

```

// DisplayHID.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 "stdio.h"
#include "stdlib.h"

extern "C" {
// Declare the C libraries used
#include "setupapi.h" // Must link in setupapi.lib
#include "hidsdi.h" // Must link in hid.lib
}

int main(int argc, char* argv[]) {
    struct _GUID GUID;
    SP_INTERFACE_DEVICE_DATA DeviceInterfaceData;
    struct {DWORD cbSize; char DevicePath[256];} FunctionClassDeviceData;
    HIDD_ATTRIBUTES HIDAttributes;
    SECURITY_ATTRIBUTES SecurityAttributes;
    int Device;
    HANDLE PnPHandle, HIDHandle;
    ULONG BytesReturned;
    bool Success, ManufacturerName, ProductName;
    WCHAR ManufacturerBuffer[256], ProductBuffer[256];
    const WCHAR NotSupplied[] = L"NULL";

// Say Hello to the user
printf("USB Design By Example: Display currently attached Human Interface Devices\n\n");

// 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 GetLastError(); }

// Lets look for a maximum of 20 Devices
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 GetLastError(); }
}

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

// Get the information about this HID
Success = HidD_GetAttributes(HIDHandle, &HIDAttributes);
if (!Success) { printf("Could not get HID attributes\n"); return GetLastError(); }
ManufacturerName = HidD_GetManufacturerString(HIDHandle, ManufacturerBuffer, 256);
ProductName = HidD_GetProductString(HIDHandle, ProductBuffer, 256);

// And display it!
printf("VID = %4.4x, Name = ", HIDAttributes.VendorID);
printf("%ws, ", ManufacturerName ? ManufacturerBuffer : NotSupplied);
}
}
}

```

```
        printf("PID = %4.4x, Name = ", HIDAttributes.ProductID);
        printf("%s\n", ProductName ? ProductBuffer : NotSupplied);
        CloseHandle(HIDHandle);
    }
} // if (SetupDiEnumDeviceInterfaces . .
} // for (Device = 0; (Device < 20); Device++)
SetupDiDestroyDeviceInfoList(PnPHandle);
return 0;
} // Main

extern "C" void __declspec(naked) __cdecl _chkesp() {
    _asm je okay
        printf("Stack pointer mismatch!\n");
okay:
    _asm ret
}
```