

Interface Manual USBIICMS.DLL V1.2.3

(Rev1.03)

COPTONIX



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

Dynamic Link Library DLL	3
1 Functions	3
1.1 InitUSBIIC	3
1.2 EnumDevices	3
1.3 OpenDeviceByIndex	3
1.4 OpenDeviceBySerial	3
1.5 CloseDevice	4
1.6 CurrentDeviceIndex	4
1.7 GetProductVersion	4
1.8 GetVendorName.....	4
1.9 GetProductName	4
1.10 GetSerialNumber.....	4
1.11 SetMode	4
1.12 WriteI2C.....	5
1.13 ReadI2C.....	5
1.14 WRDI2C.....	5
1.15 ChkSlvAddr.....	5
1.16 ScanI2C	6
1.17 SetSCLHiLo	6
1.18 GetSCLHiLo	6
1.19 SetSCL	6
1.20 GetSCL.....	6
1.21 ResetIRQ	7
1.22 GetI2CStatusString.....	7
1.23 SetI2CSlvAddr	7
1.24 WriteI2CSlaveBuf.....	7
1.25 GetErrorString.....	7
1.26 MemWR8Bit	8
1.27 MemWR16Bit	8
1.28 MemWR32Bit	8
1.29 MemWRBlock	8
1.30 MemRD8Bit	8
1.31 MemRD16Bit	8
1.32 MemRD32Bit	8
1.33 MemRDBlock	9
1.34 SaveSettings	9
1.35 ResetSettings	9
3. Further definitions	10
3.1 Error codes	10

Dynamic Link Library DLL

1 Functions

1.1 InitUSBIIC

function initusbiic (pIRQCallback, pSlvCallback: Pointer; pcMsgID : PChar) : Cardinal; Stdcall;

On starting the host application this function must be called if callback functions are needed. The arguments *pIRQCallback* and *pSlvCallback* are the addresses of callback functions.

TIRQCallback = Procedure; stdcall;

This function is called when an interrupt has been detected.

TSlvCallback = Procedure(const data : Pointer, wSize : Word); stdcall;

This function is called when the converter is in SLAVE_MODE and it receives data from a master.

data is a pointer to the data were received, and *wSize* is the number of bytes were received.

If callback functions are not needed, then just pass *nil* to the function.

e.g. iniusbiic(*nil, nil*, 'My_IIC_Msg_ID');

This function defines also a new window message that is guaranteed to be unique throughout the system. The argument *pcMsgID* (as PChar e.g. "My_USBI2C_App" should be unique) . If more than one application use the same *pcMsgID*, so they will share the same window message ID. If the function successes, the return value is a message identifier in the range 0xC000 through 0xFFFF. If the function fails, the return value is zero. The return 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.

1.2 EnumDevices

function enumdevices : integer; stdcall;

This function enumerates and creates a list of all connected devices and returns the number of devices were found.

1.3 OpenDeviceByIndex

function opendevicebyindex(Index : Integer) : DWORD; stdcall;

If more than one USB I2C converters are connected, use the function "OpenDeviceByIndex" to select one of them for use. This function is 0-based. That means, the first device was connected has the index 0, the second one has the index 1, and so on. If the function fails, the return value is non zero.

1.4 OpenDeviceBySerial

function opendevicebyserial(pcserialnum : PCHAR) : DWORD; stdcall;

This function selects a devices for use (as OpenDeviceByIndex). The argument *pcserialnum* is the serial number (e.g. 251/2009/01/02) of a device . If the function fails, the return value is non zero.

1.5 CloseDevice

function closedevice : DWORD; stdcall;

“CloseDevice” closes the currently opened device. If the function fails, the return value is non zero.

1.6 CurrentDeviceIndex

function currentdeviceindex : integer; stdcall;

This function returns the index (0-based) of the device currently selected. If the device is closed the return value is -1.

1.7 GetProductVersion

function getproductversion(index : integer) : PCHAR; stdcall;

This function returns the version of the converter with index “*index*”.
PCHAR is a NULL terminated string.

1.8 GetVendorName

function getvendorname(index : integer) : PCHAR; stdcall;

This function returns the Vendor’s name of the converter with index “*index*”.

1.9 GetProductName

function getproductname(index : integer) : PCHAR; stdcall;

This function returns the Product’s name of the converter with index “*index*”.

1.10 GetSerialNumber

function getserialnumber(index : integer) : PCHAR; stdcall;

This function returns the serial number of the converter with index “*index*”.

1.11 SetMode

**function setmode(var ucMode, ucSlvAddr : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

Selects one of two modes: MASTER_MODE or SLAVE_MODE.

ucMode = 0 -> will not effect any changes. The function would only return current mode.

ucMode = 1 -> MASTER_MODE

ucMode = 2 -> SLAVE_MODE

ucSlvAddr is the own slave address of the converter (valid only in SLAVE_MODE).

If the LSB is 1, then the General Call is active.

dwTimeOut is time out value in millisecond.

1.12 WriteI2C

**function writei2c(ucSlvAddr : Byte; const pi2cdata; wSize : WORD;
var wStatus : WORD; dwTimeOut : DWORD) : DWORD; stdcall;**

writes data to a I2C device. *ucSlvAddr* is the slave address of a I2C device. *pi2cdata* is a pointer to the buffer containing the data to be written to a device. This buffer must remain valid for the duration of the write operation.

wSize is the number of bytes to be written to the device. *wStatus* returns the state of the I2C bus. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.13 ReadI2C

**function readi2c(ucSlvAddr : Byte; var pi2cdata; var wSize, wStatus : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads data from a I2C device. *ucSlvAddr* is the slave address of a I2C device. *pi2cdata* is a pointer that receives the data read from a device. This buffer must remain valid for the duration of the write operation.

wSize is the number of bytes to be read from the device. *wSize* returns also the number of bytes read from the device. *wStatus* returns the state of the I2C bus. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.14 WRDI2C

**function wrdi2c(ucSlvAddr : Byte; const pwrdata; wwrSize : WORD;
var prddata; var wrdSize, wStatus : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes and reads data from a I2C device. *ucSlvAddr* is the slave address of a I2C device. *pwrdata* is a pointer to the buffer containing the data to be written to a device. This buffer must remain valid for the duration of the write operation. *wwrSize* is the number of bytes to be written to the device. *prddata* is a pointer that receives the data read from a device. This buffer must remain valid for the duration of the write operation. *wrdSize* is the number of bytes to be read from the device. *wrdSize* returns also the number bytes read from the device. *wStatus* returns the state of the I2C bus. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.15 ChkSlvAddr

**function chkslvaddr(ucslvaddr : Byte; var ucConnected : byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

checks if the slave address *ucSlvAddr* is connected to the I2C bus. If *ucConnected* returns 0, then the device is not connected. If *ucConnected* returns 1, then the device is connected to the I2C bus. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.16 ScanI2C

**function scani2c(var pscandata; var ucSize : byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

scans all I2C device currently connected to the bus. *pscandata* is a pointer that receives the list of slave addresses (devices) were detected. This buffer must remain valid for the duration of the scan operation. *ucSize* returns the number of devices were detected. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.17 SetSCLHiLo

function setsclhilo(wSCLH, wSCLL: WORD; dwTimeOut : DWORD) : DWORD; stdcall;

Use this function to set the I2C clock frequency and the duty cycle. *wSCLH* determines the high time and *wSCLL* the low time of the I2C clock. *wSCLH* and *wSCLL* together determine the clock frequency generated by the master. The clock frequency is determined by the following formula:

$$I2C_{frequency} = 60000000 / (wSCLH + wSCLL)$$

The values for *wSCLH* and *wSCLL* should not necessarily be the same. Software can set different duty cycles on SCL by setting different values *wSCLH* and *wSCLL*. The values must ensure the data rate is in the data rate range of 500 Hz through 1MHz.

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

1.18 GetSCLHiLo

**function getsclhilo(var wSCLH, wSCLL: WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads the current settings for I2C SCL clock frequency (see *SetSCLHiLo* for further information). *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.19 SetSCL

function setscl(dwfreq : DWORD; dwTimeOut : DWORD) : DWORD; stdcall;

sets a new value for I2C clock frequency. *dwfreq* is the new frequency (in [Hz]) in the range of 500Hz through 1MHz. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.20 GetSCL

function getscl(var dwfreq : DWORD; dwTimeOut : DWORD) : DWORD; stdcall;

reads the value for I2C clock frequency. *dwfreq* is the frequency (in [Hz]) in the range of 500Hz through 1MHz. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.21 ResetIRQ

function resetirq(var ucIRQMode : Byte; dwTimeOut : DWORD) : DWORD; stdcall;

The converter has an interrupt input. e.g. if you use an IO-Expander, you could connect its interrupt output with the interrupt input of the converter. Then your software would recognize all events of the IO-Expander if the state of the IOs has been changed.

Possible values for **ucIRQMode**:

0: disable interrupt.

1: enable interrupt. Interrupt is falling-edge sensitive.

2: enable interrupt. Interrupt is rising-edge sensitive.

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

1.22 GetI2CStatusString

function geti2cstatusstring(wStatus : WORD) : PCHAR; stdcall;

To obtain a status string of the last I2C read/write operation, use the function *GetI2CStatusString*. **wStatus** is the value returned when calling the functions *WriteI2C*, *ReadI2C* and *WRDI2C*. The return value is the status string.

1.23 SetI2CSlvAddr

function seti2cslvaddr(var ucslvaddr : Byte; dwTimeOut : DWORD) : DWORD; stdcall;

If the converter operates in SLAVE_MODE, so it is possible to change the slave address any time using this function. **ucslvaddr** is the new slave address of the converter. **dwTimeOut** is time out value in millisecond. If the function fails the return value is non zero.

1.24 WriteI2CSlaveBuf

**function writei2cslvbuf(const pi2cdata; wSize : WORD; ucEvent: Byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes data to the I2C slave output buffer. **pi2cdata** is a pointer to the buffer containing the data to be written to the buffer. This buffer must remain valid for the duration of the write operation. **wSize** is the number of bytes to be written to the buffer. If **ucEvent** is set to 1, the converter sends an interrupt signal to the master. Thus the converter informs the master that there is data ready to read. **dwTimeOut** is time out value in millisecond. If the function fails the return value is non zero.

1.25 GetErrorString

function geterrorstring(dwErr : DWORD) : PCHAR; stdcall;

To obtain an error string for error codes, use the function *GetErrorString*. For a complete list of error codes provided by the DLL, see section 3.1 *Error codes*. **dwErr** is the return value of all functions mentioned in this manual. The return value is the error string.

1.26 MemWR8Bit

**function memwr8bit(wAddr : WORD; ucVal : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes *ucVal* (1 BYTE / 8Bit value) into the onboard memory at address *wAddr*. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.27 MemWR16Bit

**function memwr16bit(wAddr : WORD; wVal : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes *wVal* (1 WORD / 16Bit value) into the onboard memory at address *wAddr*. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.28 MemWR32Bit

**function memwr32bit(wAddr : WORD; dwVal : DWORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes *dwVal* (1 DWORD / 32Bit value) into the onboard memory at address *wAddr*. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.29 MemWRBlock

**function memwrblock(wAddr : WORD; const pmemdata; wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

writes data into the onboard memory. *pmemdata* is a pointer to the buffer containing the data to be written to the memory. This buffer must remain valid for the duration of the write operation. *wSize* is the number of bytes to be written to the memory. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.30 MemRD8Bit

**function memrd8bit(wAddr : WORD; var ucVal : Byte;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads 1 BYTE (8Bit) from memory at address *wAddr*. *ucVal* returns the BYTE read. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.31 MemRD16Bit

**function memrd16bit(wAddr : WORD; var wVal : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads 1 WORD (16Bit) from memory at address *wAddr*. *wVal* returns the WORD read. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.32 MemRD32Bit

**function memrd32bit(wAddr : WORD; var dwVal : DWORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads 1 DWORD (32Bit) from memory at address *wAddr*. *dwVal* returns the DWORD read. *dwTimeOut* is time out value in millisecond. If the function fails the return value is non zero.

1.33 MemRDBlock

**function memrdblock(wAddr : WORD; var pmemdata; var wSize : WORD;
dwTimeOut : DWORD) : DWORD; stdcall;**

reads number of bytes (**wSize**) of data from memory starting at address **wAddr**. **pmemdata** is a pointer that receives the data read from a memory. This buffer must remain valid for the duration of the write operation. **wSize** is the number of bytes to be read from the memory. **wSize** returns also the number of bytes read from the memory. **dwTimeOut** is time out value in millisecond. If the function fails the return value is non zero.

1.34 SaveSettings

function savesettings(dwTimeOut : DWORD) : DWORD; stdcall;

By calling the function “savesettings”, the settings (operation mode, slave address and SCL frequency) are stored into onboard memory. **dwTimeOut** is time out value in millisecond. If the function fails the return value is non zero.

1.35 ResetSettings

function resetsettings(dwTimeOut : DWORD) : DWORD; stdcall;

loads factory (default) settings:

MASTER_MODE

SLAVE ADDRESS = 0x00

SCL FREQUENCY = 100 kHz. (wSCLL = 0x012C and wSCLH = 0x012C)

dwTimeOut is time out value in millisecond. If the function fails the return value is non zero.

3. Further definitions

3.1 Error codes

\$00000000	RET_OK
\$00010001	ERR_NO_DEVICES_ATTACHED
\$00010002	ERR_DEVICE_CLOSED
\$00010003	ERR_INVALID_DEVICE_INDEX
\$00010004	ERR_DEVICE_INDEX_OUT_OF_RANGE
\$00010005	ERR_SERIAL_NUM_NOT_FOUND
\$00010006	ERR_DEVICE_NOT_CONNECTED
\$00010007	ERR_CREATE_WR_EVENT
\$00010008	ERR_CREATE_WR_HANDLE
\$00010009	ERR_CREATE_SYNC_EVENT
\$0001000A	ERR_CREATE_RD_HANDLE
\$0001000B	ERR_ALLOCATE_THREADPARAM_MEM
\$0001000C	ERR_CREATE_RDTHREAD_HANDLE
\$0001000D	ERR_CREATE_RDTHREAD_REX_EVENT
\$00020001	ERR_INVALID_WR_HANDLE
\$00020002	ERR_INVALID_RD_HANDLE
\$00020003	ERR_INVALID_RD_THREAD_HANDLE
\$00020004	ERR_INVALID_TH_TO_IO_EVENT_HANDLE
\$00020005	ERR_INVALID_IO_TO_TH_EVENT_HANDLE
\$00020006	ERR_INVALID_RD_TH_PARAM_MEM
\$00030001	ERR_USB_WRITE
\$00030002	ERR_USB_WRITE_TIMEOUT
\$00030003	ERR_USB_WRITE_RESULT
\$00030004	ERR_USB_READ_TIMEOUT
\$00030005	ERR_USB_READ_WAIT_FAILED
\$00030006	ERR_USB_READ_WAIT_ABANDONED
\$00030007	ERR_USB_IO_CALLBACK
\$00040001	ERR_IO_TO_TH_SETEVENT
\$00040002	ERR_TH_TO_IO_SETEVENT
\$00040003	ERR_TH_TO_IO_RESETEVENT
\$00050001	ERR_RD_TH_NOT_ACTIVE
\$00050002	ERR_RD_TH_GET_EXIT_CODE
\$00050003	ERR_RD_TH_READ_RESULT
\$00050004	ERR_RD_TH_READFILE
\$00050005	ERR_RD_TH_TIMEOUT
\$00050006	ERR_RD_TH_WAIT_FAILED
\$00050007	ERR_RD_TH_WAIT_ABANDONED
\$00060001	ERR_CREATE_MMF_HANDLE
\$00060002	ERR_CREATE_MMF
\$00060003	ERR_CREATE_MMF_LOCK_HANDLE
\$00060004	ERR_LOCK_MMF
\$00060005	ERR_MMF_MAX_APP_NUM_REACHED

\$00070001	ERR_WRI2C_INVALID_SIZE
\$00070002	ERR_WRI2C_FAILED
\$00070003	ERR_WRI2C_FAILED_UNKNOWN
\$00070004	ERR_RDI2C_INVALID_SIZE
\$00070005	ERR_RDI2C_FAILED
\$00070006	ERR_RDI2C_FAILED_UNKNOWN
\$00070007	ERR_SCNI2C_FAILED_UNKNOWN
\$00070008	ERR_SETSCL_FAILED_UNKNOWN
\$00070009	ERR_GETSCL_FAILED_UNKNOWN
\$0007000A	ERR_INVALID_SCL_FREQUENCY
\$0007000B	ERR_SET_IRQ_FAILED_UNKNOWN
\$0007000C	ERR_WRDI2C_FAILED
\$0007000D	ERR_WRDI2C_FAILED_UNKNOWN
\$0007000E	ERR_I2C_MODE
\$0007000F	ERR_I2C_UNKNOWN_CMD
\$00070010	ERR_WRI2C_SLAVE_BUF
\$00070011	ERR_SET_SLAVE_ADDR
\$00070012	ERR_GET_SLAVE_ADDR
\$00090003	ERR_MEMWRITE_FAILED
\$00090004	ERR_MEMWRITE_FAILED_UNKNOWN
\$00090005	ERR_MEMREAD_FAILED
\$00090006	ERR_MEMREAD_FAILED_UNKNOWN
\$00090007	ERR_WRMEM_INVALID_SIZE
\$00090008	ERR_RDMEM_INVALID_SIZE
\$00090009	ERR_SAVE_SETTINGS_FAILED
\$0009000A	ERR_SAVE_SETTINGS_FAILED_UNKNOWN
\$0009000B	ERR_RESET_SETTINGS_FAILED
\$0009000C	ERR_RESET_SETTINGS_FAILED_UNKNOWN