPGA1* corresponds to the first channel, *wPGA2* to the second channel, and *wPGA3* to the third 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.

1.3.13 ls_SetADC3xPGA

function ls_setadc3xpga(wPGA : WORD) : DWORD;

ls_SetADC3xPGA sets all PGA Gain registers with same value. For further information please refer to section 1.3.12. If the function fails, the return value (*dwErr*) is non zero.

1.3.14 ls_SetADCPGA1

function ls_setadcpga1(wPGA : WORD) : DWORD;

ls_SetADCPGA1 sets PGA Gain register of first channel. For further information please refer to section 1.3.12. 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 1.3.12. 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 1.3.12. 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 1.3.12. 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 1.3.12. 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 1.3.12. If the function fails, the return value (*dwErr*) is non zero.

1.3.20 ls_SetADCOffset

function ls_setadcoffset(wROffSet, wGOffSet, wBOffSet : WORD) : DWORD;

There are three Offset Registers for individually programming the offset of all 3 channels. *wOffSet1* corresponds to the first channel, *wOffSet2* to the second channel, and *wOffSet3* to the third 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_SetADC3xOffSet

function ls_setadc3xoffset(wOffSet : WORD) : DWORD;

ls_SetADC3xOffSet sets all Offset registers with same value. 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_SetADCOffSet1

function ls_setadcoffset1(wOffSet : WORD) : DWORD;

ls_SetADCOffSet1 sets 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.

1.3.23 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.

1.3.24 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.

1.3.25 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.

1.3.26 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.

1.3.27 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.

1.3.28 ls_SetADCConfig
function ls_setadcconfig(wConfig : WORD) : DWORD;

1.3.29 ls_GetADCConfig
function ls_getadcconfig(var wConfig : WORD) : DWORD;

1.3.30 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.

1.4 I2C-Bus Functions

1.4.1 ls_WriteI2C
function ls_writei2c(I2C_Addr : Byte; pBuf : Pointer; wLength : WORD;
Var ucState : Byte) : DWORD;
Ls_WriteI2C is used to address and write data to I2C slave devices. *I2C_Addr* is the address of a I2C slave device. The LSB of the address is set to “0” by hardware. *pBuf* is the pointer that points to data to write to I2C slave devices. *wLength* is the number bytes to write. *ucStatus* returns the state of the I2C bus. If the function fails, the return value (*dwErr*) is non zero.

1.4.2 ls_ReadI2C
function ls_readi2c(I2C_Addr : Byte; pBuf : Pointer; Var wLength : WORD;
Var ucState : Byte) : DWORD;
Ls_ReadI2C is used to address and read data from I2C slave devices. *I2C_Addr* is the address of a I2C slave device. *PBuf* returns a pointer to an array of byte. This array contains data were read. *wLength* is the number bytes to read. *wLength* returns also the numbers of bytes were read. *ucStatus* returns the state of the I2C bus. If the function fails, the return value (*dwErr*) is non zero.

1.4.3 ls_SetI2CFreq

function ls_seti2cfreq(freq : Byte) : DWORD;

ls_SetI2CFreq sets a new value for the I2C Clock frequency. If *ucfreq* = 0, the I2C bus operates at approximately 100 kHz, if *ucfreq* = 1, the I2C bus operates at approximately 400 kHz. If the function fails, the return value (*dwErr*) is non zero.

1.4.4 ls_GetI2CFreq

function ls_geti2cfreq(Var freq : Byte) : DWORD;

ls_GetI2CFreq reads the actual I2C Clock frequency. *ucfreq* returns the actual I2C Clock frequency. For further information please refer to section 1.4.3. If the function fails, the return value (*dwErr*) is non zero.

1.4.5 ls_GetI2CStat

function ls_geti2cstat(Var STAT : Byte) : DWORD;

ls_GetI2CStat reads the state/result of the last I2C read/write operation. Use ls_GetI2CString to obtain the status string. If the function fails, the return value (*dwErr*) is non zero.

1.4.6 ls_GetI2CString

function ls_geti2cstring(STAT : Byte) : PChar;

To obtain a status string of the last I2C read/write operation, use the function ls_GetI2CString. *ucStat* is the value returned when calling the functions ls_Writei2C and ls_Readi2C. The return value is the status string.