

```

// DisplayUSB.cpp: Defines the entry point for the console application.
//
// 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 "winioctl.h"
#include "usbioctl.h"
#include "stdio.h"
#include "stdlib.h"

bool DEBUG; // Set to true to enable DEBUG messages
SECURITY_ATTRIBUTES SA; // Needed for Win2000

const char ClassName[] [20] = {
    "Reserved",          "Audio",          "Communications",    "Human Interface",
    "Monitor",          "Physical Interface", "Power",            "Printer",
    "Storage",          "Hub",            "Vendor Specific",   "**ILLEGAL VALUE**"
};

const char ConnectionStatus[] [30] = {
    "No device connected", "Device connected", "Device FAILED enumeration",
    "Device general FAILURE", "Device caused overcurrent", "Not enough power for device"
};

// Define all structures using UCHAR or BOOLEAN so that the variables are not 'aligned' by the compiler
typedef struct DESCRIPTOR_REQUEST {
    ULONG ConnectionIndex;
    struct {UCHAR bmRequest; UCHAR bRequest; UCHAR wValue[2]; UCHAR wIndex[2]; UCHAR wLength[2];} SetupPacket;
    UCHAR Data[2048];
};

typedef struct DEVICE_DESCRIPTOR {
    UCHAR bLength;          UCHAR bDescriptorType;          UCHAR bcdUSB[2];
    UCHAR bDeviceClass;    UCHAR bDeviceSubClass;          UCHAR bDeviceProtocol;
    UCHAR bMaxPacketSize0; UCHAR idVendor[2];              UCHAR idProduct[2];
    UCHAR bcdDevice[2];    UCHAR iManufacturer;           UCHAR iProduct;
    UCHAR iSerialNumber;   UCHAR bNumConfigurations;
};

typedef struct HUB_DESCRIPTOR {
    UCHAR bDescriptorLength;  UCHAR bDescriptorType;          UCHAR bNumberOfPorts;
    UCHAR wHubCharacteristics[2];  UCHAR bPowerOnToPowerGood;     UCHAR bHubControlCurrent;
    UCHAR bRemoveAndPowerMask[64];
};

typedef struct NODE_INFORMATION {
    USB_HUB_NODE NodeType;      HUB_DESCRIPTOR HubDescriptor;   BOOLEAN HubIsBusPowered;
};

typedef struct NODE_CONNECTION_INFORMATION {
    ULONG ConnectionIndex;      DEVICE_DESCRIPTOR DeviceDescriptor;  UCHAR CurrentConfigurationValue;
    BOOLEAN LowSpeed;          BOOLEAN DeviceIsHub;             UCHAR DeviceAddress[2];
    UCHAR NumberOfOpenPipes[4];  UCHAR ConnectionStatus[4];       USB_PIPE_INFO PipeList[32];
};

USHORT DisplayStringDescriptor (HANDLE HubHandle, ULONG PortIndex, USHORT LanguageID, UCHAR Index) {
    if (DEBUG) printf("\nIn DisplayStringDescriptor with HubHandle = %x, PortIndex = %x, LanguageID = %x, Index = %x\n",
        HubHandle, PortIndex, LanguageID, Index);
    DESCRIPTOR_REQUEST Packet;
    DWORD BytesReturned;
    bool Success;
    if (LanguageID == 0) { // Get the language ID
        memset(&Packet, 0, sizeof(Packet));
        Packet.ConnectionIndex = PortIndex;
        Packet.SetupPacket.bmRequest = 0x80;
        Packet.SetupPacket.bRequest = USB_REQUEST_GET_DESCRIPTOR;
        Packet.SetupPacket.wValue[1] = USB_STRING_DESCRIPTOR_TYPE;
        Packet.SetupPacket.wLength[0] = 4;
        Success = DeviceIoControl(HubHandle, IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, &Packet,

```

```

        sizeof(Packet), &Packet, sizeof(Packet), &BytesReturned, NULL);
    if (!Success) printf(" *** ERROR *** String Descriptor 0 not returned, ErrorCode = %d\n",
GetLastError());
    LanguageID = Packet.Data[2] + (Packet.Data[3] << 8);
}
memset(&Packet, 0, sizeof(Packet));
Packet.ConnectionIndex = PortIndex;
Packet.SetupPacket.bmRequest = 0x80;
Packet.SetupPacket.bRequest = USB_REQUEST_GET_DESCRIPTOR;
Packet.SetupPacket.wValue[1] = USB_STRING_DESCRIPTOR_TYPE;
Packet.SetupPacket.wValue[0] = Index;
Packet.SetupPacket.wIndex[0] = LanguageID & 0xFF;
Packet.SetupPacket.wIndex[1] = (LanguageID >> 8) & 0xFF;
Packet.SetupPacket.wLength[0] = 255;
Success = DeviceIoControl(HubHandle, IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, &Packet,
    sizeof(Packet), &Packet, sizeof(Packet), &BytesReturned, NULL);
if (!Success) printf(" *** ERROR *** String Descriptor %d not returned. ErrorCode = %d\n", Index,
GetLastError());
printf(" = %ws", &Packet.Data[2]);
return LanguageID;
}

```

```

USHORT DisplayDeviceDescriptor (HANDLE HubHandle, ULONG PortIndex, USHORT LanguageID, PCHAR BufferPtr) {
    if (DEBUG) printf("In DisplayDeviceDescriptor with HubHandle = %x, PortIndex = %x, LanguageID = %x\n",
HubHandle, PortIndex, LanguageID);
    UCHAR LowByte;
    printf("Device Descriptor");
    BufferPtr--; // Backup pointer to prepare for pre-increment
    printf("\n bLength          %2.2x", *++BufferPtr);
    printf("\n bDescriptorType      %2.2x", *++BufferPtr);
    LowByte = *++BufferPtr;
    printf("\n bcdUSB              %4.4x", LowByte + (*++BufferPtr << 8));
    printf("\n bDeviceClass         %2.2x", *++BufferPtr);
    printf("\n bDeviceSubClass      %2.2x", *++BufferPtr);
    printf("\n bDeviceProtocol      %2.2x", *++BufferPtr);
    printf("\n bMaxEP0Size          %2.2x", *++BufferPtr);
    LowByte = *++BufferPtr;
    printf("\n wVendorID            %4.4x", LowByte + (*++BufferPtr << 8));
    LowByte = *++BufferPtr;
    printf("\n wProductID           %4.4x", LowByte + (*++BufferPtr << 8));
    LowByte = *++BufferPtr;
    printf("\n wDeviceID            %4.4x", LowByte + (*++BufferPtr << 8));
    printf("\n iManufacturer        %2.2x", *++BufferPtr);
    if (*BufferPtr != 0) LanguageID = DisplayStringDescriptor(HubHandle, PortIndex, LanguageID, *BufferPtr);
    printf("\n iProduct             %2.2x", *++BufferPtr);
    if (*BufferPtr != 0) LanguageID = DisplayStringDescriptor(HubHandle, PortIndex, LanguageID, *BufferPtr);
    printf("\n iSerialNumber        %2.2x", *++BufferPtr);
    if (*BufferPtr != 0) LanguageID = DisplayStringDescriptor(HubHandle, PortIndex, LanguageID, *BufferPtr);
    printf("\n bNumConfigurations  %2.2x\n", *++BufferPtr);
    return LanguageID;
}

```

```

USHORT DisplayConfigurationDescriptor(HANDLE HubHandle, ULONG PortIndex, USHORT LanguageID) {
    if (DEBUG) printf("In DisplayConfigurationDescriptor with HubHandle = %x, PortIndex = %x, LanguageID =
%x\n", HubHandle, PortIndex, LanguageID);
    DWORD BytesReturned;
    bool Success;
    UCHAR LowByte;
    DESCRIPTOR_REQUEST Packet;
    int i;
    printf("\nConfiguration Descriptor");
    // First need to get the configuration descriptor
    memset(&Packet, 0, sizeof(Packet));
    Packet.ConnectionIndex = PortIndex;
    Packet.SetupPacket.bmRequest = 0x80;
    Packet.SetupPacket.bRequest = USB_REQUEST_GET_DESCRIPTOR;
    Packet.SetupPacket.wValue[1] = USB_CONFIGURATION_DESCRIPTOR_TYPE;
    Packet.SetupPacket.wLength[1] = 1; // Using a 2K buffer
    Success = DeviceIoControl(HubHandle, IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, &Packet,
        sizeof(Packet), &Packet, sizeof(Packet), &BytesReturned, NULL);
    if (!Success) printf(" *** ERROR *** Configuration Descriptor not returned. ErrorCode = %d\n",
GetLastError());
    PCHAR BufferPtr = &Packet.Data[0];
    UCHAR Length = *BufferPtr;
    while (Length != 0) {
        UCHAR Type = *++BufferPtr;
        switch (Type) {
            case 2:
                printf("\n bLength          %2.2x", Length);

```

```

printf("\n bDescriptorType      %2.2x = Configuration Header", Type);
LowByte = *++BufferPtr;
printf("\n wTotalLength        %4.4x", LowByte + (*++BufferPtr << 8));
printf("\n bNumInterfaces          %2.2x", *++BufferPtr);
printf("\n bConfigValue              %2.2x", *++BufferPtr);
printf("\n iConfiguration            %2.2x", *++BufferPtr);
if (*BufferPtr != 0) LanguageID = DisplayStringDescriptor(HubHandle, PortIndex, LanguageID,
*BufferPtr);
printf("\n bmAttributes              %2.2x", *++BufferPtr);
LowByte = *++BufferPtr;
printf("\n bMaxPower                 %2.2x = %d mA", LowByte, (LowByte << 1));
break;
case 4:
printf("\n bLength                    %2.2x", Length);
printf("\n bDescriptorType            %2.2x = Interface Descriptor", Type);
printf("\n bInterfaceNum              %2.2x", *++BufferPtr);
printf("\n bAlternateSetting          %2.2x", *++BufferPtr);
printf("\n bNumEndpoints              %2.2x", *++BufferPtr);
LowByte = *++BufferPtr;
if ((LowByte > 9) & (LowByte < 255)) LowByte = 11;
if (LowByte == 255) LowByte = 10;
printf("\n bInterfaceClass            %2.2x = %s", *BufferPtr, ClassName[LowByte]);
printf("\n bSubClass                  %2.2x", *++BufferPtr);
printf("\n bProtocol                  %2.2x", *++BufferPtr);
printf("\n iInterface                 %2.2x", *++BufferPtr);
if (*BufferPtr != 0) LanguageID = DisplayStringDescriptor(HubHandle, PortIndex, LanguageID,
*BufferPtr);
break;
case 5:
printf("\n bLength                    %2.2x", Length);
printf("\n bDescriptorType            %2.2x = Endpoint Descriptor", Type);
printf("\n bEndpointAddress           %2.2x", *++BufferPtr);
printf("\n bmAttributes                %2.2x", *++BufferPtr);
LowByte = *++BufferPtr;
printf("\n wMaxPacketSize             %4.4x", LowByte + (*++BufferPtr << 8));
printf("\n bInterval                  %2.2x", *++BufferPtr);
break;
case 0x21:
printf("\n bLength                    %2.2x", Length);
printf("\n bDescriptorType            %2.2x = HID Descriptor", Type);
LowByte = *++BufferPtr;
printf("\n wHIDVersion                %4.4x", LowByte + (*++BufferPtr << 8));
printf("\n bCountryCode                %2.2x", *++BufferPtr);
printf("\n bHIDDescriptorCount        %2.2x", *++BufferPtr);
printf("\n bHIDReportType              %2.2x", *++BufferPtr);
LowByte = *++BufferPtr;
printf("\n wHIDReportLength           %4.4x", LowByte + (*++BufferPtr << 8));
break;
default:
printf("\nUnknown descriptor with Length = %2.2xH and Type = %2.2xH", Length, Type);
BufferPtr-=2; // Back up to start of descriptor
for (i = 0; i < Length; i++) {
    if ((i % 16) == 0) printf("\n");
    printf("%2.2x ", *++BufferPtr);
}
break;
}
Length = *++BufferPtr;
printf("\n");
}
return LanguageID;
}

```

```

void GetPortData(HANDLE HubHandle, UCHAR PortCount, int HubDepth) {
    if (DEBUG) printf("In GetPortData with HubHandle = %x, PortCount = %x, HubDepth = %x\n", HubHandle,
PortCount, HubDepth);
    DWORD BytesReturned;
    bool Success;
    int i;
    ULONG PortIndex;
    USHORT LanguageID;
    UCHAR ThisDevice, PortStatus;
    char ConnectedHubName[256] = "\\.\\";
    HANDLE ConnectedHubHandle;
    NODE_INFORMATION NodeInformation;
    NODE_CONNECTION_INFORMATION ConnectionInformation;
    struct {ULONG ConnectionIndex; ULONG ActualLength; WCHAR Name[256];} ConnectedHub;

```

```
// Iterate over the ports to discover what is connected to each one
```

```

for (PortIndex = 1; PortIndex < (ULONG)PortCount + 1; PortIndex++) {
    LanguageID = 0; // Reset for each port
    ConnectionInformation.ConnectionIndex = PortIndex;
    Success = DeviceIoControl(HubHandle, IOCTL_USB_GET_NODE_CONNECTION_INFORMATION, &ConnectionInformation,
        sizeof(ConnectionInformation), &ConnectionInformation, sizeof(ConnectionInformation), &BytesReturned,
NULL);
    if (!Success) printf(" *** ERROR *** Node connection information not returned\n");

    PortStatus = ConnectionInformation.ConnectionStatus[0]; // Save some typing!
    ThisDevice = (PortStatus == DeviceConnected) ? ConnectionInformation.DeviceAddress[0] : 0;

// Create an indented display so that hubs and their connections are more easily seen
// First the common header
// printf("%2.2x", ThisDevice);
for (i=0; i<HubDepth; i++) printf("+1");
printf("          Port[%d] = ", PortIndex);

// Now the connection specific information
if (PortStatus != DeviceConnected) printf("%s\n", ConnectionStatus[PortStatus]);
else { // have a device or a hub connected to this port
    if (!ConnectionInformation.DeviceIsHub) {

// There is an I/O device connected. Print out it's descriptors
// Note that many current devices do not respond correctly if ConfigID != 0. So only request the first
configuration
        printf("I/O device connected\n");
        LanguageID = DisplayDeviceDescriptor(HubHandle, PortIndex, LanguageID,
&ConnectionInformation.DeviceDescriptor.bLength);
        LanguageID = DisplayConfigurationDescriptor(HubHandle, PortIndex, LanguageID);
    }
    else {

// There is a hub connected and we need to iterate over it's ports
printf("Hub connected\n");

// Get the system name of the connected hub so that we can make a connection to it
        ConnectedHub.ConnectionIndex = PortIndex;
        Success = DeviceIoControl(HubHandle, IOCTL_USB_GET_NODE_CONNECTION_NAME, &ConnectedHub,
            sizeof(ConnectedHub), &ConnectedHub, sizeof(ConnectedHub), &BytesReturned, NULL);
        if (!Success) printf(" *** ERROR *** Node connection name not returned\n");
        WideCharToMultiByte(CP_ACP, 0, &ConnectedHub.Name[0], (ConnectedHub.ActualLength)/2,
&ConnectedHubName[4], 252, NULL, NULL);
        ConnectedHubHandle = CreateFile(ConnectedHubName, GENERIC_WRITE, FILE_SHARE_WRITE, &SA,
OPEN_EXISTING, 0, NULL);
        if (DEBUG) printf("Connected hub handle %d created\n", ConnectedHubHandle);

// Connected hub is open. Collect the node information
        Success = DeviceIoControl(ConnectedHubHandle, IOCTL_USB_GET_NODE_INFORMATION, &NodeInformation,
            sizeof(NodeInformation), &NodeInformation, sizeof(NodeInformation), &BytesReturned, NULL);
        if (!Success) printf(" *** ERROR *** Node information not returned\n");
        GetPortData(ConnectedHubHandle, NodeInformation.HubDescriptor.bNumberOfPorts, HubDepth+1);
        if (DEBUG) printf("Closing connected hub handle %d\n", ConnectedHubHandle);
        CloseHandle(ConnectedHubHandle);
    };
};
}

DWORD EnumerateHostController(HANDLE HostControllerHandle) {
    DWORD BytesReturned;
    bool Success;
    struct {ULONG Length; WCHAR Name[256];} UnicodeName;
    char RootHubName[256] = "\\.\\";
    HANDLE RootHubHandle;
    NODE_INFORMATION NodeInformation;

// First get the system name of the host controller for display
    Success = DeviceIoControl(HostControllerHandle, IOCTL_GET_HCD_DRIVERKEY_NAME, &UnicodeName,
sizeof(UnicodeName),
        &UnicodeName, sizeof(UnicodeName), &BytesReturned, NULL);
    if (!Success) return GetLastError();
    printf("System name is %ws\n", &UnicodeName.Name[0]);

// Now get the system name of it's root hub for interrogation
    Success = DeviceIoControl(HostControllerHandle, IOCTL_USB_GET_ROOT_HUB_NAME, &UnicodeName,
sizeof(UnicodeName),
        &UnicodeName, sizeof(UnicodeName), &BytesReturned, NULL);
    if (DEBUG) printf(" and root hub name is %ws\n", &UnicodeName.Name[0]);
}

```

```

// Now open the root hub. Need to construct a char name from "\\.\\" + UnicodeName
WideCharToMultiByte(CP_ACP, 0, &UnicodeName.Name[0], (UnicodeName.Length)/2, &RootHubName[4], 252, NULL,
NULL);
RootHubHandle = CreateFile(RootHubName, GENERIC_WRITE, FILE_SHARE_WRITE, &SA, OPEN_EXISTING, 0, NULL);
if (RootHubHandle == INVALID_HANDLE_VALUE) return GetLastError();
if (DEBUG) printf("Root hub handle %d created\n", RootHubHandle);

// Root hub is open. Collect the node information
Success = DeviceIoControl(RootHubHandle, IOCTL_USB_GET_NODE_INFORMATION, &NodeInformation,
sizeof(NodeInformation), &NodeInformation, sizeof(NodeInformation), &BytesReturned, NULL);
if (!Success) return GetLastError();

// Can now iterate over the ports
GetPortData(RootHubHandle, NodeInformation.HubDescriptor.bNumberOfPorts, 0);
if (DEBUG) printf("Closing root hub handle %d\n", RootHubHandle);
CloseHandle(RootHubHandle);
return 0;
}

int main(int argc, char* argv[]) {
HANDLE HostControllerHandle;
char HostControllerName[] = "\\.\HCD0";
int i;
DWORD ErrorCode = 0;
FILE *stream;
// Say Hello to the user and setup the security attributes for Win2000
printf("USB Design By Example: Display currently attached USB Devices\n\n");
DEBUG = !true;
SA.nLength = sizeof(SEURITY_ATTRIBUTES);
SA.lpSecurityDescriptor = NULL;
SA.bInheritHandle = false;

// Search for USB host controllers. Product such as Lucent's QuadraBus can make this number large
// Redirect the output text to a file for later review
printf("Re-directing console output to 'DisplayUSB.txt'. Please wait\n\n");
stream = freopen("C:\\My Documents\\DisplayUSB.txt", "w", stdout);
if (stream == NULL) {
printf("Cannot redirect output\n");
return -1;
}
printf("USB Design By Example: Display currently attached USB Devices\n");

for (i=0; i<10; i++) {
HostControllerName[7] = i + '0';
HostControllerHandle = CreateFile(HostControllerName, GENERIC_WRITE, FILE_SHARE_WRITE, &SA,
OPEN_EXISTING, 0, NULL);
if (DEBUG) printf("Host controller handle %d created\n", HostControllerHandle);
if ((HostControllerHandle != INVALID_HANDLE_VALUE) & (ErrorCode == 0)) {
printf("\nHost Controller %s found. ", HostControllerName);
ErrorCode = EnumerateHostController(HostControllerHandle);
if (DEBUG) printf("Closing host controller handle %d\n", HostControllerHandle);
CloseHandle(HostControllerHandle);
}
}
fclose(stream);
system("notepad C:\\My Documents\\DisplayUSB.txt");
return ErrorCode;
}

```