

Application Interface for Windows®



Version 3.1.3
20.01.2005

Cleware GmbH
Nedderend 3
D-24876 Hollingstedt
Germany
www.cleware.de

Cleware USB devices API

Contents

- 1. The application interface in general3
- 2. USBaccess.h6
- 3. API Functions9
- 4. API C++ Example.....14
- 5. API C Example16

Cleware USB devices API

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 five 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.

1. USBaccess.h – Definition of the API
2. UBSaccess.lib – Link information
3. USBaccess.dll – Executables
4. USBswitchAX.ocx – ActiveX-Interface for the USB-Switch
5. USBtempAX.ocx – ActiveX-Interface for the USB-Temp

The Methods for controlling the temperature sensor with ActiveX are:

CountSensors [PropGet] returns int	- Number of found sensors
SensorNumber [PropGet] returns int	- Number of the current sensor
SensorNumber [PropPut] input int	- Selection of the sensor (1 ... sensor count)
Temperature [PropGet] returns double	- measured temperature of the current sensor
Humidity [PropGet] returns double	- measured humidity of the current sensor
SerialNumber [PropPut] returns int	- Serial number of the current sensor

If only one sensor is connected this one is automatically active and must not be selected. If the temperature returned by “Temperature” is -200. ° C this value is invalid and an error accrued while reading the sensor. In case of the method Humidity, an error is indicated by a negative humidity.

The Methods for controlling the switch with “USBswitchAX.ocx” are very similar and looks like this:

CountSwitches [PropGet] returns int	- Number of found switches and watchdogs
SwitchNumber [PropGet] returns int	- Number of the current switch
SwitchNumber (long number) [PropPut]	- Selection of the switch (1 ... switch count)
SwitchState [PropGet] returns int	- State of the switch
SwitchState(long neu) [PropPut]	- Set switch to state “neu”
SerialNumber [PropGet] returns int	- Serial number of the current switch
SerialNumber(long neu) [PropPut]	- Select device with this serial number
CalmWatchdog(long minutes) [PropPut]	- Minutes until alarm starts
ContactTimer[PropGet] returns int	- Testinterval USB-Contact 1/100 seconds
ContactTimer(long neu) [PropPut]	- Testinterval USB-Contact 1/100 seconds

The returned value from SwitchState may be “0” indicating “off” and “1” indicating “on” state. If the returned value is -1, an error occurred and the switch could not be reached.

The device USB-AutoReset has two independent contacts. To turning them two new methods were defined:

Cleware USB devices API

SwitchXState(short switchID) [PropGet] returns int - the state of the contact

SwitchXState(short switchID, long neu) [PropPut] - turning the contact

The first contact is reached with the SwitchID 16, the second with 17, etc. (see USBaccess.h, enum SWITCH_IDS). To get the number of available contacts in the device, the following method could be used:

SwitchConfig [PropGet] returns int - Returns the number of contacts and buttons

The returned value consist of two parts, the number of relais (contacts) and the number of buttons. The number of contacts are multiplied by 256 (or << 8) and added to the number of buttons.

This is a sample for turning the switch from VisualBasic:

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

To turn a second contacs, the following command would do:

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

The ActiveX control may also be used directly from HTML. Thos is a simple sample for turning the switch with two buttons:

```
<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>
```

Cleware USB devices API

For ease of use the method for controlling the USB-Watch (CalmWatchdog) is integrated in the "USBswitchAX.ocx". CalmWatchdog must be called with the minutes until starting the alarm.

The device USB-Contact is controlled also with "USBswitchAX.ocx" using the method "SwitchState[PropGet]". In addition to switches, an event called "ContactChanged" will be initiated by the control if the state of the USB-Contact changes. The event has four arguments (device number, serial number, status, mask). To support also contact sensors with multiple contacts, the state is coded bitwise in the status (0=open, 1=closed). To define, which of the contacts did change, the appropriate bit is set to 1 in the mask m.

Contact #	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
Mask	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m	m

The contact sensor will be scanned by an internal timer. The default time slice is 1/10 second. The time slice may be changed with the method ContactTimer. The interval is defined in 1/100 seconds.

To handle multiple devices with the ActiveX control, the devices are distinguished by their serial number. There is an internal pointer to the device that gets the commands. To set this pointer, the easiest way is to assign the serial number of the requested device with the method "SerialNumber".

Note:

Before using a new ActiveX control, the old one should be removed:

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

After copying the new one, it must be registered with

```
regsvr32 USBswitchAX.ocx
```

Cleware USB devices API

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.

```
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) ;

```

Cleware USB devices API

```
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) ;

```

Cleware USB devices API

```
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 = Returns TRUE if ok, FALSE in case of an error
2 = Return -1 in case of an error
```

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. 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.

Cleware USB devices API

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.

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.

Cleware USB devices API

```
int SetTempOffset(int deviceNo, double Sollwert, double Istwert) ;  
int FCWSetTempOffset(CUSBaccess *obj, int deviceNo, double Sollwert,  
double Istwert) ;
```

Calibration of a temperature device. The second parameter “Sollwert” is the correct temperature and the “Istwert” is the current value to be corrected. The return value is 0 in case of an error, else > 0.

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

Request for the current temperature. 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 FCW Get Humidity (CUSBaccess *obj, int deviceNo, double * Humidity,  
int *timeID) ;
```

Request for the current humidity. 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.

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

Cleware USB devices API

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). 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(CUSBaccess *obj,) ;

Get the version of the DLL.

Cleware USB devices API

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 SyncDevice(int deviceNo, unsigned long int mask) ;

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

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 be 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 assumed and this one is turned on or off. This is useful to control the switch with cmdline commands.

```
#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 &&
```

Cleware USB devices API

```
        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;
}
```

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”.

Cleware USB devices API

5. API C Example

```
// 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':
```

Cleware USB devices API

```
        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 ;
}

}
```