Programmier-Interface

für Windows®

Version 3.1.3
20.01.2005

Cleware GmbH
Nedderend 3
24876 Hollingstedt
www.cleware.de

Inhalt

1. Allgemeines zur Programmschnittstelle (API) und zu ActiveX.....3

2. USBaccess.h 6

3. API-Funktionen.....9

4. API-Beispiel C++.....14

5. API-Beispiel C 16

1. Allgemeines zur Programmschnittstelle (API) und zu ActiveX

Die Geräte der Cleware GmbH können auch Teil dritter Produkte werden, indem sie durch eine einfache Schnittstelle integriert werden. Hierzu werden fünf Dateien zur Verfügung gestellt, die Sie im Installationsverzeichnis von ClewareControl finden:

1. USBaccess.h – Definition der Schnittstelle
2. UBSaccess.lib – Linkinformationen
3. USBaccess.dll – Ausführbare Routinen
4. USBswitchAX.ocx – ActiveX-Interface zum USB-Switch
5. USBtempAX.ocx – ActiveX-Interface zum USB-Temp

Die Methoden zur Kontrolle des Temperatursensors über ActiveX sind:

CountSensors [PropGet] returns int	- Anzahl der angeschlossenen Sensoren
SensorNumber [PropGet] returns int	- Nummer des aktuellen Sensors
SensorNumber [PropPut] input int	- Auswahl des Sensors (1 ... Anzahl Sensoren)
Temperature [PropGet] returns double	- die gemessene Temperatur des akt. Sensors
Humidity [PropGet] returns double	- die realtive Feuchtigkeit des akt. Sensors
SerialNumber [PropGet] returns int	- Seriennummer der akt.Sensors

Wenn nur ein Sensor angeschlossen ist, ist der einzige Sensor aktiv und muß nicht mehr mit „SensorNumber“ aktiviert werden. Eine von der Methode „Temperature“ gelieferte Temperatur von -200. Grad bedeutet, daß kein Meßwert besorgt werden konnte und ein Fehler vorliegt. Für die Methode „Humidity“ wird ein Fehler durch einen negativen Feuchtigkeitswert signalisiert.

Die Methoden zur Kontrolle des Netzschalters über „USBswitchAX.ocx“ sind sehr ähnlich aufgebaut:

CountSwitches [PropGet] returns int	- Zahl der gefundenen Schalter / Watchdogs
SwitchNumber [PropGet] returns int	- Nummer des aktuellen Schalters
SwitchNumber(long nummer) [PropPut]	- Auswahl des Schalters (1 ... Anzahl Schalter)
SwitchState[PropGet] returns int	- der Zustand des Schalters
SwitchState(long neu) [PropPut]	- Schalter setzen
SerialNumber [PropGet] returns int	- Seriennummer der akt. Schalters
SerialNumber (long neu) [PropPut]	- Gerät mit dieser Seriennummer verwenden
CalmWatchdog(long minutes) [PropPut]	- Minuten bis zum Alarm
ContactTimer[PropGet] returns int	- Abfrageinterval USB-Contact (1/100 Sekunden)
ContactTimer(long neu) [PropPut]	- Abfrageinterval USB-Contact (1/100 Sekunden)

Die Antwort der Methode “SwitchState” ist 0 im ausgeschalteten Zustand und 1 im eingeschalteten. Wird ein Wert von –1 zurückgegeben, konnte der Schalter nicht erreicht werden.

Um das Gerät USB-AutoReset, daß zwei Kontakte besitzt, mittels ActiveX schalten zu können, wurden mit Version 2.9 zwei weitere Methoden definiert:

SwitchXState(short switchID) [PropGet] returns int - der Zustand des Schalterkontaktes
SwitchXState(short switchID, long neu) [PropPut] - Schalterkontakt setzen

Der erste Kontakt hat die SwitchID 16, der zweite den Wert 17 und ggfs. der dritte den Wert 18 (siehe auch USBaccess.h, `enum SWITCH_Ids`). Um herauszufinden, wieviele Relais vorhanden sind, kann die folgende Methode verwendet werden:

SwitchConfig [PropGet] returns int - Zahl der gefundenen Relais und Taster

Der von SwitchConfig gelieferte Wert besteht aus 2 Teilen, nämlich der Zahl der Relais (multipliziert mit 256, bzw. << 8) und der Zahl der Taster (siehe USB-Plug).

Hier ein Beispiel zum Umschalten eines Schalters aus VisualBasic:

```
If (USBswitchAX1.SwitchState = 1) Then
    USBswitchAX1.SwitchState = 0
Else
    USBswitchAX1.SwitchState = 1
End If
```

Um ein zweites Relais einzuschalten, wäre der folgende Aufruf geeignet:

```
    USBswitchAX1.SwitchXState(17) = 1
```

Das ActiveX-Control kann auch direkt aus HTML aufgerufen werden. Hier eine sehr einfache Steuerung eine USB-Switch:

```
<HTML>
<HEAD>
<TITLE>USBswitch Control</TITLE>

</HEAD>
<BODY>
<CENTER>

<OBJECT
    CLASSID="clsid:27C9039A-A892-44C8-AD6A-F946801C4968"
    ID="USBswitchAX1"
    HEIGHT=200
    WIDTH=100
    >
</OBJECT>

<P>
<INPUT TYPE="BUTTON" NAME="cmdChange" VALUE="Switch On"
 OnClick="USBswitchAX1.SwitchState=1">

<INPUT TYPE="BUTTON" NAME="cmdChange" VALUE="Switch Off"
 OnClick="USBswitchAX1.SwitchState=0">

</CENTER>
</BODY>
</HTML>
```

Die Steuerung des USB-Watchdog wurde der Einfachheit halber durch die Integration in „USBswitchAX.ocx“ realisiert. Hierfür ist die Methode „CalmWatchdog“ entscheidend, die mit der Minutenangabe bis zum Auslösen des Alarms aufgerufen wird.

Auch das Gerät USB-Contact wird über das Control „USBswitchAX.ocx“ mit der Methode „SwitchState[PropGet]“ abgefragt. Zusätzlich wird vom Control ein Ereignis mit Namen „ContactChanged“ ausgelöst, wenn der Zustand des Kontaktes sich ändert. Das Ereignis wird von vier Argumenten begleitet, nämlich der Geratenummer, der Seriennummer des Gerätes, dem Status der Kontakte und einer Maske. Für Kontaktsensoren mit mehreren Kontakten ist der Status bitweise kodiert:

Kontakt Nr.	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Status	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Maske	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

Das Statusbit 0 zeigt an, daß der Kontakt offen ist und eine 1 steht für einen geschlossenen Kontakt. Um zu erkennen, welcher Kontakt sich geändert hat, werden die betreffenden Kontakte in der Maske m mit einer 1 markiert.

Der Kontakssensor wird regelmäßig abgefragt. Das Abfrageinterval ist standardmäßig auf 1/10 Sekunde eingestellt. Andere Abfrageintervalle lassen sich mit der Methode ContactTimer einstellen. Das Interval wird in 1/100stel Sekunden festgelegt.

Sind mehrere Geräte gleichzeitig über das Control zu bearbeiten, lassen sich diese durch die Seriennummer unterscheiden. Es gibt einen Zeiger auf das Gerät, das die Kommandos über ActiveX erhält. Dieser Zeiger läßt sich am einfachsten festlegen, indem der Methode „SerialNumber“ die Seriennummer des gewünschten Gerätes zugewiesen wird.

Hinweis zum Registrieren der ActiveX-Controls:

Vor dem Verwenden einer neuen Version des Controls muß das alte entfernt werden.

```
regsvr32 /u USBswitchAX.ocx
```

Das neue wird nach dem Einkopieren registriert.

```
regsvr32 USBswitchAX.ocx
```

2. 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 = 109 ;

#ifdef __cplusplus

class USBACCESS_API CUSBaccess {
  public:
    enum USBactions {  LEDs=0, EEwrite=1, EEread=2, Reset=3,
                       KeepCalm=4, GetInfo=5,
                       StartMeasuring=6   // USB-Humidity, USB-Contact
                    } ;
    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,
                          SWITCH1_DEVICE=0x08, SWITCH2_DEVICE=0x09,
                          SWITCH3_DEVICE=0x0a, SWITCH4_DEVICE=0x0c,
                          TEMPERATURE_DEVICE=0x10,
                          TEMPERATURE2_DEVICE=0x11,
                          TEMPERATURE5_DEVICE=0x15,
                          HUMIDITY1_DEVICE=0x20,
                          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) ;
      // try to find disconnected devices, return true if succeeded
    virtual int¹ GetValue(int deviceNo, unsigned char *buf,
                                        int bufsize) ;
    virtual int¹ SetValue(int deviceNo, 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 SetTempOffset(int deviceNo, double Sollwert,
                                             double Istwert) ;
```

```
    virtual int¹ GetTemperature(int deviceNo, double *Temperature,
                                              int *timeID) ;
    virtual int¹ GetHumidity(int deviceNo, double *Humidity,
                                               int *timeID) ;
    virtual int¹ ResetDevice(int deviceNo) ;
    virtual int¹ StartDevice(int deviceNo) ;
    virtual int¹ 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 int² GetManualOnCount(int deviceNo) ;
             // returns how often switch is manually turned on
    virtual int² GetManualOnTime(int deviceNo) ;
             // returns how long (seconds) switch is manually turned on
    virtual int² GetOnlineOnCount(int deviceNo) ;
             // returns how often switch is turned on by USB command
    virtual int² GetOnlineOnTime(int deviceNo) ;
      // returns how long (seconds) switch is turned on by USB command
    virtual int² GetMultiSwitch(int deviceNo, unsigned long int *mask,
                              unsigned long int *value, int seqNumber) ;
    virtual int² SetMultiSwitch(int deviceNo, unsigned long int value);
    virtual int² SetMultiConfig(int deviceNo, unsigned long int dir) ;
    virtual int² GetCounter(int deviceNo, int counter) ;
    virtual int² SyncDevice(int deviceNo, unsigned long int mask) ;
  } ;

extern "C" {
  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 };
enum FCWLED_IDs {     LED_0=0, LED_1=1, LED_2=2, LED_3=3 } ;
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
                          } ;

#endif // __cplusplus


// functional C interface (FCW = Function CleWare)
#ifdef __cplusplus
 extern "C" {
#endif // __cplusplus
  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) ;
…
#ifdef __cplusplus
   } ;
#endif // __cplusplus
```

```
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.

3. 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.

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 SetTempOffset(int deviceNo, double Sollwert, double Istwert) ;
int FCWSetTempOffset(CUSBaccess *obj, int deviceNo, double Sollwert, double Istwert) ;
Kalibration des Temperatursensors. Der Rückgabewert ist 0 im Fehlerfall, sonst > 0.

int GetTemperature(int deviceNo, double *Temperature, int *timeID) ;
int FCW GetTemperature (CUSBaccess *obj, int deviceNo, double *Temperature, int *timeID) ;

Abfrage der Temperatur des Sensors. 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 FCW Get Humidity (CUSBaccess *obj, int deviceNo, double * Humidity, int *timeID) ;

Abfrage des Feuchtigkeitmeßwertes des Sensors. 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.

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

Abfrage des Schalters USB-Switch. Das Argument „Switch“ bestimmt, welcher Schalter 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. 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 255 Minuten angegeben. Bei dem Gerät USB-AutoReset kann ein sofortiger Reset ausgelöst werden, wenn hier der Wert –1 übergeben wird.

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.

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

int GetDLLVersion() ;
int FCWGetDLLVersion(CUSBaccess *obj,) ;
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 GetMultiSwitch(int deviceNo, unsigned long int *mask,
unsigned long int *value, int seqNumber) ;
int FCWGetMultiSwitch(CUSBaccess *obj, int deviceNo,...);
Alle Eingänge der Geräte USB-Contact und 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 0 handelt es sich um einen Eingang, bei einer 1 um einenm Ausgang.

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

4. 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 ;
         int     zeit ;
         if (!CWusb.GetTemperature(devID, &temperatur, &zeit)) {
           printf("GetTemperature(%d) failed\n", devID) ;
           break ;
           }
         printf("Measured %lf degrees Celsius, time = %d\n",
                                                 temperatur, zeit) ;
         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

5. 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 ;
  }
```