Programmier-Interface
für Windows®

Version 5.2.10
03.06.2022

Cleware GmbH
Am Lornsenpark 31
24837 Schleswig
www.cleware.de

1. Inhalt

2. Allgemeines zur Programmschnittstelle (API) und zu ActiveX

Die Geräte der Cleware GmbH können auch in Projekte eingebunden werden, indem sie durch eine einfache Schnittstelle integriert werden. Hierzu werden drei Dateien zur Verfügung gestellt, die Sie im Downloadbereich von ClewareControl finden:

USBaccess.h	Definition der Schnittstelle
USBaccess.lib	Linkinformationen
USBaccess.dll	Ausführbare Routinen
USBaccessX64.dll	Interface in 64 bit
USBaccessX64.lib	Linkinformationen für 64 bit

Hinweis zu ActiveX: Diese Schnittstelle wird ab Version 4 nicht mehr unterstützt.

Ab der Version 4.5.0 können beim Aufruf der Funktionen statt der „devNo“ nun auch Seriennummern verwendet werden.

Mit Version 5.2.6 wurde das FCW-Interface vereinfacht, damit der Zugriff nur noch über Integer-Variablen erfolgt. Das vereinfacht den Zugriff aus X64-Umgebungen, z.B. Python. Wenn das erste Argument der FCW-Funktionen auf 0 gesetzt ist, wird ein intern gespeichertes USBaccess-Objekt verwendet.
Eine Anwendung ist der Beschreibung für Python am Ende dieser Dokumentation zu finden.

In Version 5.2.9 wurden SetCounter und GetContact hinzugefügt.

3. USBaccess.h

Die Datei USBaccess.h beinhaltet die Schnittstelle zu dem USB-Geräten der Cleware GmbH. Nach dem Einbinden der Datei können die Geräte geöffnet und gelesen werden.

```
const int USBaccessVersion = 529 ;

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
             .. die anderen Einträge sind nur für internen Einsatz  } ;
    enum LED_IDs {     LED_0=0, LED_1=1, LED_2=2, LED_3=3 } ;
    enum SWITCH_IDs {  SWITCH_0=0x10,      // use this
                       SWITCH_1=0x11, …, SWITCH_15=0x1f } ;
    enum USBtype_enum {   ILLEGAL_DEVICE=0,
                          LED_DEVICE=0x01,
                          WATCHDOG_DEVICE=0x05,
                          AUTORESET_DEVICE=0x06,
                          WATCHDOGXP_DEVICE=0x07,
```

```
                        SWITCH1_DEVICE=0x08, SWITCH2_DEVICE=0x09,
                        SWITCH3_DEVICE=0x0a, SWITCH4_DEVICE=0x0c,
                        TEMPERATURE_DEVICE=0x10,
                        TEMPERATURE2_DEVICE=0x11,
                        TEMPERATURE5_DEVICE=0x15,
                        HUMIDITY1_DEVICE=0x20, HUMIDITY2_DEVICE=0x21,
                        SWITCHX_DEVICE=0x28,
                        CONTACT00_DEVICE=0x30, CONTACT01_DEVICE=0x31,
                        …, CONTACT15_DEVICE=0x3f
                        } ;
private:
   class CUSBaccessBasic * X  ;

public:
   CUSBaccess() ;
   virtual ~CUSBaccess() ;   // maybe used as base class

   virtual int OpenCleware() ;  // returns number of Cleware devices
   virtual int CloseCleware() ; // close all Cleware devices
   virtual HANDLE GetHandle(int deviceNo) ;
   virtual int  Recover(int devNum) ;
   virtual int¹ GetValue(int devNo, unsigned char *buf,int bufsize) ;
   virtual int¹ SetValue(int devNo, unsigned char *buf, int bufsize) ;
   virtual int¹ SetLED(int deviceNo, enum LED_IDs Led, int value) ;
                              // value: 0=off 7=medium 15=highlight
   virtual int¹ SetSwitch(int deviceNo,enum SWITCH_IDs Switch,int On);
                                                    // On: 0=off, 1=on
   virtual int² GetSwitch(int deviceNo, enum SWITCH_IDs Switch) ;
   virtual int² GetSeqSwitch(int deviceNo, enum SWITCH_IDs Switch,
                             int seqNum) ; // On: 0=off, 1=on, -1=error
   virtual int² GetSwitchConfig(int deviceNo, int *switchCount,
                                          int *buttonAvailable) ;
   virtual int¹ GetTemperature(int deviceNo) ;
   virtual int¹ GetHumidity(int deviceNo) ;
   virtual int¹ GetTemperature(int devNo, double *Temp,int *timeID);
   virtual int¹ GetHumidity(int devNo, double *Humidity, int *timeID) ;
   virtual int  SelectADC(int deviceNo, int subDevice) ;
   virtual float GetADC(int deviceNo, int sequenceNumber, int subDevice);
   virtual int¹ ResetDevice(int deviceNo) ;
   virtual int¹ StartDevice(int deviceNo) ;
   virtual int¹ CalmWatchdog(int devNo, int min, int min2restart) ;
   virtual int  GetVersion(int deviceNo) ;
   virtual int  GetUSBType(int deviceNo) ;
   virtual int  GetSerialNumber(int deviceNo) ;
   virtual int  GetDLLVersion() { return USBaccessVersion ; }
   virtual int² GetManualOnCount(int deviceNo) ;
   virtual int² GetManualOnTime(int deviceNo) ;
   virtual int² GetOnlineOnCount(int deviceNo) ;
   virtual int² GetOnlineOnTime(int deviceNo) ;
   virtual int² GetMultiSwitch(int deviceNo, unsigned long int *mask,
                               unsigned long int *value, int seqNumber) ;
   virtual int² GetContact(int deviceNo) ; // returns (mask<<16)+value
   virtual int² SetMultiSwitch(int deviceNo, unsigned long int value);
   virtual int² SetMultiConfig(int deviceNo, unsigned long int dir) ;
   virtual int² GetCounter(int deviceNo, int counterID) ;
   virtual int² SetCounter(int deviceNo, int counter, int counterID) ;
   virtual int² SyncDevice(int deviceNo, unsigned long int mask) ;
   virtual int² GetMultiConfig(int deviceNo) ;
   virtual int² IsAmpel(int deviceNo) ;
```

```
      virtual int² IsLuminus(int deviceNo) ;
      virtual int² IsAlarm(int deviceNo) ;
      virtual int² IsCutter(int deviceNo) ;
      virtual int² IsSolidState(int deviceNo) ;
      virtual int² GetHWversion(int deviceNo) ;
      virtual int² IsMonitoredSwitch(int deviceNo) ;
      virtual int² SetTempOffset(int deviceNo, double Soll, double Ist) ;
      virtual int² GetFrequency(int devNo, unsigned long int *cnt, int s);
   } ;

extern "C" {
   USBACCESS_API CUSBaccess * _stdcall USBaccessInitObject() ;
   USBACCESS_API void _stdcall USBaccessUnInitObject(CUSBaccess *) ;
   } ;
...

// functional C interface (FCW = Function CleWare)

   USBACCESS_API CUSBaccess * _stdcall FCWInitObject() ;
   USBACCESS_API void _stdcall    FCWUnInitObject(CUSBaccess *obj) ;
   USBACCESS_API int _stdcall     FCWOpenCleware(CUSBaccess *obj) ;
   USBACCESS_API int _stdcall     FCWCloseCleware(CUSBaccess *obj) ;
   USBACCESS_API int _stdcall     FCWRecover(CUSBaccess *obj,
                                             int deviceNo) ;
…

¹ = Returnwert TRUE falls ok, FALSE im Fehlerfall
² = Returnwert im Fehlerfall -1
```

Die Methoden in der Klasse CUSBaccess wurde jetzt mit dem Attribut „virtual“ versehen, damit eine Verwendung mit Delphi möglich wird. Zur Nutzung der Klasse unter Delphi muss die Funktion USBaccessInitObject aufgerufen werden, da die Klasse unter C++ angelegt werden muss.

Alternativ gibt es die Möglichkeit, ohne die Verwendung von Klassen mit einfachen Funktionen auf die Cleware Geräte zuzugreifen. Das erlaubt die Nutzung von C als Programmiersprache. Es stehen alle Methoden der Klasse USBaccess als Funktion zur Verfügung. Vor dem Namen der Methode wurde das Kürzel FCW vorangestellt, um eine namentliche Unterscheidung zu erlauben.

Vor der ersten Verwendung der Funktionen muss der Zugriff mit dem Aufruf von FCWInitObject() initialisiert werden. Der hier zurückgelieferte Wert wird als erstes Argument sämtlicher Funktionen benötigt. Vor dem Beenden des Programms sollte die Funktion FCWUnInitObject(obj) aufgerufen werden. Ein Beispiel für die Anwendung der Funktionen ist in Kapitel 5 beschrieben.

Die API Version 5.2.6 vereinfacht die Verwendung des Interfaces. Das von FCWInitObject zurückgegebene Objekt kann durch die Zahl 0 in den folgenden Aufrufen von FCW Funktionen ersetzt werden. Das ist hilfreich, wenn die Skript-Sprache nur mit Integer Werten gut klar kommt. Aus dem gleichen Grund wurden auch die Funktionen FCWIGetTemperature und FCWIGetHumidity zur Verfügung gestellt. Diese Fuktionen geben den Messwert, multipliziert mit 1000., als Integer Zahl zurück. Ein Anwendungsbeispiel ist im Python Kapitel zu finden.

Der Aufruf von OpenCleware erstellt ein Feld (Array) mit Methoden zum Zugriff auf die Geräte- Die Anzahl der Feldelemente entspricht den gefundenen Cleware USB Geräten. Da die Zuordnung, welches Gerät an welcher Stelle des Feldes zu finden ist, mehr oder weniger zufällig ist, muss bei mehreren Geräten vor deren Verwendung sichergestellt werden, dass es sich um das gewünschte Gerät handelt.

Zur Auswahl der Geräte werden die Funktionen mit einem Parameter „deviceNo“ aufgerufen. Das ist ein Index in dem Feld, beginnend bei 0 … (Anzahl USB-Geräte – 1).

Ab der Version 4.5.0 können beim Aufruf der Funktionen als „deviceNo“ nun auch Seriennummern übergeben werden.

Alle USB-Ampeln ab der Geräte-Version 104 (ab September 2014) unterstützen das Blinken. Es wird mit der Funktion SetSwitch gesteuert.

Der neue „USB-Switch++M“, ein abgesicherter Netzschalter, der beim Ausbleiben von Lebenszeichen automatisch abschaltet, wird mit CalmWatchdog abgesichert.

4. API-Funktionen

CUSBaccess *FCWInitObject() ;
Initialisiert den funktionalen Zugriff auf die API-Funktionen.

void FCWUnInitObject(CUSBaccess *obj) ;
Beendet den funktionalen Zugriff auf die API-Funktionen.

int OpenCleware() ;
int FCWOpenCleware(CUSBaccess *obj) ;
Sucht die angeschlossenen Cleware USB-Geräte und öffnet diese. Die Anzahl der gefundenen Geräte wird zurückgegeben.

int CloseCleware() ;
int FCWCloseCleware(CUSBaccess *obj) ;
Schließt die Verbindung zu den Cleware USB-Geräten.

int Recover (int deviceNo) ;
int FCWRecover (CUSBaccess *obj, int deviceNo) ;
Wenn das Lesen oder Schreiben mit dem USB-Gerät mehrfach mit einem Fehler abbricht, kann mit dem Aufruf von Recover die Verbindung zu diesem Gerät regeneriert werden.

float GetTemperature(int deviceNo) ;
float FCWDGetTemperature (CUSBaccess *obj, int deviceNo) ;
int FCWDGetTemperature (CUSBaccess *obj, int deviceNo) ;
Vereinfachte Abfrage der Temperatur eines Sensors. Die Funktion „GetTemperature“ liefert entweder die Temperatur oder im Fehlerfall einen Wert von -200. Bei der I-Funktion ist der Rückgabewert mit 1000. multipliziert ein Integer.

float GetHumidity(int deviceNo) ;
float FCWDGetHumidity(CUSBaccess *obj, int deviceNo) ;
int FCWIGetHumidity(CUSBaccess *obj, int deviceNo) ;
Vereinfachte Abfrage der Luftfeuchtigkeit eines Sensors. Die Funktion „GetHumidity“ liefert entweder die relative Luftfeuchtigkeit oder im Fehlerfall einen Wert von -200. Bei der I-Funktion ist der Rückgabewert mit 1000. multipliziert ein Integer.

int GetTemperature(int deviceNo, double *Temperature, int *timeID) ;
int FCWGetTemperature (CUSBaccess *obj, int deviceNo, double *Temperature, int *timeID) ;
Elementare Abfrage der Temperatur eines Sensors. Es wird empfohlen, die einfache Version von GetTemperature zu verwenden.

Die Funktion „GetTemperature“ liefert neben der Temperatur auch eine Zeitangabe. Diese Zeit wird aus einer internen Zeitbasis innerhalb des Sensors abgeleitet. Wenn ein Meßwert abgefragt wird, sollte anhand der Zeit überprüft werden, ob die Zeit größer als die Zeit der letzten Abfrage ist. Ist dies nicht der Fall, lag noch kein neuer Messwert vor. Wenn sich die

Zeit mehrfach hintereinander nicht verändert, sollte der Sensor mit der Funktion „ResetDevice“ neu initialisiert werden. Zwischen den einzelnen Abfragen des Temperatursensors sollte aber mindestens eine Sekunde liegen. Der Rückgabewert ist 0 im Fall eines Fehlers, ansonsten > 0.

int GetHumidity(int deviceNo, double *Humidity, int *timeID) ;
int FCWGet Humidity (CUSBaccess *obj, int deviceNo, double * Humidity, int *timeID) ;

Abfrage des Feuchtigkeitmeßwertes des Sensors. Es wird empfohlen, die einfache Version von GetHumidity zu verwenden.

Die Funktion „GetHumidity“ liefert neben der Feuchtigkeit zwischen 0 und 100 % auch eine Zeitangabe. Diese Zeit wird aus einer internen Zeitbasis innerhalb des Sensors abgeleitet. Wenn ein Meßwert abgefragt wird, sollte anhand der Zeit überprüft werden, ob die Zeit größer als die Zeit der letzten Abfrage ist. Ist dies nicht der Fall, lag noch kein neuer Meßwert vor. Zwischen den einzelnen Abfragen des Feuchtigkeitssensors sollte aber mindestens zwei Sekunden liegen. Der Rückgabewert ist 0 im Fall eines Fehlers, ansonsten > 0.

int SetSwitch(int deviceNo, enum SWITCH_IDs Switch, int On) ;
int FCWSetSwitch(CUSBaccess *obj, int deviceNo, enum SWITCH_IDs Switch, int On) ;

Setzen des Schalters USB-Switch. Das Argument „Switch“ bestimmt, welcher Schalter angesprochen werden soll. Mit einem Wert On=1 wird der Schalter eingeschaltet und mit 0 ausgeschaltet Der Rückgabewert ist 0 im Fall eines Fehlers, ansonsten > 0. Bei dem Produkt USB-Relais wird mit dem Argument „SWITCH_1“ das zweite Relais angesteuert.

Alle USB-Ampeln ab der Geräte-Version 104 (ab September 2014) unterstützen das Blinken. Es wird mit das Argument „On“ gesteuert. Normalerweise ist hier 0=aus und 1=ein. Wenn Sie hier nun den Wert 16 setzen, wird die angesprochene Leuchte schnell blinken (Takt 0,5 Sekunden). Sie können aber auch den Wert 17 verwenden, dann verdoppelt sich der Takt auf 1 Sekunde, bei 18 ist er alle 2 Sekunden.

int GetSwitch(int deviceNo, enum SWITCH_IDs Switch) ;
int FCWGetSwitch(CUSBaccess *obj, int deviceNo, enum SWITCH_IDs Switch) ;

Abfrage des Schalters USB-Switch und des USB-Contact. Das Argument „Switch“ bestimmt, welcher Schalter bzw. Kontakt angefragt werden soll. Bei dem Produkt USB-Relais wird mit dem Argument „SWITCH_1“ das zweite Relais angesteuert.Die Antwort ist 1 wenn der Schalter eingeschaltet und 0 wenn er ausgeschaltet ist. Ein Rückgabewert von –1 signalisiert einen Fehler.

Wird „GetSwitch“ erstmals nach dem Hochfahren des PCs aufgerufen, wird ein eingeschalteter Schalter manchmal nicht richtig erkannt. Eine sichere Abfrage wird hier erreicht, indem vor der Abfrage „GetSwitch“ die Abfrage „GetOnlineOnCount“ aufgerufen wird. Alternativ kann GetSeqSwitch verwendet werden.

int GetSeqSwitch(int deviceNo, enum SWITCH_IDs Switch, int seqNum) ;
int FCWGetSeqSwitch(CUSBaccess *obj, int deviceNo,
enum SWITCH_IDs Switch, int seqNum) ;
Abfrage des Schalters USB-Switch und des USB-Contact. Das Argument „Switch“ bestimmt, welcher Schalter angefragt werden soll. Die Antwort ist 1 wenn der Schalter eingeschaltet und 0 wenn er ausgeschaltet ist. Das Argument „seqNum“ sollte 0 sein. Ein Rückgabewert von –1 signalisiert einen Fehler.

Mit diesem Befehl kann das Problem behoben werden, daß die Abfrage des USB-Zustands vom Betriebssystem über einen mehrstufigen Zwischenpuffer erfolgt. Der Zwischenpuffer wird abgebaut und der Rückgabewert entspricht dem Zustand zum Zeitpunkt des Abfrage.

int GetSwitchConfig(int deviceNo, int *switchCount, int *buttonAvailable) ;
int FCWGetSwitchConfig(CUSBaccess *obj,
int deviceNo, int *switchCount, int *buttonAvailable) ;
Abfrage der Eigenschaften Schalters USB-Switch bzw. USB-Relais. Dem Argument „switchCount“ wird die Anzahl der verfügbaren Schalter innerhalb des Gerätes zugewiesen. Das Argument „buttonAvailable” gibt an, ob das Gerät mit einem eingebauten Taster manuell bedienbar ist, wie beispielsweise in dem Proukt „USB-Plug“.

int ResetDevice(int deviceNo) ;
int FCWResetDevice(CUSBaccess *obj, int deviceNo) ;
Das angewählte Gerät neu initialisieren. Der Rückgabewert ist 0 im Fall eines Fehlers, ansonsten > 0. Der Aufruf dieser Funktion führt bei den Sensoren für Temperatur und Feuchtigkeit zu einem Reset des Sensorelements. Bei allen anderen Geräten sorgt der Aufruf dafür, das nach dem Trennen der USB-Stromversorgung in jedem Fall ein Kaltstart durchgeführt wird. Die alten vorher eingestellten Zustände werden ignoriert.

int StartDevice(int deviceNo) ;
int FCWStartDevice(CUSBaccess *obj, int deviceNo) ;
Der USB-Humidity benötigt ein Kommando zum Starten der Meßautomatik. Das Kommando ist notwendig nach einem Reset und wenn die Kommandos SetValue oder GetValue verwendet wurden. Ein Aufruf des StartDevice-Befehls bei anderen USB-Geräten wird ignoriert. Falls das Kommando fehlschlägt, wird 0 zurückgegeben, ansonsten != 0.

int GetUSBType(int deviceNo) ;
int FCWGetUSBType(CUSBaccess *obj, int deviceNo) ;
Art des Gerätes. Mögliche Werte sind dem USBtype_enum aufgelistet, z.B. SWITCH1_DEVICE oder TEMPERATURE2_DEVICE.

int GetVersion(int deviceNo) ;
int FCWGetVersion(CUSBaccess *obj, int deviceNo) ;
Version des Gerätes.

int CalmWatchdog(int deviceNo, int time1 , int time2) ;
int FCWCalmWatchdog(CUSBaccess *obj, int deviceNo, int time1 , int time2) ;
Der USB-Watchdog bzw. USB-AutoReset wird beruhigt. Die Zeit bis zum Auslösen des Alarms wird mit dem Argument „time1“ im Bereich von 1 bis 254 Minuten angegeben. Bei dem Gerät USB-AutoReset kann ein sofortiger Reset ausgelöst werden, wenn hier der Wert –1 bzw. 255 übergeben wird. Ist der Wert 0, wird der AutoReset deaktiviert.

Der USB-Switch++M erhält mit diesem Befehl die Zeit bis zur automatisch Abschaltung. Die Zeit time1 wird in Sekunden angegeben, time2 ist ohne Bedeutung.

Das Argument „time2“ definiert die Zeitdauer bis zum zweiten Reset, wenn das System auf den ersten Reset nicht reagiert hat. Der Wert kann zwischen 0 und 255 Minuten liegen, wobei ein Wert 0 den zweiten Reset deaktiviert. Die Zeitdauer des zweiten Resets wird auch als Zeit bis zum Reset nach einem Kaltstart des Systems angenommen. Ist der Wert 0, erfolgt nach dem Kaltstart kein Reset bis das erste Lebenszeichen empfangen wurde.

Ist das angesprochene Gerät ein USB-WatchLight wird mit Empfang des Befehl CalmWatchdog die grüne Leuchte eingeschaltet. Mit dem Argument „time1“ wird die Zeit bis zum Einschalten der roten und mit „time2“ der gelben Leuchte definiert. Diese Zeiten werden im Bereich von 1 – 255 Sekunden angegeben.

Wenn das Gerät ein USB-SwitchM (monitored Switch) ist, definiert das Argument time1 die Zeit in Sekunden, die das Gerät im eingeschalteten Zustand verbleibt, bevor eine Abschaltung erzwungen wird. Das Kommando muss also regelmässig innerhalb der Zeit time1 gesendet werden, damit der USB-SwitchM aktiv bleibt.

int GetSerialNumber(int deviceNo) ;
int FCWGetSerialNumber(CUSBaccess *obj, int deviceNo) ;
Seriennummer des Gerätes abfragen.

int GetDLLVersion() ;
int FCWGetDLLVersion( ) ;
Version der DLL abfragen.

int GetManualOnCount(int deviceNo) ;
int FCWGetManualOnCount(CUSBaccess *obj, int deviceNo) ;
Bei den Geräten USB-Watchdog und USB-AutoReset gibt dieser Wert an, wie oft ein Reset durch ein USB-Kommando ausgelöst wurde.

int GetManualOnTime(int deviceNo) ;
int FCWGetManualOnTime(CUSBaccess *obj, int deviceNo) ;
int GetOnlineOnTime(int deviceNo) ;
int FCWGetOnlineOnTime(CUSBaccess *obj, int deviceNo) ;
Diese Befehle wird bei USB-Switches mit manuellem Schalter verwendet. Im Fehlerfall ist der Wert –1.

int GetOnlineOnCount(int deviceNo) ;
int FCWGetOnlineOnCount(CUSBaccess *obj, int deviceNo) ;
Bei den Geräten USB-Watchdog und USB-AutoReset gibt dieser Wert an, wie oft ein Reset durch das Fehlen des Lebenssignals ausgelöst wurde.

int GetContact(int deviceNo) ;
int FCWGetContact(CUSBaccess *obj, int deviceNo) ;
Der Zustand des USB-Contact und ähnlicher Geräte wird als Integer zurückgegeben. In den unteren 16 Bit ist der aktuelle Zustand des Kontaktes angezeigt, in der oberen Hälfte wird angezeigt, ob sich der Zustand seit dem letzten Aufruf geändert hat. Siehe auch das Python-Beispiel am Ende der Doku.

int GetMultiSwitch(int deviceNo, unsigned long int *mask,
unsigned long int *value, int seqNumber) ;
int FCWGetMultiSwitch(CUSBaccess *obj, int deviceNo,...);
Alle Eingänge der Geräte USB-IO16 können mit „GetMultiSwitch“ gleichzeitig abgefragt werden. Die Maske „mask“ zeigt bitweise die Kanäle an, die sich seit dem letzten Aufruf geändert haben. Kanal 0 ist das letzte Bit rechts (LSB). In „value“ sind die aktuellen Werte gespeichert. Das Argument „seqNum“ sollte 0 sein.

int SetMultiSwitch(int deviceNo, unsigned long int value) ;
int FCWSetMultiSwitch(CUSBaccess *obj, int deviceNo, unsigned long int value) ;
Alle Ausgänge des USB-IO16 können mit „SetMultiSwitch“ gleichzeitig gesetzt werden. Kanal 0 ist das letzte Bit rechts (LSB).

int SetMultiConfig(int deviceNo, unsigned long int directions) ;
int FCWSetMultiConfig(CUSBaccess *obj, int deviceNo, unsigned long int dirs) ;
Die Konfiguration des USB-IO16 kann mit „SetMultiConfig“ eingestellt werden. Jedes Bit enspricht einem Kanal, wobei Kanal 0 ist das letzte Bit rechts (LSB) ist. Ist das Bit 1 handelt es sich um einen Eingang, bei einer 0 um einen Ausgang.

int GetMultiConfig(int deviceNo) ;
int FCWGetMultiConfig(CUSBaccess *obj, int deviceNo) ;
Die Konfiguration des USB-IO16 kann mit „GetMultiConfig“ abgefragt werden. Jedes Bit im Rückgabewert enspricht einem Kanal, wobei Kanal 0 ist das letzte Bit rechts (LSB) ist. Ist das Bit 1 handelt es sich um einen Eingang, bei einer 0 um einen Ausgang. Wird der Wert -1 zurückgegeben, ist ein Fehler beim Aufruf des Programms aufgetreten.

int IsAmpel(int deviceNo) ;
int FCWIsAmpel(CUSBaccess *obj, int deviceNo) ;
Die USB-Ampeln melden sich im Interface als Switch-Device an. Um zu erkennen, ob mit den Schaltern eine Ampel bedient wird, dient der Aufruf IsAmpel. Wenn dieser einen Wert grösser 0 antwortet, handelt es sich bei dem Gerät um eine Ampel.

int IsAlarm(int deviceNo) ;
int FCWIsAlarm(CUSBaccess *obj, int deviceNo) ;
Die USB-Alarm Summer melden sich beim PC als Switch-Device an. Um zu erkennen, ob mit dem Schalter ein Summer geschaltet wird, dient der Aufruf IsAlarm. Wenn dieser einen Wert grösser 0 antwortet, handelt es sich bei dem Gerät um einen USB-Alarm.

int IsLuminus(int deviceNo) ;
int FCWIsLuminus(CUSBaccess *obj, int deviceNo) ;
Der Lichtstärke-Sensor USB-Luminus meldt sich beim PC als ADC-Device an. Um zu erkennen, ob es ein Lichtstärke-Sensorist, dient der Aufruf IsLuminus. Wenn dieser einen Wert ungleich 0 antwortet, handelt es sich bei dem Gerät um einen USB-Luminus.

int IsCutter(int deviceNo) ;
int FCWIsCutter(CUSBaccess *obj, int deviceNo) ;
Der USB-Cutter melden sich beim PC als Switch-Device an. Um zu erkennen, ob mit dem Schalter ein USB-Cutter geschaltet wird, dient der Aufruf IsCutter. Wenn dieser einen Wert grösser 0 antwortet, handelt es sich bei dem Gerät um einen USB-Cutter.

int IsSolidState(int deviceNo) ;
int FCWIsSolidState(CUSBaccess *obj, int deviceNo) ;
Handelt es sich bei dem USB-Schalter um einen elektronischen Niedervolt-Schalter, liefert dieser Aufruf den Wert 1 zurück.

int GetHWversion(int deviceNo) ;
int FCWGetHWversion(CUSBaccess *obj, int deviceNo) ;
Zeigt an, ob das Gerät auf einem alten (bis 2013, Antwort 0) oder einen neuen (ab 2013, Antwort 13) Prozessor basiert.

int IsMonitoredSwitch (int deviceNo) ;
int FCW IsMonitoredSwitch (CUSBaccess *obj, int deviceNo) ;
Gibt true zurück, wenn es sich um einen überwachten USB-Switch handelt.

int SelectADC(int deviceNo, int SubDevice) ;
int FCWSelectADC(CUSBaccess *obj, int deviceNo) ;
Starte die interne Messung des Kanals „SubDevice“. Die Messung erhält eine interne Nummer, die als Ergebnis der Funktion zurück gegeben wird.

float GetADC(int deviceNo, int sequenceNumber, int SubDevice) ;
float FCWGetADC(CUSBaccess *obj, int devNo, int seqNumber, int SubDevice) ;
float FCWGetADC(CUSBaccess *obj, int devNo, int seqNumber, int SubDevice) ;
Hole den gemessenen Wert aus dem USB-ADC. Die Sequenznumber ist der Wert, der von SelectADC zurückgegeben wurde. Das Ergebnis ist ein Wert zwischen 0 und 100. oder im Fehlerfall einen Wert von -200. Bei der I-Funktion ist der Rückgabewert mit 1000. multipliziert ein Integer.

int GetCounter(int deviceNo, enum COUNTER_IDs counter) ;
int FCWGetCounter(CUSBaccess *obj, int devNo, enum COUNTER_IDs countr) ;
Der Zählerstand des USB-Counter wird ausgelesen.

int SetCounter(int deviceNo, int counter, enum COUNTER_IDs counterID) ;
int FCWSetCounter(CUSBaccess *obj, int devNo,
int counter, enum COUNTER_IDs counterID);
Der Zähler *counterID* wird auf den Wert *counter* gesetzt.

int SetTempOffset(int deviceNo, double SollWert, double IstWert) ;
int FCWSetTempOffset(CUSBaccess *obj, double SollWert, double IstWert) ;
Es wird eine Kalibrierung des USB-Temp durchgeführt. Die GetTemperature Messwerte werden um die Differenz zwischen IstWert und Sollwert angepaßt. Die Korrektur wird dauerhaft im Gerät gespeichert. Schlägt die Funktion fehl wird 0 zurückgegeben, ansonsten 1.

int SyncDevice(int deviceNo, unsigned long int mask) ;
int FCWSyncDevice(CUSBaccess *obj, int deviceNo, unsigned long int mask) ;
Für interne Zwecke.

HANDLE GetHandle(int deviceNo) ;
HANDLE FCWGetHandle (CUSBaccess *obj, int deviceNo) ;
Für spätere Anwendungen.

int GetValue(int deviceNo, unsigned char *buf, int bufsize) ;
int FCWGetValue(CUSBaccess *obj, int deviceNo, unsigned char *buf,
int bufsize) ;
Für spätere Anwendungen. Der Rückgabewert ist 0 im Fall eines Fehlers, sonst > 0.

int SetValue(int deviceNo, unsigned char *buf, int bufsize) ;
int FCWSetValue(CUSBaccess *obj, int deviceNo,unsigned char *buf,int bufsize);
Für spätere Anwendungen. Der Rückgabewert ist 0 im Fall eines Fehlers, sonst > 0.

int SetLED(int deviceNo, enum LED_IDs Led, int value) ;
int FCWSetLED(CUSBaccess *obj, int deviceNo, enum LED_IDs Led, int value) ;
Für spätere Anwendungen. Der Rückgabewert ist 0 im Fall eines Fehlers, sonst > 0.

int GetFrequency(int deviceNo, unsigned long int *counter, int SubDevice) ;
int FCWGetCounter(CUSBaccess *obj, int devNo, unsigned long int *counter,
int SubDevice) ;
Die gemessene Taktfrequenz (1/s) wird returniert. Tritt ein Fehler auf wird -1 zurück gegeben. In Counter wird der Wert des internen Zählers zurückgegeben. SubDevice gibt den verwendeten Kanal im Bereich 0-3 an.

5. API-Beispiel C++

Das folgende einfache Beispiel names „Example“ zeigt die Anwendung der API zum Auslesen des Temperatursensors und zum Schalten eines Schalters unter C++. Wird Example mit dem Argument 0 aufgerufen, wird ein angeschlossener Schalter ausgeschaltet. Ist das Argument eine 1, wird der Schalter eingeschaltet. Ein anderes Argument, z.B. „?“ führt zum Auslesen des aktuellen Schaltzustandes.

Das Programm kann natürlich beliebig umgenannt werden. Der Name des Programms wird ebenfalls ausgewertet, um einfache Anwendungen zu erlauben. Ist in dem Namen der Text „on“ vorhanden, schaltet das Programm den Schalter ein. Beinhaltet der Name den Text „off“, schaltet der Schalter aus. Ansonsten versucht das Programm beim Aufruf ohne Argument, einen Sensor USB-Temp auszulesen und einige Meßwerte anzuzeigen.

```
#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;
}
```

Hier einige Beispiel-Befehle:

copy Example.exe SwitchOn.exe	Kopie anlegen
copy Example.exe SwitchOff.exe	2. Kopie anlegen
Example 1	Schaltet den Schalter ein
SwitchOff	Schaltet den Schalter aus
SwitchOn	Schaltet den Schalter wieder ein
Example ?	Antwortet mit dem Schaltzustand
Example	Zeigt einige Meßwerte des USB-Temp

6. API-Beispiel C

```
// WatchService.cpp : Send a signal to all watchdog devices every second
// Options: -b   run a thread in background

#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 ;
   }
```

7. API-Beispiel C#

```
// Simple example to show how to access Cleware device through C#
// Please extend DLL import section as needed
// 15.03.2012   Version 1.0

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 = " +
temperature + "°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
        }
    }
}
```

8. API-Beispiel Python

*from ctypes import **
mydll=windll.LoadLibrary("USBaccessX64.dll")
cw=mydll.FCWInitObject()
devCnt=mydll.FCWOpenCleware(0)
print("found ", devCnt, " devices")
serNum=mydll.FCWGetSerialNumber(0,0)
devType=mydll.FCWGetUSBType(0,0) # Diese Aufrufe nur zur Info
print("first serNum = ", serNum, ", devType = ", devType)
Temperature = mydll.FCWIGetTemperature(0, 0)
print("Temperature = ", Temperature/1000.)

Ein Beispiel für GetContact

```
# test if USB-Contact was pressed

from ctypes import *
import time
```

```
mydll=windll.LoadLibrary("USBaccessX64.dll")    # sometimes, the full path must be
declared

cw=mydll.FCWInitObject()
devCnt=mydll.FCWOpenCleware(0)
if (devCnt != 1) :
   print("no device found")
   exit()
state = 0
aufrufe=0
while (1) :
   contact = mydll.FCWGetContact(0, 0)
   newstate = contact & 1
   changed = contact & 0x10000
   if (state != newstate) : print("Contact = ", newstate)
   else :
      if (changed != 0) : print("contact changed, now", newstate)
   state = newstate
   time.sleep(0.5)
```