1 序説

現在の資料はATMEL USB HID DLI関数を使用するための情報を与えます。
以下のような実装の種類を示すいくつかの簡単なコードもあります。

1つ目 : UsbHidSmallDemoCode
これはVID=$03EBとPID=$2013を使用して装置に接続する簡単なコンソール応用です。
一旦、装置に接続されると、基板上で釦が押されるまでLEDが点滅します。
情報が装置から読まれると、応用は装置を閉じて抜け出します。

2つ目 : UsbHidDemoCode
これは基板上に存在するLEDの設定、捕獲釦の動き、ファームウェア更新動作へ基板を設定、既定のPIDとVIDの変更、装置が接続または切断にされているかの検出に使用することができるダイアログ ボックス応用です。
これらの実演応用はVisual C++ 6.0とVisual C++サービスパック6で書かれています。

USB HID DLIはAtUsbHidディレクトリ内のそれのインクルード ファイルで得られます。

2 DLI関数

2.1 findHidDevice

指定された供給者 IDと製品 IDに一致するHIDのものに対して接続されたUSB装置を走査します。

入力

const UINT VendorID : これは供給者 IDです。
const UINT PorductID : これは製品 IDです。

出力

失敗ならば0、より多くの情報はGetLastError()を使用して得ることができます。
GetLastErrorは以下を返します。
装置を見つけることができなかった場合は、ERROR_USB_DEVICE_NOT_FOUND
装置が見つかるも能力を取得できない場合は、ERROR_USB_DEVICE_NO_CAPABILITIES
成功裏に接続された場合は、1

2.2 cbseDevice

USB装置と全てのハンドルを閉じます。

2.3 writeData

データを装置へ送ります。データ長が報告された大きさと等しいかまたはより小さいと時にこの関数を呼んでください。
緩衝は接続された装置の書き込み緩衝部の能力を超えてはなりません。

入力

UCHAR* buffer :書かれるべきメッセージへのポインタ

出力

失敗ならば 1、GetLastError()は ERROR_WRITE_FAULT符号を返します。
メッセージが成功裏におくられたならば 0。

2.4 readData

装置からデータを読みます。
データ長が報告された大きさと等しいかまたはより小さい時にこの関数を呼んでください。

入力

UCHAR* buffer :受信したパケットを含む緩衝部へのポインタ

出力

データが利用可能でない場合に 0。
データが UCHAR* buffer内に書かれている場合に 1。

2.5 startBootLoader

FLIPを用いてファームウェア更新を実行するために装置をブートローダ動作に設定。

出力

ブートローダ動作設定失敗の場合に 0。
ブートローダ動作が装置動作へ成功裏に送られた場合に 1。

2.6 hidRegisterDeviceNotification

ウィンドウが通知を受け取るための装置を登録します。

入力

HWND hWnd :ウィンドウへのハンドル

出力

関数失敗の場合に 0。拡張異常情報を得るには GetLastErrorを呼んでください。
関数成功の場合に 1

2.7 hidUnregisterDeviceNotification

この関数は指定した装置通知ハンドルを閉じます。

入力

HWND hWnd :ウィンドウへのハンドル

出力

関数失敗の場合に0。拡張異常情報を得るにはGetLastErrorを呼んでください。
関数成功の場合に1。

2.8 isMyDeviceNotification

この関数はこれがOnDeviceChangeの呼び出しを起動する我々の装置かを調べるために呼ばなければなりません。

入力

DWORD dwData :OnDeviceChangeの第2パラメータによって与えられる値

出力

これが接続した状態を変更する我々の装置ならば1。
これが別の装置ならば0。

3 VisualC++応用でのDLL読み込み設定

AtUsbHid.hファイルはATMEL USB HID DLL内に存在する関数の読み込み設定と使用を手助けするいくつかのマクロを提供します。
DLLを使用する応用を設計する時に、最初に以下を行うことが必要です。

- DLL :HINSTANCE hLib = NULL;へのハンドラを作成
- hLib = LoadLibrary(AT_USB_HID_DLL);を使用してDLLを読み込み設定
- badFuncPointers(hLib);を使用して各DLL関数を読み込み設定

一旦これらの段階が異常なしで実行されると、DLLと関すはあなたの応用内に読み込み設定され、今やDYNCALL(DllFunction());マクロを使用して呼び出すことができます。

プログラムが終えられた時にFreeLibrary(hLib);関数を使用してメモリからDLLを開放することは良いことです。メモリからDLLを開放する前にUSB装置が閉じられることを確実にしなければなりません。

4 自動装置接続 /切断機能の使い方

DLLは装置の状態が変わったかを検出するために使用者を助ける関数を提供します。
そうするためにあなたの応用で以下を行わなければなりません。

- DYNCALL(hidRegisterDeviceNotification)((m_hWnd))**を使用して装置変化通知を得るために**
  あなたの応用を登録してください。
- **あなたのメッセージ割り当て応用で**ON_WM_DEVICECHANGE()**関数を追加してください。**
- **装置状態変化毎に呼び出される**OnDeviceChange(UINT nEventType,DWORD dwData)**と呼ば**
  れる関数を作成してください。
- OnDeviceChange**関数に於いて関数を呼んでください。**
  DYNCALL(isMyDeviceNotification(dwData))**はあなたの装置状態が変わったかをあなたに告**
  げます (UsbHidDemoCodeDlg.cpp内の実演コードをご覧ください)

抜け出す時に、応用がDYNCALL(hidUnregisterDeviceNotification(m_hWnd))**関数を使用してあなたの**
応用を登録解除することは良いことです。

5 readDataの使い方

データは装置によって継続的に送られ得るため、計時器に基づく関数を使用してデータを読むことが
重要です。これは readData関数の継続的なポーリングを避けます。
そうするためにあなたの応用内で以下を行わなければなりません。

- **あなたのメッセージ割り当て応用内に**ON_WM_TIMER()**関数を追加してください。**
- DYNCALL(readData(sbuffer))**関数を呼ぶ**OnTimer(UINT nIDEvent)**関数を作成してください。**
- **その後、あなたの装置が接続された時に**SetTimer(1,50,0)**を使用して読み込み周期のために**
  計時器の設定を追加してください。SetTimer関数は読み込み周期を設定します。
- **あなたの装置が切断される時に**KillTimer(1)**を使用して計時器を削ってください。**