Application Interface
for Windows®

Version 5.0.3
05.2019

Cleware GmbH
Nedderend 3
24876 Hollingstedt
www.cleware.de

Contents

1. The application interface in general 2
2. USBaccess.h 2
3. API Functions 6
4. API C++ Example 12
5. API C Example 14
6. API-Beispiel C# 16

1. The application interface in general

The devices build by Cleware GmbH may be controlled by programs build by the user or by products from third party vendors. For this purpose three files are supplied for interfacing on Windows® systems. The Linux API is described in the separate Linux documentation. The API files are located in the installation directory of Cleware.

USBaccess.h	Definition of the API
UBSaccess.lib	Link information
USBaccess.dll	Library

ActiveX note: This interface will not be supperted any longer. If these interfaces were needed the previous version 3.6 or earlier must be used.

Starting with version 4.5.0 access to the library functions support the use of serial numbers as device number.

2. USBaccess.h

The file USBaccess.h contains the interface to the Cleware USB devices. After including this header file the devices could be opened and accessed.

```
class USBACCESS_API CUSBaccess {
   public:
      enum USBactions {LEDs=0, EEwrite=1, EEread=2, Reset=3, KeepCalm=4, GetInfo=5,
               StartMeasuring=6,     // USB-Humidity
               Configure=7,     // USB-IO16-V10, USB-Counter-V05
               Display=8,       // USB/Display
               RunPoint=10,     // USB-Encoder
               ContactWrite=11,   // 613
               ContactRead=12,    // 613
               EEread4=13,      // 613
               Programm=15      // Transfer new Firmware (internal use only)
   } ;

   enum USBInfoType{ OnlineTime=1, OnlineCount=2, ManualTime=3, ManualCount=4 } ;
   enum LED_IDs     { LED_0=0, LED_1=1, LED_2=2, LED_3=3 } ;
   enum ADCTypes    { ADC_HX71=1, ADC_PT100=2, ADC_PT1000=3, ADC_121=5  } ;
   enum COUNTER_IDs{ COUNTER_0=0, COUNTER_1=1 } ;
   enum SWITCH_IDs { SWITCH_0=0x10, SWITCH_1=0x11, SWITCH_2=0x12, SWITCH_3=0x13,
             SWITCH_4=0x14, SWITCH_5=0x15, SWITCH_6=0x16, SWITCH_7=0x17,
             SWITCH_8=0x18, SWITCH_9=0x19, SWITCH_10=0x1a, SWITCH_11=0x1b,
                 SWITCH_12=0x1c, SWITCH_13=0x1d, SWITCH_14=0x1e, SWITCH_15=0x1f
   } ;


   enum USBtype_enum {ILLEGAL_DEVICE=0,
         LED_DEVICE=0x01,
         POWER_DEVICE=0x02,
         DISPLAY_DEVICE=0x03,
         WATCHDOG_DEVICE=0x05,
         AUTORESET_DEVICE=0x06,
         WATCHDOGXP_DEVICE=0x07,
         SWITCH1_DEVICE=0x08,SWITCH2_DEVICE=0x09, SWITCH3_DEVICE=0x0a,
```

```
        SWITCH4_DEVICE=0x0b,SWITCH5_DEVICE=0x0c, SWITCH6_DEVICE=0x0d, SWITCH7_DEVICE=0x0e,
        SWITCH8_DEVICE=0x0f,
        TEMPERATURE_DEVICE=0x10,
        TEMPERATURE2_DEVICE=0x11,
        TEMPERATURE5_DEVICE=0x15,
        HUMIDITY1_DEVICE=0x20, HUMIDITY2_DEVICE=0x21,
        SWITCHX_DEVICE=0x28,     // new switch 3,4,8
        CONTACT00_DEVICE=0x30, CONTACT01_DEVICE=0x31, CONTACT02_DEVICE=0x32, CONTACT03_DEVICE=0x33,
        CONTACT04_DEVICE=0x34, CONTACT05_DEVICE=0x35, CONTACT06_DEVICE=0x36, CONTACT07_DEVICE=0x37,
        CONTACT08_DEVICE=0x38, CONTACT09_DEVICE=0x39, CONTACT10_DEVICE=0x3a, CONTACT11_DEVICE=0x3b,
        CONTACT12_DEVICE=0x3c, CONTACT13_DEVICE=0x3d, CONTACT14_DEVICE=0x3e, CONTACT15_DEVICE=0x3f,
        F4_DEVICE=0x40,
        KEYC01_DEVICE=0x41, KEYC16_DEVICE=0x42, MOUSE_DEVICE=0x43,
        KEYC813_DEVICE=0x48, KEYC816_DEVICE=0x48, MOUSE813_DEVICE=0x4a,
        ADC0800_DEVICE = 0x50, ADC0801_DEVICE = 0x51, ADC0802_DEVICE = 0x52,
        ADC0803_DEVICE = 0x53,ADC0804_DEVICE = 0x54, ADC0805_DEVICE = 0x55, ADC0806_DEVICE = 0x56,
        ADC0807_DEVICE = 0x57, ADC0808_DEVICE = 0x58, ADC0809_DEVICE = 0x59, ADC0810_DEVICE = 0x5a,
        ADC0811_DEVICE = 0x5b,
        ADC0812_DEVICE = 0x5c, ADC0813_DEVICE = 0x5d, ADC0814_DEVICE = 0x5e, ADC0815_DEVICE = 0x5f,
        COUNTER00_DEVICE=0x60, COUNTER01_DEVICE=0x61, COUNTER02_DEVICE=0x62,
        CONTACTTIMER00_DEVICE=0x70, CONTACTTIMER01_DEVICE=0x71, CONTACTTIMER02_DEVICE=0x72,
        ENCODER01_DEVICE=0x80,
        IDETEC_DEVICE=0x81,         // for internal use only
        CLEWLAN_DEVICE=0x88, RBV24_DEVICE=0x89,  // for internal use only
        IR_DEVICE=0x90,          // Infrared Sender/Receiver
        COLOR_DEVICE=0xa0, COLOR_RD_DEVICE = 0xa1, COLOR_GN_DEVICE = 0xa2,
        COLOR_BL_DEVICE = 0xa3,  // Color Sensor, returns white and r g b
        LOGIC_DEVICE = 0x100,    // for internal use only, collects 00-07
        LOGIC00_DEVICE = 0x110, LOGIC01_DEVICE = 0x111, LOGIC02_DEVICE = 0x112,
        LOGIC03_DEVICE = 0x113,  // for internal use only
        LOGIC04_DEVICE = 0x114, LOGIC05_DEVICE = 0x115, LOGIC06_DEVICE = 0x116,
        LOGIC07_DEVICE = 0x117,  // for internal use only
        BUTTON_NODEVICE=0x1000
} ;



   private:
      class CUSBaccessBasic *  X;// avoid export of internal USB variables

   public:
      CUSBaccess(void) ;
      virtual ~CUSBaccess(void) ;  // maybe used as base class
      virtual int   OpenCleware(void) ;
      virtual int CloseCleware(void) ;  // close all Cleware devices
      virtual int   Recover(int devNum) ;
      virtual HANDLEGetHandle(int deviceNo) ;
      virtual int   GetValue(int deviceNo, unsigned char *buf, int bufsize) ;
      virtual int   SetValue(int deviceNo, unsigned char *buf, int bufsize) ;
      virtual int   SetLED(int deviceNo, enum LED_IDs Led, int value) ;
      // value: 0=off 7=medium 15=highlight
      virtual int   SetSwitch(int deviceNo, enum SWITCH_IDs Switch, int On) ;
      // On: 0=off, 1=on
      virtual int   GetSwitch(int deviceNo, enum SWITCH_IDs Switch) ;        // On: 0=off, 1=on, -1=error
      virtual int   GetSeqSwitch(int deviceNo, enum SWITCH_IDs Switch, int seqNum) ;
       //On: 0=off, 1=on,  -1=error
      virtual int GetSwitchConfig(int deviceNo, int *switchCount, int *buttonAvailable) ;
      virtual int GetTemperature(int deviceNo, double *Temperature, int *timeID, int subDevice=0) ;
      virtual float  GetTemperature(int deviceNo) ;
      virtual int GetHumidity(int deviceNo, double *Humidity, int *timeID, int subDevice=0) ;
```

```
    virtual float GetHumidity(int deviceNo) ;
    virtual int   SelectADC(int deviceNo, int subDevice) ;
    virtual float GetADC(int deviceNo, int sequenceNumber, int subDevice) ;
    virtual int   GetADCType(int deviceNo) ;    // returns type of ADC
    virtual int   ResetDevice(int deviceNo) ;
    virtual int   StartDevice(int deviceNo) ;
    virtual int   CalmWatchdog(int deviceNo, int minutes, int minutes2restart) ;
    virtual int   GetVersion(int deviceNo) ;
    virtual int   GetUSBType(int deviceNo) ;
    virtual int   GetSerialNumber(int deviceNo) ;
    virtual int   GetDLLVersion() { return USBaccessVersion ; }
    virtual int   GetManualOnCount(int deviceNo) ;
   // returns how often switch is manually turned on
    virtual int   GetManualOnTime(int deviceNo) ;
   // returns how long (seconds) switch is manually turned on
    virtual int   GetOnlineOnCount(int deviceNo) ;
   // returns how often switch is turned on by USB command
    virtual int   GetOnlineOnTime(int deviceNo) ;
   // returns how long (seconds) switch is turned on by USB command
   virtual int GetMultiSwitch(int deviceNo, unsigned long int *mask, unsigned long int *value,
   int se qNumber) ;
    virtual int   SetMultiSwitch(int deviceNo, unsigned long int value) ;
    virtual int   SetMultiConfig(int deviceNo, unsigned long int directions) ;
     virtual int  GetCounter(int deviceNo, enum COUNTER_IDs counterID) ;// COUNTER_IDs ununsed until now
    virtual int   SetCounter(int deviceNo, int counter, enum COUNTER_IDs counterID) ; //  -1=error, COUN
     TER_IDs ununsed until now
    virtual int   SyncDevice(int deviceNo, unsigned long int mask) ;
    virtual int   SetDisplay(int deviceNo, int byte1, int byte2, int segmentDirect=0);   //  1=ok, 0=error
    virtual int   GetMultiConfig(int deviceNo) ;   // returns directions
   virtual int IsCutter(int deviceNo) ;   // return true if this is a cutter device 1=Cutter, 2=Connect,
     3=Multi2, 4=Multi2X, 17=Cutter3.0, 18=Connect3.0
    virtual int   IsAlarm(int deviceNo) ;       // return true if this is a alarm buzzer device
    virtual int   IsAmpel(int deviceNo) ;       // return true if this is an ampel (traffic light) device
    virtual int   IsLuminus(int deviceNo) ;     // return true if this is a Luminus device
    virtual int   IsSolidState(int deviceNo) ;  // return true if device is low voltage non relais switch
    virtual int   GetHWversion(int deviceNo) ;  // return HWversion (0 for pre 2014 designed devices, 13
    for new devices)
    virtual int   IsWatchdogInvert(int deviceNo) ;      // return true if watchdog inverted
    virtual int   IOX(int deviceNo, int addr, int data) ;     // for internal use only, wrong usage may
    destroy device
    virtual int   IsIdeTec(int deviceNo) ;      // return true if watchdog inverted
    virtual int   IsMonitoredSwitch(int deviceNo) ;  // return true if this is a monitored switch
    virtual int   SetTempOffset(int deviceNo, double Sollwert, double IstWert) ; // returns 1 if ok or 0
    in case of an error
    virtual int   GetFrequency(int deviceNo, unsigned long int *counter, int subDevice) ;
    virtual int   GetColor(int deviceNo, int *count_white, int *count_red, int *count_green, int
    *count_blue) ;

#ifdef CC_USE_DEBUGWRITE
    virtual void  DebugWrite(CString &s) ;// for internal use only
    virtual void  DebugWrite(_TCHAR *s) ;  // for internal use only
#endif  // CC_USE_DEBUGWRITE
} ;

extern "C" {
   // for use of Class CUSBaccess from Delphi
   USBACCESS_API CUSBaccess * _stdcall USBaccessInitObject() ;
   USBACCESS_API void _stdcall USBaccessUnInitObject(CUSBaccess *) ;
   } ;
#else // __cplusplus
```

```
typedef unsigned long int CUSBaccess ;// typedef void * HANDLE ; // defined in windows.h
enum FCWUSBactions {  LEDs=0, EEwrite=1, EEread=2, Reset=3, KeepCalm=4, GetInfo=5, StartMeasuring=6 } ;
enum FCWLED_IDs {  LED_0=0, LED_1=1, LED_2=2, LED_3=3 } ;
enum FCWADCTypes { ADC_HX71=1, ADC_PT100=2, ADC_PT1000=3, ADC_121=5  } ;// ADC_121 is the standard ADC
device
enum FCWSWITCH_IDs {  SWITCH_0=0x10, SWITCH_1=0x11, SWITCH_2=0x12, SWITCH_3=0x13 } ;
enum FCWUSBtype_enum{ILLEGAL_DEVICE=0,
        LED_DEVICE=0x01,
        WATCHDOG_DEVICE=0x05,
        AUTORESET_DEVICE=0x06,
        SWITCH1_DEVICE=0x08,
        TEMPERATURE_DEVICE=0x10,
        TEMPERATURE2_DEVICE=0x11,
        TEMPERATURE5_DEVICE=0x15,
        HUMIDITY1_DEVICE=0x20,
        CONTACT00_DEVICE=0x30
} ;

#endif  // __cplusplus

…
```

The Methods of the class CUSBaccess got the attribute “virtual” to enable the usage with Delphi. To use the class USBaccess with Delphi, the function USBaccessInitObject() must be called first, because the C++ class must be created in the C++ context.

An alternative is the use of simple functions without a class. This allows the usage of the language C, LabView or others. All methods of the class USBaccess are also available as functions. The function names always starts with the letters FCW.

Before using the functionals interface, the access must be prepared by calling the function FCWInitObject(). The returned value is used as the first argument in all other FCW functions. Before leaving the program, the function FCWUnInitObject(obj) will do the cleanup of the the environment. A sample for using the functional interface is shown in chapter 5.

The devices found by OpenCleware are stored internally inside an array. If multiple Cleware devices are used, there is no guarantee which device is located at what array position. It is necessary to check the devices returned by OpenCleware. The array member are accessed by using the index “deviceNo” in the API functions.

Starting with API version 4.5.0 the function argument “deviceNo” may also be the serial number simplifying the use of dedicated devices.

All USB traffic light devices with version at least 104 (September 2014) support flashing of the lights. This feature is controlled by the API function SetSwitch.

3. API Functions

CUSBaccess *FCWInitObject() ;
Initialize the functional interface.

void FCWUnInitObject(CUSBaccess *obj) ;
Closes the functional interface.

int OpenCleware() ;
int FCWOpenCleware(CUSBaccess *obj) ;
Looks for Cleware USB devices and opens them. The number of found devices is returned.

int CloseCleware() ;
int FCWCloseCleware(CUSBaccess *obj) ;
Closes all open Cleware USB devices.

int Recover (int deviceNo) ;
int FCWRecover (CUSBaccess *obj, int deviceNo) ;
When reading from and writing to the USB device failed several times, the device could be automatically searched and setup to the initial state.

float GetTemperature(int deviceNo) ;
float FCWDGetTemperature (CUSBaccess *obj, int deviceNo) ;
Simple and recommended function to get the current temperature. The returned value is the temperature or the value -200. to indicate an error.

float GetHumidity(int deviceNo) ;
float FCWDGetHumidity(CUSBaccess *obj, int deviceNo) ;
Simple and recommended function to get the current humidity. The returned value is the humidity or the value -200. to indicate an error.

int GetTemperature(int deviceNo, double *Temperature, int *timeID) ;
int FCWGetTemperature (CUSBaccess *obj, int deviceNo, double *Temperature, int *timeID) ;
Former request for the current temperature. Please use the simplyfied funcion call.

This function will set the temperature and timeID to the actual values. The returned time is based on a device internal time. This time is used to assure that two requests deliver different times. At least one second must separate adjacent calls. If several requests returns the same time, the device may be in trouble and a reset is necessary. This must be initiated by calling the “ResetDevice” function. The return value is 0 in case of an error, else > 0.

int GetHumidity(int deviceNo, double *Humidity, int *timeID) ;
int FCWGetHumidity (CUSBaccess *obj, int deviceNo, double * Humidity, int *timeID) ;
Former request for the current humidity. Please use the simplyfied funcion call.

This function will set the humidity and timeID to the actual values. The returned time is based on a device internal time. This time is used to assure that two requests deliver different times. At least two seconds must separate adjacent calls. The return value is 0 in case of an error, else > 0.

int SetSwitch(int deviceNo, enum SWITCH_IDs Switch, int On) ;
int FCWSetSwitch(CUSBaccess *obj, int deviceNo, enum SWITCH_IDs Switch, int On) ;
Turns an USB-Switch on or off. The argument “Switch” defines which switch to set. If the argument “On” is 1, the switch is turned on. If “On” is 0, the switch is turned off. The return value is 0 in case of an error, else > 0.

All kind of traffic light devices support flashing of the lights. It is controlled with the “On”. If the value is 16, the light will flash every 0.5 seconds. A value of 17 increases the interval to one second and 18 to 2 seconds.

int GetSwitch(int deviceNo, enum SWITCH_IDs Switch) ;
int FCWGetSwitch(CUSBaccess *obj, int deviceNo, enum SWITCH_IDs Switch) ;
Get the current switch or contact state. The argument “Switch” defines which switch to get the status from. If the returned value is 1 if the switch is on and it is 0 otherwise. In case of an error, the return value is set to –1.

int GetSeqSwitch(int deviceNo, enum SWITCH_IDs Switch, int seqNum) ;
int FCWGetSeqSwitch(CUSBaccess *obj, int deviceNo, enum SWITCH_IDs Switch, int seqNum) ;
Get the current switch state. The argument “Switch” defines which switch to get the status from. If the returned value is 1 if the switch is on and it is 0 otherwise. The seqNum argument should be 0. In case of an error, the return value is set to –1.

This command solves the problem that the USB data is buffered and maybe outdated. Using GetSeqSwitch will repeatedly get data from the buffer until the state represent the situation at the time of the call.

int ResetDevice(int deviceNo) ;
int FCWResetDevice(CUSBaccess *obj, int deviceNo) ;
Reset a Cleware device. The return value is 0 in case of an error, else > 0. If the device is a temperature or humidity sensor, the command causes a hardware reset of the sensor. Other devices will do a cold start when disconnected for a short time.

int StartDevice(int deviceNo) ;
int FCWStartDevice(CUSBaccess *obj, int deviceNo) ;
The USB-Humidity must get a start command to enter the measuring loop. This command must be send after a reset or when the commands SetValue oder GetValue were used. When StartDevice is called for other devices it will be ignored. In case of an error, the return value ist 0, otherwise != 0.

int CalmWatchdog(int deviceNo, int time1 , int time2) ;
int FCWCalmWatchdog(CUSBaccess *obj, int deviceNo, int time1 , int time2) ;
Send a life signal to an USB-Wathdog. The time until the next life signal must be detected is supplied in minutes. If “time1” is –1 the device is activated immediately (USB-AutoReset). A value of 0 will deactivate the watchdog. The return value is 0 in case of an error, else > 0.

If the device is an USB-AutoReset “time2” defines the time distance for the secondary reset for cases the first reset will not restart the PC up to point where life signals will be send again. The value is defined in minutes and could be in the range 0 – 255. If the value is 0, the secondary reset is turned off.

If the device is an USB-WatchLight, the green light is turned on when the CalmWatchdog command is executed. The argument “time1” defines the offset in seconds until the red light is turned on. The yellow light is turned on after “time2” seconds.

int GetVersion(int deviceNo) ;
int FCWGetVersion(CUSBaccess *obj, int deviceNo) ;
Version number of this device.

int GetUSBType(int deviceNo) ;
int FCWGetUSBType(CUSBaccess *obj, int deviceNo) ;
Device type. Possible values are defined in the USBtype_enum, eg. SWITCH1_DEVICE or TEMPERATURE2_DEVICE.

int GetSerialNumber(int deviceNo) ;
int FCWGetSerialNumber(CUSBaccess *obj, int deviceNo) ;
Serial number of this device.

int GetDLLVersion() ;
int FCWGetDLLVersion( ) ;
Get the version of the DLL.

int GetManualOnCount(int deviceNo) ;
int FCWGetManualOnCount(CUSBaccess *obj, int deviceNo) ;
If the device is configured as an USB-Watchdog or USB-AutoReset, this counter reflects how often a reset was done by an USB command.

int GetManualOnTime(int deviceNo) ;
int FCWGetManualOnTime(CUSBaccess *obj, int deviceNo) ;
int GetOnlineOnTime(int deviceNo) ;
int FCWGetOnlineOnTime(CUSBaccess *obj, int deviceNo) ;
These commands are used with an USB-Switch equiped with a button for manual switching. In case of an error –1 will be returned.

int GetOnlineOnCount(int deviceNo) ;
int FCWGetOnlineOnCount(CUSBaccess *obj, int deviceNo) ;
Get the “turned on by USB command” counter (USB-Switch version 15 and higher). In case of an error –1 will be returned. If the device is configured as an USB-Watchdog or USB-AutoReset, this counter reflects how often a reset was done by missing the USB life signal.

int GetMultiSwitch(int deviceNo, unsigned long int *mask, unsigned long int *value, int seqNumber) ;
int FCWGetMultiSwitch(CUSBaccess *obj, int deviceNo,...);
All input channels of the device USB-IO16 could be scanned with the command “GetMultiSwitch”. The argument “mask” indicates the channels which changes the values since the last call of GetMultiSwitch. Channel 0 is the least significant bit (LSB). The argument shows the current status of all channels. The argument “seqNum” should be set to 0. Please note: The USB-Contact is polled using GetSwitch(..).

int SetMultiSwitch(int deviceNo, unsigned long int value) ;
int FCWSetMultiSwitch(CUSBaccess *obj, int deviceNo, unsigned long int value) ;
All output channels of the USB-IO16 will be set with the command “SetMultiSwitch”. The argument “value” holds the new state of the channels. Channel 0 is the least significant bit (LSB).

int SetMultiConfig(int deviceNo, unsigned long int directions) ;
int FCWSetMultiConfig(CUSBaccess *obj, int deviceNo, unsigned long int dirs) ;
The configuration of the USB-IO16 is defined with the command SetMultiConfig. Every channel is assigned a bit in the direction argument. Channel 0 is the least significant bit (LSB). If the bit is 1 the channel is configured as an input, otherwise as an output.

int GetMultiConfig(int deviceNo) ;
int FCWGetMultiConfig(CUSBaccess *obj, int deviceNo) ;
The configuration of the USB-IO16 is returned with the command SetMultiConfig. Every channel is assigned a bit in the return value. Channel 0 is the least significant bit (LSB). If the bit is 1 the channel is configured as an input, otherwise as an output. A return value of -1 indicates an error.

int IsAmpel(int deviceNo) ;
int FCWIsAmpel(CUSBaccess *obj, int deviceNo) ;
The USB traffic light devices are registered as switch-devices. To find out, if the switch will turn a traffic light on or off, the function IsAmpel may be called. It it returns a value greater than 0, the device is a traffic light.

int IsAlarm(int deviceNo) ;
int FCWIsAlarm(CUSBaccess *obj, int deviceNo) ;
The USB-Alarm buzzer is registered as an USB-Switch. This function tells if this switch turn an integrated buzzer on and off,

int IsLuminus(int deviceNo) ;
int FCWIsLuminus(CUSBaccess *obj, int deviceNo) ;
The USB-Luminus buzzer is registered as an USB-ADC0800. This function indicates if this is the special version with an lumination sensor.

int IsCutter(int deviceNo) ;
int FCWIsCutter(CUSBaccess *obj, int deviceNo) ;
The USB-Cutter is registered as an USB-Switch. This function tells if this switch is an USB-Cutter.

int IsSolidState(int deviceNo) ;
int FCWIsSolidState(CUSBaccess *obj, int deviceNo) ;
When the switch is an electronic low voltage switch, this function return a value greater than 0..

int GetHWversion(int deviceNo) ;
int FCWGetHWversion(CUSBaccess *obj, int deviceNo) ;
Indicate the processor used to build the device. The old one until 2013 return 0, newer one will return 13.

int IsMonitoredSwitch (int deviceNo) ;

int FCW IsMonitoredSwitch (CUSBaccess *obj, int deviceNo) ;
Return true if the USB-Switch needs to be monitored.

int SelectADC(int deviceNo, int SubDevice) ;
int FCWSelectADC(CUSBaccess *obj, int deviceNo) ;
Start the mesurement cycle of channel „SubDevice“. The cycle is idetified by an internal id which is returned by this function.

float GetADC(int deviceNo, int sequenceNumber, int SubDevice) ;
float FCWGetADC(CUSBaccess *obj, int deviceNo) ;
Get the measured value of the cycle „sequenceNumber“. This function returns a value betweeen 0. and 100. or in case of an error the value -200.

int GetCounter(int deviceNo, enum COUNTER_IDs counter) ;
int FCWGetCounter(CUSBaccess *obj, int devNo, enum COUNTER_IDs countr);
The counter value of the USB-Counter is requested.

int SetTempOffset(int deviceNo, double newVal, double currVal) ;
int FCWSetTempOffset(CUSBaccess *obj, double newVal, double currVal) ;
The device USB-Temp is calibrated by using this function. The returned GetTemperature values will be corrected by the difference between newVal and currVal. The result is stored permanent inside the device. If the function fails 0 is retuned, otherwise 1.

int SyncDevice(int deviceNo, unsigned long int mask) ;
int FCWSyncDevice(CUSBaccess *obj, int deviceNo, unsigned long int mask) ;
For internal purposes.

HANDLE GetHandle(int deviceNo) ;
HANDLE FCWGetHandle (CUSBaccess *obj, int deviceNo) ;
For later use.

int GetValue(int deviceNo, unsigned char *buf, int bufsize) ;
int FCWGetValue(CUSBaccess *obj, int deviceNo, unsigned char *buf, int bufsize) ;

For later use. The return value is 0 in case of an error, else > 0.

int SetValue(int deviceNo, unsigned char *buf, int bufsize) ;
int FCWSetValue(CUSBaccess *obj, int deviceNo,unsigned char *buf,int bufsize);
For later use. The return value is 0 in case of an error, else > 0..

int SetLED(int deviceNo, enum LED_IDs Led, int value) ;
int FCWSetLED(CUSBaccess *obj, int deviceNo, enum LED_IDs Led, int value) ;
For later use. The return value is 0 in case of an error, else > 0.

int GetFrequency(int deviceNo, unsigned long int *counter, int SubDevice) ;
int FCWGetCounter(CUSBaccess *obj, int devNo, unsigned long int *counter, int SubDevice) ;
Returns the measured frequency (1/sec) or -1 in case of an error. Counter is set to the internal counter value. SubDevice defines the requested channel (0 – 3).

4. API C++ Example

The following simple example shows the usage of the API to read from a temperature sensor and to turn a switch with C++. If the program will be called without an argument, a temperature device will be searched and if found, the temperature will the read and printed 10 times. If the program will be called with an argument, a switch device is expected and turned on (1) or off (0). Any other argument will cause the program to read the current state of the switch and print it on the screen.

An special behaviour is implemented in case the program was renamed. If the name contains the string “on” or “off”, a switch is asumed and this one is turn on or off. This is useful to control the switch with cmdline comands.

```
#include "stdafx.h"
#include "USBaccess.h"
int main(int argc, char* argv[])
{
   CUSBaccess CWusb;
   printf("Start USB Access Beispiel!\n");
    int USBcount = CWusb.OpenCleware();
    printf("OpenCleware found %d devices\n", USBcount);
    int readTemperature = 1;
    int switchState = -1;
    if (argc >= 2)
    {
        readTemperature = 0;
        if (argv[1][0] == '0')
            switchState = 0;
        else if (argv[1][0] == '1')
            switchState = 1;
        // else ask for state
    }
    else
    {       // check if name contains “on” or “off”
        for (char* pt = argv[0]; *pt; pt++)
        {
            if (*pt == 'o' || *pt == 'O')
            {
                if (pt[1] > 0 && (pt[1] == 'n' || pt[1] == 'N'))
                {
                    switchState = 1;
                    break;
                }
                if (pt[1] > 0 && pt[2] > 0 && (pt[1] == 'f' || pt[1] == 'F')
                            && (pt[2] == 'f' || pt[2] == 'F'))
                {
                    switchState = 0;
                    break;
                }
```

```
            }
        }
        if (switchState >= 0)   // “on” or “off” was found
            readTemperature = 0;
    }
    if (readTemperature)
    {
        for (int devID = 0; devID < USBcount; devID++)
        {
            int devType = CWusb.GetUSBType(devID);
            if (devType != CUSBaccess::TEMPERATURE_DEVICE &&
               devType != CUSBaccess::TEMPERATURE2_DEVICE)
                continue;           // read only temperatur!
            CWusb.ResetDevice(devID);
            Sleep(500);     // wait a bit to settle after reset

            // get 10 values
            for (int cnt = 0; cnt < 10; cnt++)
            {
                double temperatur;
                temperatur = CWusb.GetTemperature(devID);
                printf("Measured %lf degrees Celsius\n", temperatur);
                Sleep(1200);
            }
        }
    }
    else
    {
        for (int devID = 0; devID < USBcount; devID++)
        {
            if (CWusb.GetUSBType(devID) == CUSBaccess::SWITCH1_DEVICE)
            {
                if (switchState >= 0)
                    CWusb.SetSwitch(devID, CUSBaccess::SWITCH_0, switchState);
                else
                {
                    int cnt = CWusb.GetOnlineOnCount(devID);
                    int state = CWusb.GetSwitch(devID, CUSBaccess::SWITCH_0);
                    printf("Switch %d: count=%d, state = %d\n",
                           devID, cnt, state);
                }
                break;
            }
        }
    }
    CWusb.CloseCleware();

    return 0;
}
```

Some command samples:

copy Example.exe SwitchOn.exe	first copy
copy Example.exe SwitchOff.exe	second copy
Example 1	turn switch on
SwitchOff	turn switch off
SwitchOn	turn switch on again
Example ?	print the current switch setting

Example	print current temperature

A typical usage of "SwitchOn" and "SwitchOff" is signaling incoming mail with the rules of "Outlook".

5. API C Example

```
#include "stdio.h"
#include "windows.h"
#include "USBaccess.h"

#define maxWatchCnt 4

DWORD WINAPI
WatchdogLoop(LPVOID lpParameter)
{
    int devCnt = 0;
    int watchIDs[maxWatchCnt];
    int watchCnt = 0;
    CUSBaccess* cw = 0;
    int i;

    cw = FCWInitObject();
    if (cw != 0) ;
    devCnt = FCWOpenCleware(cw);

    for (i = 0; i < devCnt; i++)
    {

        enum FCWUSBtype_enum type = FCWGetUSBType(cw, i);
if (type == WATCHDOG_DEVICE || type == AUTORESET_DEVICE)
watchIDs[watchCnt++] = i ;
}

if (watchCnt <= 0) {
printf("no USB-Watchdog or USB-AutoReset devices found\n");
}
else {
while (1) {      // loop forever
for (i=0 ; i<watchCnt ; i++)
FCWCalmWatchdog(cw, watchIDs[i], 1, 0); // timeout 1 minute
Sleep(1000);  // 1000 ms
}
}

return 0 ;
}


int
main(int argc, char* argv[])
{
    int DebugInfos = 0;
    int runInBackground = 0;
    char* progName = argv[0];
    int err = 0;

    for (argc--, argv++; argc > 0; argc--, argv++)
    {
```

```
        if (argv[0][0] == '-')
        {
            switch (argv[0][1])
            {
                case 'd':
                case 'D':
                    DebugInfos = 1;     // not used now
                    break;
                case 'b':
                case 'B':
                    runInBackground = 1;
                    break;
            }
        }
    }

    if (runInBackground)
    {
        char* execStr = progName;
        STARTUPINFO startupinfo;
        PROCESS_INFORMATION processInfo;
        ZeroMemory(&startupinfo, sizeof(startupinfo));
        startupinfo.cb = sizeof(startupinfo);
        startupinfo.dwFlags = 0;
        ZeroMemory(&processInfo, sizeof(processInfo));
        if (CreateProcess(0, execStr, 0, 0, FALSE,
                          NORMAL_PRIORITY_CLASS,
                          0, 0, &startupinfo, &processInfo) == 0)
        {
            err = GetLastError();
            printf("Datei %s: Fehler beim öffnen (%d)", execStr, err);
        }
    }
    else
        WatchdogLoop(0);

    return err;
}
```

6. API-Beispiel C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;       // now we found DllImport function


class cwUSB
{
    public enum SWITCH_IDs : int
    {
        SWITCH_0 = 0x10, SWITCH_1 = 0x11, SWITCH_2 = 0x12, SWITCH_3 = 0x13,
        SWITCH_4 = 0x14, SWITCH_5 = 0x15, SWITCH_6 = 0x16, SWITCH_7 = 0x17,
        SWITCH_8 = 0x18, SWITCH_9 = 0x19, SWITCH_10 = 0x1a, SWITCH_11 = 0x1b,
        SWITCH_12 = 0x1c, SWITCH_13 = 0x1d, SWITCH_14 = 0x1e, SWITCH_15 = 0x1f
    };
    public enum LED_IDs { LED_0 = 0, LED_1 = 1, LED_2 = 2, LED_3 = 3 };
    public enum USBtype_enum : int
    {
        ILLEGAL_DEVICE = 0,
        LED_DEVICE = 0x01,
        POWER_DEVICE = 0x02,
        DISPLAY_DEVICE = 0x03,
        WATCHDOG_DEVICE = 0x05,
        AUTORESET_DEVICE = 0x06,
        WATCHDOGXP_DEVICE = 0x07,
        SWITCH1_DEVICE = 0x08, SWITCH2_DEVICE = 0x09, SWITCH3_DEVICE = 0x0a,
        SWITCH4_DEVICE = 0x0b, SWITCH5_DEVICE = 0x0c, SWITCH6_DEVICE = 0x0d,
        SWITCH7_DEVICE = 0x0e, SWITCH8_DEVICE = 0x0f,
        TEMPERATURE_DEVICE = 0x10, TEMPERATURE2_DEVICE = 0x11,
        TEMPERATURE5_DEVICE = 0x15,
        HUMIDITY1_DEVICE = 0x20, HUMIDITY2_DEVICE = 0x21,
        SWITCHX_DEVICE = 0x28,      // new switch 3,4,8
        CONTACT00_DEVICE = 0x30, CONTACT01_DEVICE = 0x31,
        ADC0800_DEVICE = 0x50, ADC0801_DEVICE = 0x51
    };


    [DllImport(@"USBaccess.DLL")]
    public static extern IntPtr FCWInitObject();
    [DllImport(@"USBaccess.DLL")]
    public static extern void FCWUnInitObject(IntPtr cwHdl);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWOpenCleware(IntPtr cwHdl);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWCloseCleware(IntPtr cwHdl);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWGetUSBType(IntPtr cwHdl, int devNum);
    [DllImport(@"USBaccess.DLL")]
    public static extern float FCWDGetTemperature(IntPtr cwHdl, int devNum);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWSetLED(IntPtr cwHdl, int devNum, int Led, int value);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWSetSwitch(IntPtr cwHdl, int dvNum, int Switch, int On);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWGetSwitch(IntPtr cwHdl, int devNum, int Switch);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWSelectADC(IntPtr cwHdl, int devNum, int subDevice);
```

```
    [DllImport(@"USBaccess.DLL")]
    public static extern float FCWGetADC(IntPtr cwHdl, int dNum, int sNum, int subDev);
    [DllImport(@"USBaccess.DLL")]
    public static extern int FCWStartDevice(IntPtr cwHdl, int devNum);
}

namespace SharpTemp
{
    class Program
    {
        static void Main(string[] args)
        {
            IntPtr cwObj = cwUSB.FCWInitObject();
            int devCnt = cwUSB.FCWOpenCleware(cwObj);
            // Please note that OpenCleware should be called only once in the
            // initialisation of your programm, not every time a function is called
            System.Console.WriteLine("Found " + devCnt + " Cleware devices");
            for (int i = 0; i < devCnt; i++)
            {
                int devType = cwUSB.FCWGetUSBType(cwObj, i);
                System.Console.WriteLine("Device " + i + ": Type = " + devType);
                if ((devType == (int)cwUSB.USBtype_enum.TEMPERATURE_DEVICE) ||
                     (devType == (int)cwUSB.USBtype_enum.TEMPERATURE2_DEVICE))
                {
                    float temperature = cwUSB.FCWDGetTemperature(cwObj, i);
                    System.Console.WriteLine("Temperature at " + i + " is = " + tempera-
ture +
 "°C");
                }
                else if (devType == (int)cwUSB.USBtype_enum.SWITCH1_DEVICE)
                {
                    int state =
                                    cwUSB.FCWGetSwitch(cwObj, i,
(int)cwUSB.SWITCH_IDs.SWITCH_0);
                    System.Console.WriteLine("Switch state is = " + state);
                    cwUSB.FCWSetSwitch(cwObj, i, (int)cwUSB.SWITCH_IDs.SWITCH_0, 1);
                    System.Threading.Thread.Sleep(1000);     // wait one second
                    cwUSB.FCWSetSwitch(cwObj, i, (int)cwUSB.SWITCH_IDs.SWITCH_0, 0);
                }
                else if (devType == (int)cwUSB.USBtype_enum.ADC0800_DEVICE)
                {
                    int seqNum = cwUSB.FCWSelectADC(cwObj, i, 0);
                    // subDevice 0 is first channel, 1 would be second
                    float value = cwUSB.FCWGetADC(cwObj, i, seqNum, 0);
                    System.Console.WriteLine("Measured value at " + i + " is = " + value
+"%");
                }
            }
            if (devCnt > 0)
                cwUSB.FCWCloseCleware(cwObj);
            cwUSB.FCWUnInitObject(cwObj);   // free the allocated object
        }
    }
}
```