

Interface Manual wUSBPipe.DLL

(V1.1)

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

Dynamic Link Library DLL	3
1 Functions	3
1.1 cx_Initialize	3
1.2 cx_EnumDevices	3
1.3 cx_OpenDeviceByIndex	3
1.4 cx_OpenDeviceBySerial	3
1.5 cx_CloseDevice	3
1.6 cx_DeviceCount	4
1.7 cx_CurrentDeviceIndex	4
1.8 cx_WaitForPipe	4
1.9 cx_GetPipe	4
1.10 cx_GetVersion	4
1.11 cx_GetVendorName	4
1.12 cx_GetProductName	4
1.13 cx_GetSerialNumber	4
1.15 cx_GetErrorString	5
1.18 cx_ResetFiFo	5
1.19 cx_SetConfig	5
1.20 cx_GetConfig	6
1.21 cx_WriteI2C	6
1.22 cx_ReadI2C	6
1.23 cx_SetI2CFreq	6
1.24 cx_GetI2CFreq	6
1.25 cx_GetI2CState	7
1.26 cx_GetI2CString	7
1.28 cx_WriteUART	7
1.27 cx_ReadUART	7
1.29 cx_SetBaudrate	8
1.30 cx_GetBaudrate	8

Dynamic Link Library DLL

1 Functions

1.1 cx_Initialize

**function cx_initialize(dwPipeSize, dwReadLength, dwThreadClass : DWORD;
iThreadPrio : Integer; pcMsgID : PChar): DWORD; stdcall;**

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* = 512 means 512 bytes buffer and 67108864 = 64 Mbytes. The default value is 32MB. 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`.

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_USBCCD_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 cx_EnumDevices

function cx_enumdevices : Integer; stdcall;

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

1.3 cx_OpenDeviceByIndex

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

If more than one USB I2C converters are connected, use the function "cx_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 cx_OpenDeviceBySerial

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

This function selects a devices for use (as cx_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 cx_CloseDevice

function cx_closedevice : DWORD; stdcall;

"cx_CloseDevice" closes the currently opened device. If the function fails, the return value is non zero.

The return value is equal to zero (0) on success and non-zero on failure. Use "cx_geterrorstring" to get more information about the error.

1.6 cx_DeviceCount

function cx_devicecount : Byte; stdcall;

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

1.7 cx_CurrentDeviceIndex

function cx_currentdeviceindex : ShortInt; stdcall;

This function returns the index of the opened USB device. The return value is -1 if no USB device is opened.

1.8 cx_WaitForPipe

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

The `cx_WaitForPipe` function 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 is non zero.

1.9 cx_GetPipe

**function cx_getpipe(pPipeBuf : Pointer; dwToRead : DWORD;
Var dwRead : DWORD) : DWORD; stdcall;**

`Cx_GetPipe` reads data from the pipe (ring buffer). The argument `pPipeBuf` 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. The function's return value is zero, if reading data was successful.

1.10 cx_GetVersion

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

This function returns the version of the converter with index "*index*".

PCHAR is a NULL terminated string.

1.11 cx_GetVendorName

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

This function returns the Vendor's name of the converter with index "*index*".

1.12 cx_GetProductName

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

This function returns the Product's name of the converter with index "*index*".

1.13 cx_GetSerialNumber

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

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

1.15 cx_GetErrorString

function cx_geterrorstring(err : DWORD) : PChar; stdcall;

This function converts the error code **err** to a zero terminated string.

err is the return value of other functions like *OpenDriver*, *CloseDriver* and others.

The function returns the error message as a zero terminated string.

1.18 cx_ResetFiFo

function cx_resetfifo : DWORD; stdcall;

The device contains a 2kByte FiFo. The ResetFiFo function resets the FiFo without reading it.

The return value is equal to zero (0) on success and non-zero on failure. Use "cx_geterrorstring" to get more information about the error.

1.19 cx_SetConfig

function cx_setconfig(ucCfg : Byte) : DWORD; stdcall;

SetConfig configures the data and control interfaces of the device.

The argument *ucCfg* is a bit-wise "OR" combination of the following:

IFCLKSRC	= 0x80	This bit selects the clock source for the FIFO. IFCLKSRC = 0, the external clock on the IFCLK pin is selected. IFCLKSRC = 1, an internal 30 or 48 MHz clock is used.
3048MHZ	= 0x40	Internal FIFO Clock Frequency. 3040MHZ = 0, then 30 MHz is used 3048MHZ = 1, then 40 MHz is used
IFCLKOE	= 0x20	IFCLK pin output enable IFCLKOE = 0, Tri-state IFCLKOE = 1, Output enabled
IFCLKPOL	= 0x10	Invert the IFCLK signal IFCLKPOL = 0, the clock is not inverted IFCLKPOL = 1, the clock is inverted
ASYNC	= 0x08	Slave FIFO Asynchronous Mode. When ASYNC = 0, the Slave FIFO operate synchronously. A Clock is supplied either internally or externally on the IFCLK pin; The FIFO control signals function as read and write enable signals for the clock signal. When ASNC = 1, the Slave FIFO operate asynchronously. No Clock signal input to IFCLK is required; the FIFO control signals function directly as read and write strobes.
WORDWIDE	= 0x01	Select Byte/Word FIFO on data port. WORDWIDE = 0, 8 bit interface [8:0] is used WORDWIDE = 1, 16 bit interface [15:0] is used

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.20 cx_GetConfig

function cx_getconfig(Var ucCfg : Byte) : DWORD; stdcall;

The function GetConfig reads the configuration of the device. The Argument *ucCfg* returns the actual configuration of the device. For further information please see the function SetConfig 1.19.

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.21 cx_WriteI2C

**function cx_writei2c(ucSlvAdr : Byte; pBuf : Pointer;
wLength : WORD; Var ucStatus : Byte) : DWORD; stdcall;**

Use this function to address and write data to I2C slave devices.

This function takes four arguments:

ucSlvAdr 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.

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.22 cx_ReadI2C

**function cx_readi2c(ucSlvAdr : Byte; pBuf : Pointer;
Var wLength : Word; Var ucStatus : Byte) : DWORD; stdcall;**

Use this function to address and read data from I2C slave devices.

This function takes four arguments:

ucSlvAdr 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.

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.23 cx_SetI2CFreq

function cx_seti2cfreq(ucfreq : Byte) : DWORD; stdcall;

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

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.24 cx_GetI2CFreq

function cx_geti2cfreq(var ucfreq : Byte) : DWORD; stdcall;

GetI2CFreq reads the actual I2C Clock frequency. *ucfreq* returns the actual I2C Clock frequency. For further information please see the function SetI2CFreq 1.23.

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.25 cx_GetI2CState

function cx_geti2cstate(var ucStat : Byte) : DWORD; stdcall;

reads the state/result of the last I2C read/write operation. Please use the function GetI2CString to obtain the status string. The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.26 cx_GetI2CString

function cx_geti2cstring(var ucStat : Byte) : DWORD; stdcall;

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 WriteI2C and ReadI2C. The return value is the status string.

1.28 cx_WriteUART

function cx_writeuart(pBuf : Pointer; Var wLength : WORD) :DWORD; stdcall;

The WriteUART function writes data to serial interface (TTL) to an external device. *pBuf* is the pointer, that points to the data structure. *wLength* is the number of bytes to write.

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.27 cx_ReadUART

function cx_readuart(Reg : Byte; pBuf : Pointer; Var wLength : WORD; ucTimeOut : Byte) :DWORD; stdcall;

ReadUART reads data from serial interface from an external device. If the argument *Reg* is unequal zero (0), then the value of *Reg* is written first to the serial interface. After *Reg* is written, then reading data starts. *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. The device waits for *ucTimeOut* (time-out interval) to read the requested number of bytes from external device. The time-out value will be expected in 100 ms units. A value of 10 corresponds to 1 sec. The max. 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
```

The return value is equal to zero (0) on success and non-zero on failure. Use “cx_geterrorstring” to get more information about the error.

1.29 cx_SetBaudrate

function cx_setbaudrate(ucBaud : Byte): DWORD; stdcall;

This function sets the Baud rate of the UART / serial interface of the device. The argument *ucBaud* is the Baud rate to be set. The Baud rate is programmable from 2400 to 57600.

Possible Baud rates:

<i>ucBaud</i> =	0	: 2400
	1	: 4800
	2	: 9600
	3	: 19200
	4	: 28800
	5	: 38400
	6	: 57600

The return value is equal to zero (0) on success and non-zero on failure. Use "cx_geterrorstring" to get more information about the error.

1.30 cx_GetBaudrate

function cx_getbaudrate(Var ucBaud : Byte): DWORD; stdcall;

Use GetBaudrate to read the actual Baud rate of the serial interface. *ucBaud* returns the Baud rate. For further information please see the function SetBaudrate 1.29.

The return value is equal to zero (0) on success and non-zero on failure. Use "cx_geterrorstring" to get more information about the error.