

```

// TEST.CPP -- test program for BlockIO.SYS
//
// 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 "winioctl.h"
#include "objbase.h"
#include <initguid.h>
#include "guid.h"
#include <iostream.h>

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

#define IOCTL_GET_PRODUCT_NAME CTL_CODE(FILE_DEVICE_UNKNOWN, 0x800, METHOD_BUFFERED, FILE_ANY_ACCESS)

// have a few global variables
HANDLE Handle[2], HID, BlockIO;

bool OpenUSBinterfaces() {
    struct _GUID GUID[2];
    SP_INTERFACE_DEVICE_DATA DeviceInterfaceData;
    struct {DWORD cbSize; char DevicePath[256];} FunctionClassDeviceData;
    int Device, Interface, i;
    HANDLE PnPHandle;
    char buffer[256];
    unsigned long BytesReturned;
    bool Success, Openned;
    const char *DeviceName = "ECHO1";
    SECURITY_ATTRIBUTES SecurityAttributes;

// Initialize the GUID arrays and setup the security attributes for Win2000
    HidD_GetHidGuid(&GUID[0]);
    memcpy(&GUID[1], &BlockIO_GUID, sizeof(BlockIO_GUID));
    SecurityAttributes.nLength = sizeof(SECURITY_ATTRIBUTES);
    SecurityAttributes.lpSecurityDescriptor = NULL;
    SecurityAttributes.bInheritHandle = false;

// Our device has two interfaces, 0 = HID and 1 = BlockIO. Open them both
    BytesReturned = IOCTL_GET_PRODUCT_NAME;
    for (Interface = 0; Interface<2; Interface++) {

// Get a handle for the Plug and Play node and request currently active devices
        PnPHandle = SetupDiGetClassDevs(&GUID[Interface], 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
        Handle[Interface] = 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[Interface], 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[Interface] = CreateFile(FunctionClassDeviceData.DevicePath, GENERIC_READ|GENERIC_WRITE,
                    FILE_SHARE_READ|FILE_SHARE_WRITE, &SecurityAttributes, OPEN_EXISTING, 0, NULL);
                if (Handle[Interface] == INVALID_HANDLE_VALUE) printf("Could not open %s device\n", Interface ?
"BlockIO" : "HID");
                else {

```

```

// Is it OUR Device?
    if (Interface == 0) HidD_GetProductString(Handle[Interface], buffer, sizeof(buffer));
    else DeviceIoControl(Handle[Interface], IOCTL_GET_PRODUCT_NAME, NULL, 0, buffer, sizeof(buffer),
&BytesReturned, NULL);

// Compare incoming string with UNICODE string
    Openned = true; i = 0;
    while (DeviceName[i] != 0) { if (buffer[2*i] != DeviceName[i]) {Openned = false;} i++;}
    printf("%s Interface %sfound\n", Interface ? "BlockIO" : "HID", Openned ? " " : "not ");
    }
} // if (SetupDiEnumDeviceInterfaces . .
} // for (Device = 0; (Device < 20); Device++)
SetupDiDestroyDeviceInfoList(PnpHandle);
return true;
}

int main(int argc, char* argv[]) {
char InputCharacter;
int Choice, BufferSize, LEDvalue, i;
unsigned long BytesWritten, BytesRead;
UCHAR DataBuffer[0x3FFF]; // Maximum size for 'ECHO1' firmware
UCHAR ButtonsCommand[4] = {0,1,0,0};
UCHAR LightsCommand[4] = {0,2,0,0};
UCHAR SendCommand[4] = {0,3,0,0};
UCHAR ReadCommand[4] = {0,4,0,0};
UCHAR ReplyBuffer[2];

printf("\nTest program for 'ECHO1' HIDwithBlockIO Example\n\n");
if (!OpenUSBInterfaces()) {
    printf("\nERROR EXIT\n");
    cin >> InputCharacter;
    return -1;
}

HID = Handle[0]; BlockIO = Handle[1]; // Make program easier to read
if ((HID == INVALID_HANDLE_VALUE) || (BlockIO == INVALID_HANDLE_VALUE)) {
    CloseHandle(HID);
    CloseHandle(BlockIO);
    printf("\nNeed both interfaces open for the 'ECHO1' device\n");
    printf("\nERROR EXIT\n");
    cin >> InputCharacter;
    return -1;
}

// Put something recognizable into the data buffer
for (i=0; i<sizeof(DataBuffer); i++) DataBuffer[i] = (UCHAR) i;
BufferSize = 0x3FFF;

do {
    printf("\nThe following commands are available:\n");
    printf("B = Read the device 'Buttons'\n");
    printf("L = Set the device 'Lights'\n");
    printf("C = Change the data buffer size (%d)\n", BufferSize);
    printf("S = Send the data buffer\n");
    printf("R = Receive the data buffer\n");
    printf("E = Exit this program\n");
    printf("Please enter your choice - ");
    cin >> InputCharacter;
    Choice = InputCharacter;
    switch (tolower(Choice)) {
        case 'b':
            if (!WriteFile(HID, ButtonsCommand, sizeof(ButtonsCommand), &BytesWritten, NULL))
                printf("Error writing Command to I/O device\n");
            if (!ReadFile(HID, 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[2] = LEDvalue;
            if (!WriteFile(HID, LightsCommand, sizeof(LightsCommand), &BytesWritten, NULL))
                printf("Error writing Command to I/O device\n");
            break;
        case 'c':

```

```

printf("\nEnter new value for buffer size: ");
cin >> BufferSize;
// if ((BufferSize%64) == 0) BufferSize++; // Get around a driver bug. Will be fixed
if (BufferSize > 0x3FFF) {
    BufferSize = 0x3FFF;
    printf("Maximum buffer size for 'ECHO1' device is %d (%xH)\n", 0x3FFF, 0x3FFF);
}
break;
case 's':
printf("\nSending the data buffer\n");
if (!WriteFile(HID, SendCommand, sizeof(SendCommand), &BytesWritten, NULL))
    printf("Error writing Command to I/O device\n");
if (WriteFile(BlockIO, DataBuffer, BufferSize, &BytesWritten, NULL))
    printf("Write transferred %d bytes okay\n", BytesWritten);
else printf("Error %d trying to write BlockIO data\n", GetLastError());
break;
case 'r':
printf("\nReceiving the data buffer\n");
ReadCommand[2] = BufferSize & 0xFF; ReadCommand[3] = BufferSize >> 8;
if (!WriteFile(HID, ReadCommand, sizeof(ReadCommand), &BytesWritten, NULL))
    printf("Error writing Command to I/O device\n");
if (ReadFile(BlockIO, DataBuffer, BufferSize, &BytesRead, NULL))
    printf("Read transferred %d bytes correctly\n", BytesRead);
else printf("Error %d trying to read BlockIO data\n", GetLastError());
break;
case 'e':
printf("\n\n");
CloseHandle(HID);
CloseHandle(BlockIO);
return 0;
default: printf("\nInvalid Selection\n"); break;
}
} while (1);
} // Main

```