Interface Manual
LibUSBLC64.so
LibUSBLC32.so
(V1.00)
## Dynamic Library

### Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 USB Functions</td>
<td></td>
</tr>
<tr>
<td>1.1.1</td>
<td>ls_GetErrorString</td>
</tr>
<tr>
<td>1.1.2</td>
<td>ls_Initialize</td>
</tr>
<tr>
<td>1.1.3</td>
<td>ls_SetPacketLength</td>
</tr>
<tr>
<td>1.1.4</td>
<td>ls_EnumDevices</td>
</tr>
<tr>
<td>1.1.5</td>
<td>ls_OpenDeviceByIndex</td>
</tr>
<tr>
<td>1.1.6</td>
<td>ls_OpenDeviceBySerial</td>
</tr>
<tr>
<td>1.1.7</td>
<td>ls_CloseDevice</td>
</tr>
<tr>
<td>1.1.8</td>
<td>ls_DeviceCount</td>
</tr>
<tr>
<td>1.1.9</td>
<td>ls_CurrentDeviceIndex</td>
</tr>
<tr>
<td>1.1.10</td>
<td>ls_GetMCU1Version</td>
</tr>
<tr>
<td>1.1.11</td>
<td>ls_GetVendorName</td>
</tr>
<tr>
<td>1.1.12</td>
<td>ls_GetProductName</td>
</tr>
<tr>
<td>1.1.13</td>
<td>ls_GetSerialNumber</td>
</tr>
<tr>
<td>1.1.14</td>
<td>ls_OpenDeviceBySerial</td>
</tr>
<tr>
<td>1.1.15</td>
<td>ls_OpenDeviceByIndex</td>
</tr>
<tr>
<td>1.1.16</td>
<td>ls_EnumDevices</td>
</tr>
<tr>
<td>1.1.17</td>
<td>ls_Initialize</td>
</tr>
<tr>
<td>1.1.18</td>
<td>ls_GetErrorString</td>
</tr>
<tr>
<td>1.1.19</td>
<td>ls_Initialize</td>
</tr>
<tr>
<td>1.1.20</td>
<td>ls_GetSerialNumber</td>
</tr>
<tr>
<td>1.1.21</td>
<td>ls_GetProductName</td>
</tr>
<tr>
<td>1.1.22</td>
<td>ls_GetVendorName</td>
</tr>
<tr>
<td>1.1.23</td>
<td>ls_GetMCU1Version</td>
</tr>
<tr>
<td>1.1.24</td>
<td>ls_GetMCU2Version</td>
</tr>
<tr>
<td>1.1.25</td>
<td>ls_GetMCU2SensorType</td>
</tr>
<tr>
<td>1.1.26</td>
<td>ls_GetMode</td>
</tr>
<tr>
<td>1.1.27</td>
<td>ls_GetState</td>
</tr>
<tr>
<td>1.1.28</td>
<td>ls_GetIntTime</td>
</tr>
<tr>
<td>1.1.29</td>
<td>ls_GetADCConfig</td>
</tr>
<tr>
<td>1.2 Data Functions</td>
<td></td>
</tr>
<tr>
<td>1.2.1</td>
<td>ls_WaitForPipe</td>
</tr>
<tr>
<td>1.2.2</td>
<td>ls_GetPipe</td>
</tr>
<tr>
<td>1.2.3</td>
<td>ls_ResetFiFo</td>
</tr>
<tr>
<td>1.3 Camera Functions</td>
<td></td>
</tr>
<tr>
<td>1.3.1</td>
<td>ls_GetSensorType</td>
</tr>
<tr>
<td>1.3.2</td>
<td>ls_GetSensorName</td>
</tr>
<tr>
<td>1.3.3</td>
<td>ls_SetMode</td>
</tr>
<tr>
<td>1.3.4</td>
<td>ls_SetState</td>
</tr>
<tr>
<td>1.3.5</td>
<td>ls_SetIntTime</td>
</tr>
<tr>
<td>1.3.6</td>
<td>ls_GetMCU2Version</td>
</tr>
<tr>
<td>1.3.7</td>
<td>ls_GetMCU2SensorType</td>
</tr>
<tr>
<td>1.3.8</td>
<td>ls_GetMode</td>
</tr>
<tr>
<td>1.3.9</td>
<td>ls_GetState</td>
</tr>
<tr>
<td>1.3.10</td>
<td>ls_GetIntTime</td>
</tr>
<tr>
<td>1.3.11</td>
<td>ls_GetPacketLength</td>
</tr>
<tr>
<td>1.3.12</td>
<td>ls_GetADCPGA</td>
</tr>
<tr>
<td>1.3.13</td>
<td>ls_GetADCPGA1</td>
</tr>
<tr>
<td>1.3.14</td>
<td>ls_GetADCPGA2</td>
</tr>
<tr>
<td>1.3.15</td>
<td>ls_GetADCPGA3</td>
</tr>
<tr>
<td>1.3.16</td>
<td>ls_GetADCOffset</td>
</tr>
<tr>
<td>1.3.17</td>
<td>ls_GetADCOffset1</td>
</tr>
<tr>
<td>1.3.18</td>
<td>ls_GetADCOffset2</td>
</tr>
<tr>
<td>1.3.19</td>
<td>ls_GetADCOffset3</td>
</tr>
<tr>
<td>1.3.20</td>
<td>ls_GetADCOffSet</td>
</tr>
<tr>
<td>1.3.21</td>
<td>ls_GetADCOffSet1</td>
</tr>
<tr>
<td>1.3.22</td>
<td>ls_GetADCOffSet2</td>
</tr>
<tr>
<td>1.3.23</td>
<td>ls_GetADCOffSet3</td>
</tr>
<tr>
<td>1.3.24</td>
<td>ls_GetADCOffSet4</td>
</tr>
<tr>
<td>1.3.25</td>
<td>ls_GetADCOffSet5</td>
</tr>
<tr>
<td>1.3.26</td>
<td>ls_GetADCOffSet6</td>
</tr>
<tr>
<td>1.3.27</td>
<td>ls_GetADCOffSet7</td>
</tr>
<tr>
<td>1.3.28</td>
<td>ls_GetADCOffSet8</td>
</tr>
<tr>
<td>1.3.29</td>
<td>ls_GetADCOffSet9</td>
</tr>
</tbody>
</table>

---

Coptonix GmbH • Luxemburger Str. 31 • D – 13353 Berlin

February 2016
1.3.30 ls_SaveSettings..................................................................................................................10
1.4 I2C-Bus Functions................................................................................................................11
  1.4.1 ls_WriteI2C..................................................................................................................11
  1.4.2 ls_ReadI2C...................................................................................................................11
  1.4.3 ls_SetI2CFreq...............................................................................................................11
  1.4.4 ls_GetI2CFreq...............................................................................................................11
  1.4.5 ls_GetI2CStat...............................................................................................................11
  1.4.6 ls_GetI2CString..........................................................................................................11
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:

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

1 Functions

1.1 USB Functions

1.1.1 ls_GetErrorString

```pascal
function ls_geterrorstring(iErr : integer) : PChar;
```

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

```pascal
procedure ls_initialize(pipesize, packetlength : integer);
```

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 = 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 (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.

1.1.3 ls_SetPacketLength

```pascal
function ls_setpacketlength(packetlength : integer) : integer;
```

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_EnumDevices

```pascal
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

```pascal
function ls_opendevicebyindex(index : Integer) : integer;
```

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.
1.1.6 ls_OpenDeviceBySerial
function ls_opendevicebyserial(pcserialnum : PChar) : integer;
ls_OpenDeviceBySerial connect a USB device and starts reading thread (see ls_OpenDeviceByIndex). The argument \textit{pcserialnum} is the serial number (e.g. 1400000) of a device. If the function fails, the return value (\textit{iErr}) is non zero.

1.1.7 ls_CloseDevice
function ls_closedevice : integer;
"ls_CloseDevice" disconnects the current opened device. If the function fails, the return value (\textit{iErr}) 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 \textit{index}.

1.1.11 ls_GetVendorName
function ls_getvendorname(index : Integer) : pchar;
ls_GetVendorName returns the Vendor’s name of the device with index \textit{index}.

1.1.12 ls_GetProductName
function ls_getproductname(index : Integer) : pchar;
ls_GetProductName returns the Product’s name of the device with index \textit{index}.

1.1.13 ls_GetSerialNumber
function ls_getserialnumber(index : Integer) : pchar;
ls_GetSerialNumber the serial number of the device with index \textit{index}.

1.2 Data Functions
1.2.1 ls_WaitForPipe
procedure ls_waitforpipe(timeout : 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. \textit{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_GetPipe
function ls_getpipe(lpBuffer : Pointer; nNumberOfBytesToRead: integer): integer;
ls_GetPipe reads data from the pipe (ring buffer). The argument \textit{lpBuffer} points to the buffer, which has to include the data. \textit{nNumberOfBytesToRead} specifies the length of the data which must be read. The function returns the actual number of bytes read. If \textit{nNumberOfBytesToRead} is specified with 0, then the function returns the actual number of bytes available without reading data.
1.2.3 ls_ResetFiFo

function ls_resetfifo(timeout : dword) : integer;
ls_ResetFiFo resets the hardware FIFO without reading it. *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 Camera Functions

1.3.1 ls_GetSensorType

function ls_getsensortype(var wsensortype, wpixelcount : word; timeout : dword) : integer;
ls_GetSensorType reads the type of the sensor and the sensor’s number of pixels from the sensor circuit board. See also "ls_GetMCU2SensorType". *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_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 : byte; timeout : dword) : integer;
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

function ls_setstate(ucstate : byte; timeout : dword) : integer;
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

function ls_setinttime(dwinttime : dword; timeout : dword) : integer;
ls_SetIntTime sets the integration/exposure time *dwinttime* 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_GetMCU2Version

function ls_getmcu2version(var wversion : word; timeout : dword) : integer;
ls_GetMCU2Version reads the version "wversion"of the line sensor controller.

1.3.7 ls_GetMCU2SensorType

function ls_getmcu2sensortype(var wsensortype : word; timeout : dword) : integer;
ls_GetMCU2SensorType reads the type of the sensor supported by the main circuit board. See also "ls_GetSensorType". *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_GetMode
function ls_getmode(var ucmode : byte; timeout : dword) : integer;
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.

1.3.9 ls_GetState
function ls_getstate(var ucstate : byte; timeout : dword) : integer;
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.

1.3.10 ls_GetIntTime
function ls_getdetectorinttime(var dwinttime : dword; timeout : dword) : integer;
ls_GetIntTime returns the integration/exposure time “dwinttime” 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.

1.3.11 ls_GetPacketLength
function ls_getpacketlength(var packetlength : integer; timeout : dword) : integer;
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.
1.3.12 ls_SetADCPGA

function ls_setadcpga(wRPGA, wGPGA, wBPGA : word; timeout : dword) : integer;

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×).

<table>
<thead>
<tr>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>Gain (V/V)</th>
<th>Gain (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.013</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.56</td>
<td>14.9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6.0</td>
<td>15.56</td>
</tr>
</tbody>
</table>

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

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

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.

1.3.13 ls_SetADC3xPGA

function ls_setadc3xpga(wPGA : word; timeout : dword) : integer;

ls_SetADC3xPGA sets all PGA Gain registers with same value. 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.

1.3.14 ls_SetADCPGA1

function ls_setadcpga1(wPGA : word; timeout : dword) : integer;

ls_SetADCPGA1 sets 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.

1.3.15 ls_SetADCPGA2

function ls_setadcpga2(wPGA : word; timeout : dword) : integer;

ls_SetADCPGA2 sets PGA Gain register of second 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.

1.3.16 ls_SetADCPGA3

function ls_setadcpga3(wPGA : word; timeout : dword) : integer;

ls_SetADCPGA3 sets PGA Gain register of third 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.
1.3.17 ls_GetADCPGA1
function Is_getadcpga1(var wPGA : word; timeout : dword) : integer;
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
function ls_getadcpga2(var wPGA : word; timeout : dword) : integer;
ls_GetADCPGA2 reads the value of the PGA Gain register of second 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.

1.3.19 ls_GetADCPGA3
function ls_getadcpga3(var wPGA : word; timeout : dword) : integer;
ls_GetADCPGA3 reads the value of the PGA Gain register of third 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.

1.3.20 ls_SetADCOffset
function ls_setadcoffset(wROffSet, wGOffSet, wBOffSet : word;
timeout : dword) : integer;
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. 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.

<table>
<thead>
<tr>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>Offset (mV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+300</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-300</td>
</tr>
</tbody>
</table>

1.3.21 ls_SetADC3xOffset
function ls_setadc3xoffset(wOffSet : word; timeout : dword) : integer;
ls_SpSetADC3xOffset sets all Offset registers with same value. For further information please
refer to section 1.3.20. 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_SetADCOFFSet1
function ls_setadcoffset1(wOffSet : word; timeout : dword) : integer;
ls_SpSetADCOFFSet1 sets the Offset register of first channel. For further information please
refer to section 1.3.20. 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_SetADCOffSet2
function ls_setadcoffset2(wOffSet : word; timeout : dword) : integer;
ls_SetADCOffSet2 sets the Offset register of second channel. For further information please refer to section 1.3.20. 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_SetADCOffSet3
function ls_setadcoffset3(wOffSet : word; timeout : dword) : integer;
ls_SetADCOffSet3 sets the Offset register of third channel. For further information please refer to section 1.3.20. 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_GetADCOffSet1
function ls_getadcoffset1(var wOffSet : word; timeout : dword) : integer;
ls_GetADCOffSet1 reads the value of the Offset register of first channel. For further information please refer to section 1.3.20. 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_GetADCOffSet2
function ls_getadcoffset2(var wOffSet : word; timeout : dword) : integer;
ls_GetADCOffSet2 reads the value of the Offset register of second channel. For further information please refer to section 1.3.20. 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.

1.3.27 ls_GetADCOffSet3
function ls_getadcoffset3(var wOffSet : word; timeout : dword) : integer;
ls_GetADCOffSet3 reads the value of the Offset register of third channel. For further information please refer to section 1.3.20. 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.

1.3.28 ls_SetADCCConfig
function ls_setadccconfig(wConfig : word; timeout : dword) : integer;

1.3.29 ls_GetADCCConfig
function ls_getadccconfig(var wConfig : word; timeout : dword) : integer;

1.3.30 ls_SaveSettings
function ls_savesettings(timeout : dword) : integer;
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.
1.4 I2C-Bus Functions

1.4.1 ls_WriteI2C
function ls_writei2c(i2c_addr : byte; pbuf : pointer; wLength : word;
var ucstate : byte; timeout : dword) : integer;
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. 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.

1.4.2 ls_ReadI2C
function ls_readi2c(i2c_addr : byte; pbuf : pointer; var wlength : word;
var ucstate : byte; timeout : dword) : integer;
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. 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.

1.4.3 ls_SetI2CFreq
function ls_seti2cfreq(freq : byte; timeout : dword) : integer;
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. 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.

1.4.4 ls_GetI2CFreq
function ls_geti2cfreq(var freq : byte; timeout : dword) : integer;
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. 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.

1.4.5 ls_GetI2CStat
function ls_geti2cstat(var stat : byte; timeout : dword) : integer;
ls_GetI2CStat reads the state/result of the last I2C read/write operation. Use ls_GetI2CString to obtain the status string. 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.

1.4.6 ls_GetI2CString
function ls_geti2cstring(stat : byte) : pchar;
To obtain a status string of the last I2C read/write operation, use the function cx_GetI2CString. ucStat is the value returned when calling the functions ls_Writei2C and ls_Readi2C. The return value is the status string.