

```

// Buttons and Lights.CPP -- Visual C++ version
//
// Copyright (C) 2001, Intel Corporation
// All rights reserved.
// Permission is hereby granted to merge this program code with other program
// material to create a derivative work. This derivative work may be distributed
// in compiled object form only. Any other publication of this program, in any form,
// without the explicit permission of the copyright holder is prohibited.
//
// Send questions and comments to John.Hyde@intel.com

#include "stdafx.h"
#include "objbase.h"
#include <iostream.h>

extern "C" {
// Declare the C libraries used
#include "setupapi.h"
#include "hidsdi.h"
}

HANDLE Handle;

bool OpenUSBinterface() {
    struct _GUID GUID;
    SP_INTERFACE_DEVICE_DATA DeviceInterfaceData;
    struct {DWORD cbSize; char DevicePath[256];} FunctionClassDeviceData;
    int Device, i;
    HANDLE PnPHandle;
    char buffer[256];
    unsigned long BytesReturned;
    bool Success, Openned;
    const char *DeviceName = "Buttons & Lights";
    SECURITY_ATTRIBUTES SecurityAttributes;

// Initialize the GUID array and setup the security attributes for Win2000
    HidD_GetHidGuid(&GUID);
    SecurityAttributes.nLength = sizeof(SECURITY_ATTRIBUTES);
    SecurityAttributes.lpSecurityDescriptor = NULL;
    SecurityAttributes.bInheritHandle = false;

// Get a handle for the Plug and Play node and request currently active devices
    PnPHandle = SetupDiGetClassDevs(&GUID, NULL, NULL, DIGCF_PRESENT|DIGCF_INTERFACEDevice);
    if (int(PnPHandle) == -1) { printf("Could not attach to PnP node"); return false; }

// Lets look for a maximum of 20 Devices
    Openned = false;
    Handle = INVALID_HANDLE_VALUE;
    for (Device = 0; (Device < 20); Device++) {

// Initialize our data
        DeviceInterfaceData.cbSize = sizeof(DeviceInterfaceData);

// Is there a device at this table entry
        Success = SetupDiEnumDeviceInterfaces(PnPHandle, NULL, &GUID, Device, &DeviceInterfaceData);
        if (Success) {

// There is a device here, get it's name
            FunctionClassDeviceData.cbSize = 5;
            Success = SetupDiGetDeviceInterfaceDetail(PnPHandle, &DeviceInterfaceData,
                (PSP_INTERFACE_DEVICE_DETAIL_DATA)&FunctionClassDeviceData, 256, &BytesReturned, NULL);
            if (!Success) { printf("Could not find the system name for this Device\n"); return false; }

// Can now open this Device
            Handle = CreateFile(FunctionClassDeviceData.DevicePath, GENERIC_READ|GENERIC_WRITE,
                FILE_SHARE_READ|FILE_SHARE_WRITE, &SecurityAttributes, OPEN_EXISTING, 0, NULL);
            if (Handle == INVALID_HANDLE_VALUE) printf("Could not open HID device\n");
            else {

// Is it OUR Device?
                HidD_GetProductString(Handle, buffer, sizeof(buffer));

// Compare incoming string with UNICODE string
                Openned = true; i = 0;
                while (DeviceName[i] != 0) { if (buffer[2*i] != DeviceName[i]) {Openned = false;} i++;}
                if (Openned) break;
            }
        } // if (SetupDiEnumDeviceInterfaces . .
    } // for (Device = 0; (Device < 20); Device++)
}

```

```

    SetupDiDestroyDeviceInfoList(PnPHandle);
    return Openned;
}

int main(int argc, char* argv[]) {
    char InputCharacter;
    int Choice, LEDvalue, i;
    unsigned long BytesWritten, BytesRead;
    UCHAR ReplyBuffer[2];
    UCHAR LightsCommand[2] = {0, 0};

    // Say Hello to the user
    printf("\nVisual C++ version of 'Buttons and Lights'\n\n");

    // Search for and open our HID device
    if (!OpenUSBinterface()) {
        printf("Buttons and Lights device was not found\n");
        cin >> InputCharacter;
        return -1;
    }

    // A simple, prompted interface. Needs improvement!!
    do {
        printf("\nThe following commands are available:\n");
        printf("B = Read the device 'Buttons'\n");
        printf("L = Set the device 'Lights'\n");
        printf("E = Exit this program\n");
        printf("Please enter your choice: ");
        cin >> InputCharacter;
        Choice = InputCharacter;
        switch (tolower(Choice)) {
            case 'b':
                if (!ReadFile(Handle, ReplyBuffer, sizeof(ReplyBuffer), &BytesRead, NULL))
                    printf("Error reading from HID device\n");
                printf("\nButtons value = ");
                for (i=0; i<8; i++) printf("%s", ((ReplyBuffer[1] >> i) & 1) ? "0" : "1");
                printf("\n");
                break;
            case 'l':
                printf("\nValue to be written to Lights: ");
                cin >> LEDvalue;
                LightsCommand[1] = LEDvalue;
                if (!WriteFile(Handle, LightsCommand, sizeof(LightsCommand), &BytesWritten, NULL))
                    printf("Error writing Command to I/O device\n");
                break;
            case 'e':
                printf("\n\n");
                CloseHandle(Handle);
                return 0;
            default: printf("\nInvalid Selection\n"); break;
        }
    } while (1);
} // Main

```