

```

// TwoKBDS.cpp : Defines the entry point for the 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.
//
//
// This example program shows two keyboards operating independantly
// It's goal is to show how characters from different keyboards can be identified and processed individually.
//
// This program needs KBDFILTR.SYS, a HID filter, which converts a "system" keyboard into a "user" keyboard.
//
// Send questions and comments to John.Hyde@intel.com

```

```

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

```

```

// Global Variables:
HINSTANCE hInst;           // current instance
TCHAR szTitle[100];       // The title bar text
TCHAR szWindowClass[100]; // The class name
HANDLE HidHandle;         // Handle used to communicate with User Keyboard
HWND hWnd;                // Handle for main window
BOOL connected;           // Used to prevent re-connection (messes up display)
char MainWindow[2000];    // A screen full of characters
int MainWindowIndex;      // A pointer into the main window
char userKBDchar[1000];   // Characters from the user keyboard
int userKBDIndex;         // A pointer into the user keyboard data
const char *SignOn = "\n Select Connect from the File menu to search for the User Keyboard -";
const char *Found = "-> found\n\n Characters from System Keyboard:\n";
const char *KBDsignon = "\n\n\n\n Characters from User Keyboard:\n";

```

```

// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);
BOOL FindUserKeyboard();

```

```

// Entry point for this program
int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
    MSG msg;
    HACCEL hAccelTable;
// Initialize global variables
    HidHandle = 0; userKBDIndex = 0; connected = false;
    LoadString(hInstance, IDS_APP_TITLE, szTitle, 100);
    LoadString(hInstance, IDC_TWOKBDS, szWindowClass, 100);
    MyRegisterClass(hInstance);
// Initialize application
    if (!InitInstance (hInstance, nCmdShow)) return false;
    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_TWOKBDS);
// Main message loop:
    while (GetMessage(&msg, NULL, 0, 0)) {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return msg.wParam;
}

```

```

// Registers the window class. (required)
ATOM MyRegisterClass(HINSTANCE hInstance) {
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
}

```

```

wceX.hInstance      = hInstance;
wceX.hIcon         = LoadIcon(hInstance, (LPCTSTR)IDI_TWOKBDS);
wceX.hCursor       = LoadCursor(NULL, IDC_ARROW);
wceX.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wceX.lpszMenuName  = (LPCSTR)IDC_TWOKBDS;
wceX.lpszClassName = szWindowClass;
wceX.hIconSm       = LoadIcon(wceX.hInstance, (LPCTSTR)IDI_SMALL);
return RegisterClassEx(&wceX);
}

// Initialize the program and create the main window
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {
    hInst = hInstance;
    hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW, 50, 50, 550, 450, NULL, NULL, hInstance, NULL);
    if (!hWnd) return false;
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    return true;
}

// Process messages for the main window.
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    int ID, i;
    PAINTSTRUCT ps;
    RECT rt;
    HDC hdc;

    switch (message) {
        case WM_COMMAND:
            ID = LOWORD(wParam);
// Parse the menu selections:
            switch (ID) {
                case IDM_ABOUT: DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About); break;
                case IDM_EXIT: DestroyWindow(hWnd); break;
                case IDS_CONNECT: if (!connected) FindUserKeyboard(); break;
                default: return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:
// Copy the user keyboard data into the main buffer for display
            for (i = 0; i < userKBDIndex; i++) MainWindow[MainWindowIndex + i] = userKBDchar[i];
// This is a minimal repaint. A "real" program would process characters such as BS, TAB, LF
            hdc = BeginPaint(hWnd, &ps);
            GetClientRect(hWnd, &rt);
            DrawText(hdc, MainWindow, MainWindowIndex+userKBDIndex, &rt, 0);
            EndPaint(hWnd, &ps);
            break;
        case WM_CHAR:
// Display characters from the system keyboard in the main window
            MainWindow[MainWindowIndex++] = (char)wParam; InvalidateRect(hWnd, NULL, 1); break;
        case WM_CREATE:
// Say Hello to the user
            for (MainWindowIndex = 0; MainWindowIndex < (int)strlen(SignOn); MainWindowIndex++)
                MainWindow[MainWindowIndex] = SignOn[MainWindowIndex];
            break;
        case WM_DESTROY: PostQuitMessage(0); break;
        default: return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
    int ID = LOWORD(wParam);
    switch (message) {
        case WM_INITDIALOG: return true;
        case WM_COMMAND: if (ID == IDOK || ID == IDCANCEL) {EndDialog(hDlg, ID); return true;}
    }
    return false;
}

// Support routines
BOOL DisplayError (const char *ErrorText) {int i = MessageBox(0, ErrorText, "Error", MB_ICONSTOP); return false;}

BOOL OpenUserKeyboard () {
// Search through the attached HID devices for the User Keyboard
// I used a keyboard with a VID = 046E and PID = 6782, your's may be different
// See KBDFILTR.SYS documentation which describes how to create a "User Keyboard"
}

```

```

// Declare the local data structures used
struct _GUID HidGuid;
SP_INTERFACE_DEVICE_DATA DeviceInterfaceData;
struct {DWORD cbSize; char DevicePath[256];} FunctionClassDeviceData;
int Success, HidDevice;
BOOL Openned;
HANDLE PnPHandle;
unsigned long BytesReturned;
struct _HIDD_ATTRIBUTES Attributes;
struct _HIDP_PREPARED_DATA *PreparedData;
struct _HIDP_CAPS Capabilities;

// First, get my class identifier
HidD_GetHidGuid(&HidGuid);
// Get a handle for the Plug and Play node and request currently active HID devices
PnPHandle = SetupDiGetClassDevs(&HidGuid,0,0,0x12);
if ((int)PnPHandle == -1) return DisplayError("Could not attach to PnP node");
Openned = false;
// Lets look for a maximum of 20 HID devices
for (HidDevice = 0; (HidDevice < 20) && !Openned; HidDevice++) {
// Give the user some feedback
MainWindow[MainWindowIndex++] = 45; // "-"
InvalidateRect(hWnd, NULL, 1);
// Initialize our Data
DeviceInterfaceData.cbSize = 28; // Length of data structure in bytes
// Is there a HID device at this table entry
Success = SetupDiEnumDeviceInterfaces(PnPHandle, 0, &HidGuid, HidDevice, &DeviceInterfaceData);
if (Success == 1) {
// There is a device here, get it's name
FunctionClassDeviceData.cbSize = 5;
Success = SetupDiGetDeviceInterfaceDetail(PnPHandle, &DeviceInterfaceData,
PSP_INTERFACE_DEVICE_DETAIL_DATA(&FunctionClassDeviceData), 256, &BytesReturned, 0);
if (Success == 0) return DisplayError("Could not find the system name for this HID device");
// Can now open this HID device
HidHandle = CreateFile(FunctionClassDeviceData.DevicePath, 0x00000000, 3, NULL, 3, 0, NULL);
if ((int)HidHandle == -1) return DisplayError("Could not open HID device");
// Is it OUR HID device?
if (!HidD_GetAttributes(HidHandle, &Attributes)) return DisplayError("Could not get VID/PID");
if ((Attributes.VendorID == 0x046e) && (Attributes.ProductID == 0x6782)) {
// Note that most keyboards are composite desktop devices, must find the "keyboard" usage (set to 0 by HIDFILT.RSYS)
if (!HidD_GetPreparedData(HidHandle, &PreparedData)) return DisplayError("Could not get Prepared
Data");
if (!HidP_GetCaps(PreparedData, &Capabilities)) return DisplayError("Could not get device
capabilities");
if ((Capabilities.UsagePage == 1) && (Capabilities.Usage == 0)) Openned = true; else
CloseHandle(HidHandle);
HidD_FreePreparedData(PreparedData);
}
} // if (SetupDiEnumDeviceInterfaces . .
} // for (HidDevice = 0; (HidDevice < 20) && !Openned; HidDevice++)
CloseHandle(PnPHandle);
return Openned;
}

// Declare the thread that will wait for characters from the User Keyboard
DWORD WINAPI ListenToUSB(LPVOID Param) {
// Collect characters from the user keyboard and send them to the display
char ReportBuffer[9], Key;
int Success, Shift;
unsigned long BytesReturned;
// Implementation note:
// I only check for alphanumeric key codes. A real program would check them all
// I only check for a single key code in ReportBuffer[3]
// I do check for SHIFT but not the other modifier keys (CNTRL, ALT)
const char *KeyCode = " abcdefghijklmnopqrstuvwxyz1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#%^&*()";

while (1) {
Success = ReadFile(HidHandle, ReportBuffer, 9, &BytesReturned, NULL);
Key = ReportBuffer[3];
if (Key == 44) Key = 3; // Allow SPACE
if (ReportBuffer[1] && 2) Shift = 37; else Shift = 0;
if ((Key > 2) && (Key < 40)) { userKBDchar[userKBDindex++] = KeyCode[Key - 3 + Shift];
InvalidateRect(hWnd, NULL, 1); }
};
return 0;
}

BOOL FindUserKeyboard () {
HANDLE ThreadHandle;

```

```
DWORD ThreadID;
int i;

if (!OpenUserKeyboard()) return DisplayError("Could not find User Keyboard");
else {
    connected = true;
    for (i=0; i<(int)strlen(Found); i++) MainWindow[MainWindowIndex++] = Found[i];
    for (i=0; i<(int)strlen(KBDsignon); i++, userKBDindex++) userKBDchar[i] = KBDsignon[i];
    InvalidateRect(hWnd, NULL, 1);
// Create a thread to listen for characters from the User Keyboard
    ThreadHandle = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)ListenToUSB, NULL, 0, &ThreadID);
    if ((int)ThreadHandle == 0) return DisplayError("Could not start USB Listener Thread");
}
return true;
}
```