



Cleware USB devices with Linux

Version 3.1.3
21.01.2005

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

Linux and Cleware USB devices

1.General

All Cleware devices will run with SuSE Linux 8.0 (kernel 2.4.18), SuSE Linux 9.0 (2.4.21) and SuSE Linux 9.2 (2.6.8-24). Earlier versions of Linux-Versionen need some kernel extensions to talk to the Cleware devices. The SuSE versions 8.1 and 8.2 shows some problems with USB HID devices. It is recommended to to upgrade to version 9.2.

The devices will be accessed via the library “USBaccess.a” and the corresponding headerfile “USBaccess.h”. The functions are identical to the functions supplied in the Windows DLL. Programs will run on both systems after recompilation. Only small changes in the include section are necessary.

“SuSE9.2 (Kernel 2.6.8-24).zip” contains all files needed for working with Linux. The sources of the interface and samples are included.

Before using the Cleware devices for the first time, some device names (hiddev0, hiddev1, etc) must be created in /dev/usb. They are not created by default when installing Linux. The devices may created using the script “make_hid” supplied in the zip file. As user root just call “source make_hid” and the devices are created. This must be done only one time.

The Linux delivery contains the interface and the sample programs “Example.cpp“, “USBswitch“ and “USBwatch“. The Cleware devices connected to a Linux system may act as remote devices with the application “send2cc” described in chapter 5. The measured values will be send to a Windows PC running ClewareControl in server mode.

Linux and Cleware USB devices

2.USBAccess.h

The file USBAccess.h contains the interface to the Cleware USB devices. After including it, the devices could be opened and accessed. The description of the definitions and methods are identical to the windows API functions. Consequently the description of the methods could be found in the Cleware Windows API document.

```
// DLL class definitions for access to USB HID devices
//
// (C) 2004 Copyright Cleware GmbH
// All rights reserved
//
.....

typedef int HANDLE ;
const int USBaccessVersion = 109 ;

class USBACCESS_API CUSBaccess {
public:
    enum USBactions {    LEDs=0, EEwrite=1, EEread=2, Reset=3 } ;

    enum LED_IDs {      LED_0=0, LED_1=1, LED_2=2, LED_3=3 } ;

    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,
                           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
    } ;
};
```

Linux and Cleware USB devices

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

    int      OpenCleware() ; // returns number of Cleware devices
    int      CloseCleware() ; // close all Cleware devices
    int      Recover(int devNum) ; // try to find disconnected devices,
                                   // returns true if succeeded
    HANDLE   GetHandle(int deviceNo) ;
    int1    GetValue(int deviceNo, unsigned char *buf, int bufsize) ;
    int1    SetValue(int deviceNo, unsigned char *buf, int bufsize) ;
    int1    SetLED(int deviceNo, enum LED_IDS Led, int value) ;
                                   // value: 0=off 7=medium 15=highlight
    int1    SetSwitch(int deviceNo, enum SWITCH_IDS Switch, int On) ;
                                   // On: 0=off, 1=on
    int2    GetSwitch(int deviceNo, enum SWITCH_IDS Switch) ;
    int2    GetSwitchConfig(int deviceNo, int *switchCnt,
                            int *buttonAvailable) ;
    int1    SetTempOffset(int deviceNo, double Soll, double Ist) ;
    int1    GetTemperature(int deviceNo, double *Temperature, int
*time) ;
    int1    GetHumidity(int deviceNo, double *Humidity, int *timeID) ;
    int1    ResetDevice(int deviceNo) ;
    int1    StartDevice(int deviceNo) ;
    int     GetVersion(int deviceNo) ;
    int     GetUSBType(int deviceNo) ;
    int     GetSerialNumber(int deviceNo) ;
    int     GetDLLVersion() { return USBAccessVersion ; }
    int2    GetManualOnCount(int deviceNo) ;
                                   // returns how often switch is manually turned on
    int2    GetManualOnTime(int deviceNo) ;
                                   // returns how long (seconds) switch is manually turned
on
    int2    GetOnlineOnCount(int deviceNo) ;
                                   // returns how often switch is turned on by USB command
    int2    GetOnlineOnTime(int deviceNo) ; // returns how long
(seconds)
                                   // switch is turned on by USB
                                   command
} ;
```

¹ = Returns TRUE if ok, FALSE icase of an error

² = Returns -1 in case of an error

3.API samples

The following simple sample demonstrates the usage of the API to access the temperature of an USB-Temp and to turn the state of an USB-Switch. The programs are written in C++. If this program will be called without an argument, an USB-Temp will be searched. If one is found, the temperature will be caught and printed 10 times. When an argument is supplied while starting the sample, this argument will be used as the state the USB-Switch should take (0=off, 1=on). An extended version of this sample is part of the CD.

```
...
#include "USBaccess.h"

int main(int argc, char* argv[]) {
    CUSBaccess CWusb ;
    printf("Start USB Access sample!\n") ;

    int USBcount = CWusb.OpenCleware() ;
    printf("OpenCleware found %d devices\n", USBcount) ;

    for (int devID=0 ; devID < USBcount ; devID++) {
        if (argc == 2) { // only USB-Switch
            if (CWusb.GetUSBType(devID) == CUSBaccess::SWITCH1_DEVICE) {
                if (argv[1][0] == '0')
                    CWusb.SetSwitch(devID, CUSBaccess::SWITCH_0, 0) ;
                else if (argv[1][0] == '1')
                    CWusb.SetSwitch(devID, CUSBaccess::SWITCH_0, 1) ;
                else
                    printf("Argument is wrong\n") ;
                break ; // only the first switch found
            }
            else
                continue ; // don't care about the others
        }

        if ( CWusb.GetUSBType(devID) != CUSBaccess::TEMPERATURE_DEVICE &&
            CWusb.GetUSBType(devID) != CUSBaccess::TEMPERATURE2_DEVICE)
            continue ; // only USB-Temp!

        CWusb.ResetDevice(devID) ; // reset the device (only once)
        Sleep(500) ; // wait a bit

        // 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("Temperature is %lf degree Celsius, time = %d\n",
                temperatur, zeit) ;

            Sleep(1200) ;
        }
    }
    CWusb.CloseCleware() ;

    return 0;
}
```

4. Application „send2cc“

The program “send2cc“ is a Linux program that sends informations about Cleware USB devices to the graphical user interface ClewareControl running under Windows. All Cleware temperature and humidity sensors are supported. The sensors are scanned in a 2 seconds interval to get the measured values.

Several options are available:

-s servername	specify the ClewareControl server
-p	specify the port (default 54741)
-r	repeat action in case of an error
-d	print debug info
-v	print version number
-h	show options

The server name option must be supplied anyway, the other options are optional. The server name could be supplied as the name of the server or as the IP number.

In case of an error while connecting or talking to the server the application normally exits. This could be avoided by using the option `-r`. In this case, the action will be retried in an endless loop after waiting 10 seconds.

If the program should be running as a background process with SuSE Linux, the following steps could be used to implement a service:

- login as root
- Copy send2cc to /usr/sbin
- `chmod 755 /usr/sbin/send2cc`
- Copy script rcSend2cc to /etc/rc.d
- `chmod 755 /etc/rc.d/rcSend2cc`
- `cd /etc/rc.d/rc5.d` (and rc3.d ... if applicable)
- `ln -s ../rcSend2cc S20send2cc`
- `ln -s ../rcSend2cc K20send2cc`

The name of the server must be entered in the script “rcSend2cc“. The corresponding line is “CCSERVERNAME=XXXX“. The text XXXX must be replaced by the correct name or IP address. To test the program call “rcSend2cc start” in /etc/rc.d. The application ClewareControl should show the remote sensors. For setting up ClewareControl to act as a server please refer to the ClewareControl manual.

5. Programm “readtemp”

In contrast to the “Example” described above the program “readtemp” is able to scan all connected humidity and temperature devices and display their measured values.

6. Program “USBswitch”

A little program “USBswitch” control an USB-Switch and handles more than one device. The following options are supported:

```
USBswitch [-n device] [0 | 1] [-d]
  -n device  use device with this serial number
  0 | 1      turns switch off(0) or on(1)
  -d         print debug infos
  -s         secure switching - wait and ask if switch was turned
  -r         read the current setting
  -v         print version
  -h         print command usage
```

7. Program “USBwatch”

The program “USBwatch“ send a life signal to an USB-AutoReset or USB-Watchdog every 2 seconds.