参考資料

Application Note

MSP430™ USB HID Windows API プログラマー・ガイド

William Goh, Keith Quiring *MSP430 Applications*

概要

API はデータ構造体を使用して、hiddevice.h で定義されている HID デバイスに関連するすべての情報を格納します (表 A-1 を参照)。この構造体のインスタンスは、デバイスに関係する任意の API 呼び出しに渡されます。

表 A-1. strHidDevice 構造体の定義

フィールド	概要	データの起源
HANDLE hndHidDevice	この HID デバイスのハンドル。	HID_Open() によって書き込まれます。HID_Close() をするとき、INVALID_HANDLE_VALUE に設定されます。
BOOL bDeviceOpen	デバイスが開いていること、および hndHidDevice のハンドルが有効であることを示すブール値。	HID_Open() および HID_Close() によって書き込まれます。
UINT uGetReportTimeout	データ読み取りのタイムアウト値 (ミリ秒単位)。これは、データ読み取り動作中に、デバイスからの特定のレポートを HID_ReadFile() が待機する時間です。	アプリケーションによって初期化される必要があります。デモ・アプリケーションでは、これを UsbAppDlg.cpp で実行します。
UINT uSetReportTimeout	データ書き込みのタイムアウト値 (ミリ秒単位)。これは、データ送信動作中に、デバイスが特定のレポートを受信するまで、HID_WriteFile() が待機する時間です。	アプリケーションによって初期化される必要があります。デモ・アプリケーションでは、これを UsbAppDlg.cpp で実行します。
OVERLAPPED oRead	非同期 I/O 構造体	HID_Init() によって初期化されます。
OVERLAPPED oWrite	非同期 I/O 構造体	HID_Init() によって初期化されます。
WORD wInReportBufferLength	入力レポートの最大長。	デバイスの USB ディスクリプタから派生した HID_Init() によって初期化されます。
WORD wOutReportBufferLength	出力レポートの最大長。	デバイスの USB ディスクリプタから派生した HID_Init() によって初期化されます。
BYTE inBuffer[256]	受信データを受信するためのバッファ。	HID_readFile() によって書き込まれ、アプリケーションによって読み取られます。
WORD inBufferUsed	inBuffer[] のバイト数。	HID_readFile() によって書き込まれ、アプリケーションによって読み取られます。

MSP430 USB API スタックを実行し、HID-datapipe インターフェイスを作成する USB デバイスは、表 B-1 に示すような形式の HID レポートを使用します。これは、物理 USB デバイスにある USB ディスクリプタで定義されています。この Windows API は同じ形式を使用しており、実際、HID デバイスがこの形式を使用していることを想定しています。

表 B-1. HID Datapipe レポートの構造

フィールド	サイズ
レポート ID	1 バイト (0x3F)
サイズ	1 バイト
データ	N-2 バイト。ここで、N は USB パケットのペイロード・サイズ (つまり、USB API スタックの descriptors.h ファイルで定義されている MAX_PACKET_SIZE の値) です。

1 MSP430™ USB HID Windows API プログラマー・ガイド

目次

2 はじめに

USB ヒューマン・インターフェイス・デバイス (HID) は、オペーレーティング・システムの中でも最も広範にサポートされているデバイス・クラスの 1 つです。元々はマウスとキーボード用に開発されたものですが、汎用の用途にも多くのメリットがあります。

汎用 USB アプリケーションの一般的な選択肢は、通信デバイス・クラス (CDC) を使用して実装できる仮想 COM ポートです。COM ポートはホスト・プラットフォームへの実装が簡単で、開発者にはよく理解されています。欠点は、Windows マシン用の USB 仮想 COM ポート・ソリューションでは、エンド・ユーザーがデバイスのインストール・プロセスを実行する必要があることです。このプロセスは、少々手間であることに加え、ユーザーが誤ったオプションを選択すると失敗する可能性があります。さらに、企業環境の一部のユーザーは管理者権限を持っていないため、ネットワーク管理者の助けがなければデバイスをインストールすることができません。一方、HID デバイスはサイレントにロードされ、これらの問題をすべて回避します。

HID にはいくつかの欠点があります。仮想 COM ポートと比較すると、多くのソフトウェア・エンジニアはその使用方法に慣れていません。HID は、データを伝送するのに「レポート」に依存しており、多くの場合、これらの複雑な形式は汎用アプリケーションに実際の価値を提供しません。基本的な HID 実装の帯域幅は、64KB/秒に制限されています。

これらの欠点を補うために、テキサス・インスツルメンツは Windows HID API とデモ・プロジェクトを提供しています。MSP430 HID API スタックに実装される「データパイプ・インターフェイス」と合わせて使用するように合理化されており、そのことにより HID レポートを生成する必要性が省かれます。 複合 HID デバイスとともに使用され、複数の HID インターフェイスを USB デバイスの中に複合的に組み込むことを容易にして、帯域を倍増させます。

合わせて使用すれば、MSP430 HID API スタックと Windows HID API がエンドツーエンドなソリューションを形成し、多くの汎用アプリケーションで仮想 COM ポートよりもメリットがあります。

MSP430 MCU 用の USB サポート・ソフトウェアについては、 の MSP430 USB 開発パッケージを参照してください。

3 実装

Windows MSP430 HID API を、以下では「Windows API」とします。

3.1 概要

図 3-1 に、Windows ソフトウェア・スタック内でのこの API の位置と、MSP430 デバイスの USB HID API との関係を示します。

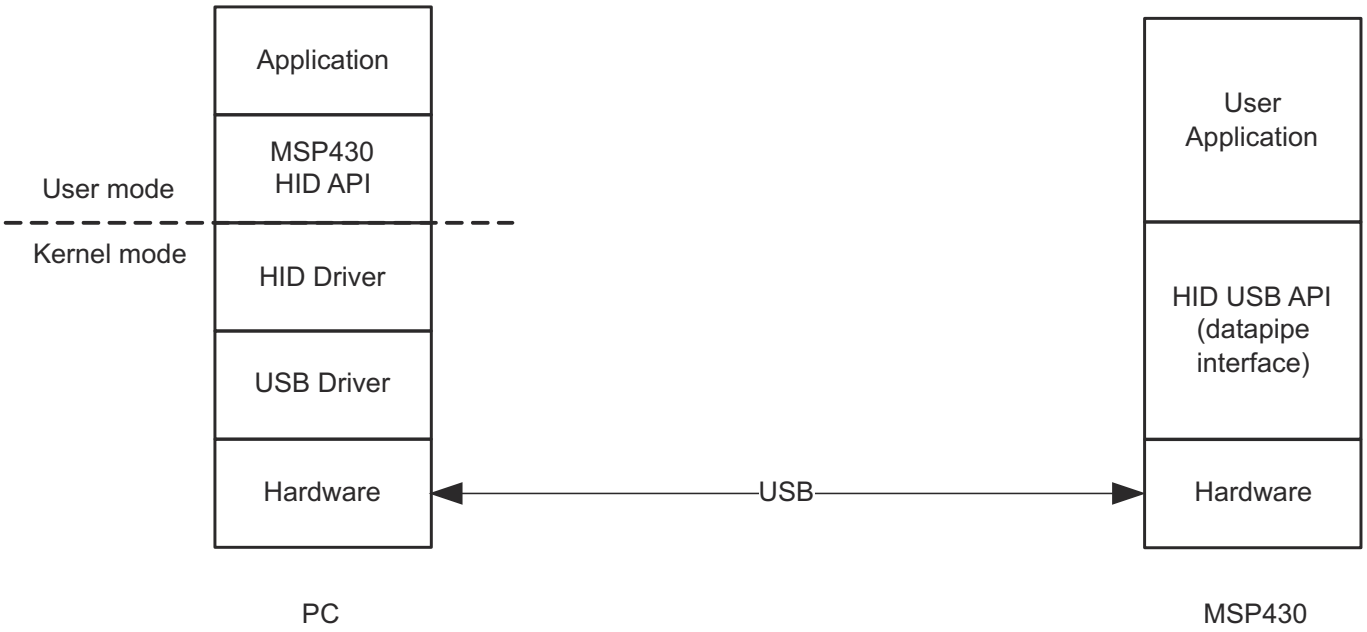

図 3-1. Windows および MSP430 ソフトウェア・スタック

このソフトウェアは、MSP430 の USB HID API スタック・ファームウェア、特に API スタックが提供するデータパイプ・インターフェイスで使用するように設計されています。API スタックと Windows API を一緒に使用することで、ソフトウェア開発者は HID レポートから抽象化され、開発者はデータ・インターフェイスを COM ポートと同様のフォーマットされていないデータ・ストリームと考えることができるようになります。

この抽象化を実現するために、データパイプ・インターフェイスは非常にシンプルな HID レポート構造を実装しており、開発者が修正する必要はありません。開発者はシンプルなコマンドを使用して任意のサイズのデータ・ブロックを送受信し、API はレポートをデータ・パケットとして使用してこのデータを転送します。(データパイプ・インターフェイスの詳細については、『MSP430 USB CDC/HID API スタック・プログラマ・ガイド』を参照してください。)

3.2 ファイル構成

表 3-1 に、API およびデモ・アプリケーションのファイルを示します。

表 3-1. ソース・コード・ファイル

ファイル	概要
デモ・アプリケーション	
*.cpp	さまざまな C++ ファイル
API	
hiddevice.c	API 関数呼び出しとサポート関数呼び出しのコードで実装。
hiddevice.h	アプリケーションで使用できる API 関数呼び出しの定義。

3.3 システム要件

配布されている Windows API は、Microsoft Foundation Class (MFC) ライブラリを使用して Visual Studio 2008 用に記述されています。そのため、Windows Vista/XP/7 のすべてのバリエーションを含む、さまざまな Windows プラットフォームで使用できます。この API は C 言語で記述されており、デモ・アプリケーションは C++ で記述されています。Visual Studio の Express バージョンは、API をコンパイルするのに十分です。ただし、デモ・アプリケーションをコンパイルするにはフル・バージョンが必要です。

この API を使用するには、Windows Driver Kit (WDK) がインストールされている必要もあります。これは Microsoft から無料で入手できます。以前のバージョンに比べて速度が向上するため、バージョン 7 を推奨します。

API を WDK にリンクするには、以下のようにしてください。

1. Microsoft から最新バージョンの WDK をダウンロードしてインストールします。
2. HidDevice.c ファイルを右クリック → Properties
3. Navigate to C/C++ → General → Additional Include Directories
4. 以下をインクルードします。
   a. c:\<WinDDK Install>\<Build Version>\inc\api\
   b. c:\<WinDDK Install>\<Build Version>\inc\crt\
5. Navigate to Linker → General → Additional Include Directories
6. c:\<WinDDK Install>\<Build Version>\lib\wxp\i386 をインクルードします。

3.4 MSP430 USB API スタック

MSP430 USB API は、USB について詳しく学習しなくても、USB デバイスを簡単に作成できるように設計されています。これには、MSP430 USB ディスクリプタ・ツールが付属しており、以下を自動的に実行します。

- USB インターフェイス (単一または複合) の任意の組み合わせに対して API スタックを構成
- 初めて動作する USB ディスクリプタを作成

HID 用の API では、2 種類の HID 実装が可能です。1 つは、従来の HID デバイス用で、HID レポートに関する詳細な知識を必要とします。もう 1 つはデータパイプ HID デバイスです。これは、HID をシンプルなデータ・キャリアとして使用する方法をテキサス・インスツルメンツが実装したものです。これにより、設計者が HID レポートを作成する必要がなくなります。

MSP430 アプリケーションにとって、HID- データパイプ・インターフェイスは COM ポートによく似ているように見え、感じられます。

- 比較的形式が設定されていないため、設計者は自分が選択した形式を適用できます。
- 任意のサイズのデータ・チャンクを処理します (USB パケット・サイズによる制限はありません)。

実際、HID- データパイプの API は、CDC (ホスト上で仮想 COM ポートを生成するために使用される) の API とほぼ同じです。

MSP430 USB API スタックの詳細については、https://www.tij.co.jp/tool/MSP430USBDEVPACK にある MSPUSBDEVPACK のドキュメントを参照してください。

3.5 Windows が物理 USB HID デバイスをホスト・アプリケーションにマッピングする方法

すべての USB デバイスには、ベンダ ID (VID) とプロダクト ID (PID) が含まれています。USB ホストは、VID/PID の組み合わせを使用して製品タイプを識別します。特定の VID のベンダから供給されている特定の PID のすべての製品が機能的に同一であると想定します。ホストは、その VID/PID を持つすべてのデバイスを特定の USB デバイス・クラス (HID や CDC など) に関連付けます。この VID/PID に遭遇すると、その特定のドライバがロードされます。

Windows HID ドライバ上では、状況が少し異なります。HID ドライバは、汎用インデックスで選択可能な「HID デバイス」のリストをバス上に保持し、このリストをその上のホスト・アプリケーションが利用できるようにします。このリストの「HID デバイス」は論理エンティティであり、物理 USB デバイスではありません。これらの論理エンティティは、必ずしもバス上の物理デバイスに直接マッピングされるわけではありません。

たとえば、特定の VID/PID を持つ複数の「HID デバイス」がホスト・アプリケーションに報告される可能性があります。これは、特定の製品タイプの複数の物理デバイスがこのホストに接続されていることを意味する場合があります。または、複数の HID インターフェイス (論理デバイスと考えることができる) を持つ複合デバイスが存在する可能性があります。さらに、4 個以上の「HID デバイス」が存在する場合は、図 3-2 に示すように、これら両方の要因が関係している可能性があります。

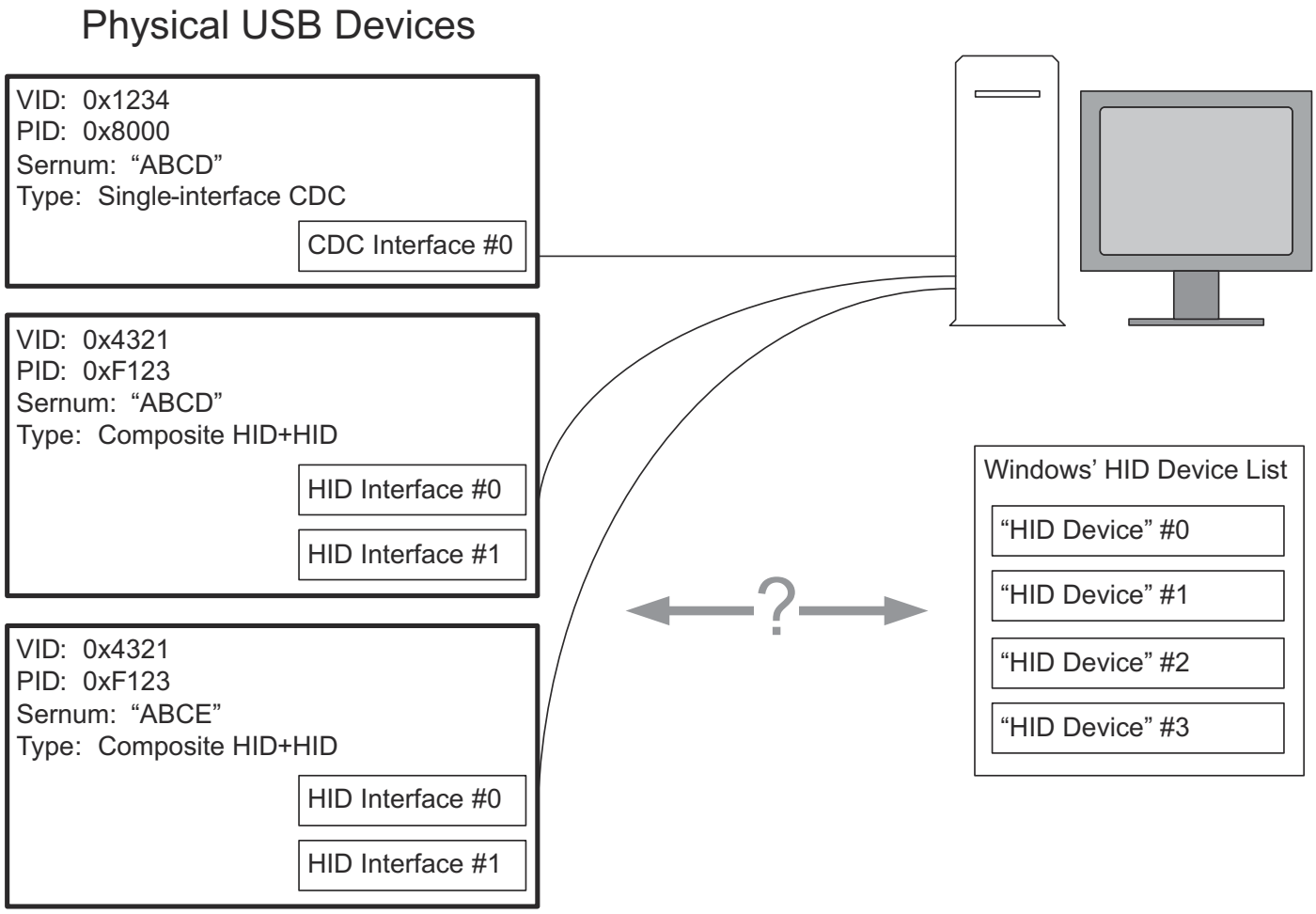

図 3-2. Windows システム上での「HID インターフェイス」から「HID デバイス」へのマッピング

この図では、Windows アプリケーションは、これらの「HID デバイス」が実世界では何を表すか、すぐに知ることはできません。

ホスト・アプリケーションが、このようなマルチデバイスの状況で機能することを望む場合、またはさらに、物理デバイス上の複数の HID データパイプを利用できるようにすることを望む場合、さらに調査する必要があります。各 HID デバイスから収集する必要がある重要な情報の 1 つは、シリアル番号 (図 3-2 では「sernum」と表記) です。USB デバイスはオプションで、同じ VID/PID を含む他のすべての物理機器の中で一意の機器を識別するシリアル番号を報告することができま

す。(MSP430 USB API スタックは、ディスクリプタ・ツールでシリアル番号を報告するように選択した場合、自動的にシリアル番号を報告できます。)

ホスト・アプリケーションは、バス上で認識されるすべての HID デバイスの VID、PID、およびシリアル番号を取得すると、バス上の状況を理解し始めることができます。特定の VID/PID を持つ 2 つの HID デバイスを見つけ、シリアル番号も同じである場合、これは 2 つの HID インターフェイスを持つ複合 HID デバイスであることがわかります。代わりに、これら 2 つの HID デバイスのシリアル番号が異なる場合、アプリケーションは 2 つのシングル・インターフェイス HID 物理デバイスを見ていることになります。

これらの手法を使用すると、開発者は複数の HID インターフェイスを持つデバイスを設計し、それぞれを特定の種類の情報のデータパイプ (複数の COM ポートに相当) として使用することができます。他の利点として、これは単一の HID インターフェイス (64kbps) の帯域幅制限を克服するのに役立ちます。MSP430 では、最大 8 つの HID インターフェイスを並列して実行でき、512kbps の帯域幅を提供します。

Windows が初期列挙時に、このデバイスがマウスまたはキーボードであることを USB ディスクリプタで認識した場合、Windows 自体がこの HID デバイスの「ホスト・アプリケーション」になることを付け加えておきます。レポート要求を発行し、そのデータを使用して画面上のマウスを制御したり、テキスト入力のためのデータを入力したりします。

3.6 システム上で特定の HID デバイス / インターフェイスを見つけて開く

このドキュメントの残りの部分では、以下の用語を使用します。

- HID デバイス: Windows HID デバイス・リスト内のデバイス
- 物理 USB デバイス:USB バス上に独自のアドレスを持つ、実際の USB ハードウェア
- HID インターフェイス:物理 USB デバイスの USB ディスクリプタ内で宣言されたインターフェイス。デバイス上で唯一のインターフェイスである場合もあれば、複合 USB デバイスの一部として複数のインターフェイスの 1 つである場合もある。HID インターフェイスは、Windows の HID デバイスに関連付けられている。

ホスト・アプリケーションの開発者は、USB デバイスへのアクセスを開始するとき、通常、関連付けられているデバイスの VID/PID ペアを知っています。両者は通常、同じ当事者によって設計されているからです。したがって、このプロセスは、HID_GetSerNums() の呼び出しから始まります。この関数は、VID/PID をパラメータとして受け取り、シリアル番号のリストを返します。各シリアル番号は、その VID/PID に関連付けられているバス上の物理 USB デバイスを表します。アプリケーションは、これらの物理デバイスのうち 1 つだけと対話することを選択することも、複数のデバイスと対話する機能を与えることもできます。

また、アプリケーション開発者は、通常、これらの物理デバイスがそれぞれいくつのインターフェイスを持っているかも知っています。何らかの理由でこれがわからない場合は、関数 HID_GetNumOfInterfaces() を呼び出すことができます。デバイスを開くために HID_Open() を呼び出すときは、物理デバイス上の HID インターフェイスの総数を知る必要があります。

VID、PID、およびシリアル番号を使用して HID デバイス・リストをフィルタリングすると、残るのは、これらのパラメータで記述された物理 USB デバイス内のすべての HID インターフェイスを表すリストだけです。このリストは通常 1 つまたは 2 つですが、MSP430 では最大 8 つになる場合があります。このリストの順序は、物理デバイスの USB ディスクリプタに HID インターフェイスがリストされている順序と同じです。このようにして、API スタックを使用するホスト・アプリケーションと MSP430 アプリケーションは互いに「見つける」ことができ、完全なデータ・リンクを形成できます。

USB デバイス内の目的の HID インターフェイスの VID/PID、シリアル番号、およびインデックスを用意すると、アプリケーションは HID_Open() を呼び出して、物理 USB デバイス上の HID インターフェイスへの接続を開くことができます。

アプリケーションがデバイスの使用を終了するとき (プログラムを終了するときなど)、アプリケーションは HID_CLOSE() で、開いている HID デバイスを閉じる必要があります。

3.7 データの送受信

HID インターフェイスが開かれると、そしてこのインターフェイスが MSP430 HID API の「データパイプ」インターフェイスとして設定されていると仮定すると、HID_writeFile() および HID_readFile() 呼び出しを使用して、任意のサイズのデータ・チャンクを API で送受信することができます。この呼び出しでは、1 つ以上の HID レポートへのパケット化が自動的に処理されます。

「データパイプ」ではなく「従来型」HID デバイスを実装する場合、つまりレポートのフォーマットがカスタマイズされ、MSP430 HID アプリケーションで従来型の HID 関数呼び出しが使用されている場合でも、この API を例として使用することは可能です。ただし、API コードをカスタマイズする必要があります。

3.8 HID デバイスの動的接続 / 切断の検出

従来の COM ポートとは異なり、USB アプリケーションでは、ユーザーがデバイスをいつでも取り外す (または接続する) ことができることを認識する必要があります。これが USB の動的な「プラグ・アンド・プレイ」の特徴です。Windows は、MFC ライブラリに実装されている通知機能によってこれを処理します。通知はアプリケーションのウィンドウ・オブジェクトに直接送られるため、これは API の外部で行われます。ただし、これらの通知を受信すると、アプリケーションはこの目的のために特別に提供された API ハンドラを呼び出すことができます。デモ・アプリケーションでは、これを行う方法を示します。

アプリケーションがこれらの通知を処理するために必要な手順は以下のとおりです。

1. アプリケーションの開始時に、アプリケーションは、HID_RegisterForDeviceNotification() を使用してデバイス変更通知を受信するためのウィンドウを登録する必要があります。
2. ON_WM_DEVICECHANGE をウィンドウのメッセージ・マップに追加します。(これが AFX_MSG_MAP のコメント・セクションの外にあることを確認してください。)
3. 関数 OnDeviceChange() をウィンドウのパブリック関数として書きます。このメッセージ・ハンドラは、ON_WM_DEVICECHANGE 通知に応答して、MFC フレームワークによって呼び出されます。

この関数の定義は、次のようになります。

```
afx_msg BOOL OnDeviceChange(
   UINT nEventType,
   DWORD_PTR dwData
);
```

この関数は、このアプリケーションで使用されている「HID デバイス」のいずれかがシステムから削除されたかどうかを判断する必要があります。そのために、オープンしている各 HID デバイスに対して HID_IsDeviceAffected() を呼び出して、それらが削除されたかどうかを判断できます。

アプリケーションを終了するときは、HID_UnregisterForDeviceNotification() を使用して、必ずアプリケーションの登録を解除する必要があります。

4 関数呼び出しのリファレンス

4.1 デバイス接続管理および初期化呼び出し

表 4-1 は、デバイス接続管理のコールをまとめたものです。

表 4-1. デバイス接続管理のコールの概要

機能	概要
VOID HID_Init()	HID デバイス / インターフェイスに関する情報を格納するために使用される HID デバイス・データ構造体を初期化します。
DWORD HID_GetNumOfInterfaces()	特定の VID/PID およびシリアル番号に関連付けられているシステム上の HID デバイスの数を返します。
BYTE HID_Open()	特定のデバイスへのハンドルを開きます。
BYTE HID_Close()	特定のデバイスへのハンドルを閉じます。
BYTE HID_GetSerNums()	特定の VID/PID に関連付けられているシリアル番号を取得します。

4.1.1 VOID HID_Init(struct strHidDevice* pstrHidDevice)

概要

この関数は pstrHidDevice を初期化します。使用する前に、すべての strHidDevice 構造体インスタンスを一度呼び出す必要があります。

パラメータ

表 4-2. HID_Init() のパラメータ

strHidDevice* pstrHidDevice	初期化される構造体。

4.1.2 DWORD HID_GetSerNums(WORD vid, WORD pid, struct strTrackSerialNumbers *serialNumList)

概要

この vid と pid の組み合わせに関連するシステム上の物理 USB デバイスの数を返します。(これらのデバイスは、HID またはその他のタイプの場合があります。)デバイスが接続されていない場合は、0 を返します。デバイスが接続されている場合、渡された strTrackSerialNumbers 構造体に、見つけられた物理デバイスに対応するシリアル番号が入力され、この関数はリスト内のシリアル番号の総数を返します。

返された結果が複数である場合は、vid/pid の組み合わせを持つ物理デバイスが複数存在することを意味します。

パラメータ

表 4-3. HID_GetSerNums() のパラメータ

WORD vid	検索するデバイスの 16 ビットのベンダ ID。
WORD pid	検索するデバイスの 16 ビットのプロダクト ID。
struct strTrackSerialNumbers *serialNumList	vid/pid に関連付けられているシリアル番号のリストを格納する構造体。 この関数は、見つかったシリアル番号を構造体に格納します。
戻り値	0:この VID/PID を持つ物理 USB デバイスが接続されていない場合。 0 以外:この VID/PID を持つシステム上の物理 USB デバイスの数。

表 4-4. strTrackSerialNumbers 構造体の定義

フィールド	概要
DWORD deviceNum	物理 USB デバイスを表すインデックス番号。
char serialNum[SERNUM_LEN];	検出された物理デバイスのシリアル番号を格納する文字列。

4.1.3 DWORD HID_GetNumOfInterfaces(WORD vid, WORD pid, DWORD numSerNums)

概要

vid と pid で識別される各物理 USB デバイス内に存在する HID インターフェイスの数を返します。そのようなデバイスが接続されていない場合、この関数は 0 を返します。

numSerNums は、この VID/PID に関連付けられている、システム上の物理デバイスの数です。この関数は、Windows HID デバイス・リスト (つまり、システムに登録されている論理 HID デバイス) をスキャンし、この VID/PID に関連付けられているデバイスをカウントし、それを numSerNums で割ることによって目的を達成します。numSerNums は、HID_GetSerNums() を呼ぶことで確認できます。

単一インターフェイスの HID デバイスでは、常に値 1 が返されます。

パラメータ

表 4-5. HID_GetNumOfInterfaces() のパラメータ

UINT vid	検索されるデバイスの 16 ビットのベンダ ID。
UINT pid	検索されるデバイスの 16 ビットのプロダクト ID。
DWORD numSerNums	この VID/PID を持つ接続されている物理デバイスの総数。
戻り値	0:この VID/PID を持つ USB デバイスが接続されていない場合。 0 以外:この VID/PID を持つシステム上の USB デバイスの数。

4.1.4 BYTE HID_Open(struct strHidDevice* pstrHidDevice, WORD vid, WORD pid, DWORD deviceIndex, char serialNumber[SERNUM_LEN], DWORD totalDevNum, DWORD totalSerNum)

概要

システム上のデバイスのリスト内で、この VID/PID およびシリアル番号に関連付けられている、インデックス deviceIndex のデバイスを見つけようとします。見つかった場合は、デバイスへのハンドルを開いて pstrHidDevice 構造体に格納し、構造体内の他のフィールドもロードします。

目的を達成するには、HID_Open() に 2 つの追加パラメータが必要です。1 つは totalDevNum で、vid と pid で記述されるデバイス上の HID インターフェイスの数です。通常、開発者はこれをすでに知っています。そうでない場合は、HID_GetNumOfInterfaces() を呼び出すことができます。もう 1 つの必須パラメータは、totalSerNum で、この VID/PID に一致するシステム上のシリアル番号 (物理デバイス) の総数です。

deviceIndex は 0 から totalDevNum-1 までの数値です。vid、pid、および serialNumber を使用して Windows HID デバイス・リストをフィルタリングすると、リストは特定の物理的 USB デバイスに含まれる HID インターフェイスを表す HID デバイスに実質的に縮小されます。残るのは、この中から 1 つのインターフェイスの選択することだけですが、これを行うのが deviceIndex です。これらの HID デバイスの順序 (0 から totalDevNum) は、HID インターフェイスがデバイスの USB ディスクリプタで宣言された順序と同じです (セクション 3.5 を参照)。

vid と pid で記述されるデバイスに HID インターフェイスが 1 つしかない場合、deviceIndex は単純に 0 にすることができます。

この関数は、このデバイスが実際に HID デバイスであり、別のタイプの USB デバイスではないことを想定しています。

パラメータ

表 4-6. HID_Open() のパラメータ

strHidDevice* pstrHidDevice	新しく開いたデバイスを格納するための構造体。
WORD vid	オープンするデバイスの 16 ビットのベンダ ID。
WORD pid	オープンするデバイスの 16 ビットのプロダクト ID。
DWORD deviceIndex	この VID/PID とシリアル番号の使用可能なデバイス内のインデックス。
char serialNumber[SERNUM_LEN]	検索するデバイスのシリアル番号の、サイズ SERNUM_LEN (40) の文字列。
DWORD totalDevNum	この VID/PID とシリアル番号で記述される物理デバイスの HID インターフェイスの総数。
DWORD totalSerNum	この VID/PID で使用可能な物理的 USB デバイスの総数
戻り値	HID_DEVICE_SUCCESS デバイスが見つかり、開かれました。 HID_DEVICE_ALREADY_OPENED デバイスはすでに開かれています。 HID_DEVICE_NOT_FOUND この VID/PID/index で識別されたデバイスが見つかりませんでした。

4.1.5 BYTE HID_Close(struct strHidDevice* pstrHidDevice)

概要

この pstrHidDevice 構造体に関連付けられているデバイスを閉じ、ハンドルを閉じるようにシステムに指示します。これは、デバイスに行う場合、常にアプリケーションで実行する必要があります。

パラメータ

表 4-7. HID_Close() のパラメータ

strHidDevice* pstrHidDevice	新しく開いたデバイスを格納するための構造体。
戻り値	HID_DEVICE_HANDLE_ERROR デバイス・ハンドルが無効です。 HID_DEVICE_SUCCESS デバイス・ハンドルが正常に閉じられました。 HID_DEVICE_NOT_OPENED 開かれていないデバイス・ハンドルを閉じようとしました。

4.1.6 BYTE HID_GetVersionNumber(struct strHidDevice* pstrHidDevice, USHORT * VersionNumber)

概要

pstrHidDevice に関連付けられているデバイスのリリース番号を VersionNumber に配置します。これは、物理的な USB デバイスのデバイス・ディスクリプタの bcdDevice フィールドで報告される値です。

パラメータ

表 4-8. HID_ GetVersionNumber() のパラメータ

strHidDevice* pstrHidDevice	HID デバイス情報を含む構造体。
USHORT* VersionNumber	デバイスのリリース番号。
戻り値	HID_DEVICE_HANDLE_ERROR デバイス・ハンドルが無効です。 HID_DEVICE_SUCCESS

4.2 データの送受信

表 4-9 に示されているコールは、データの送受信に関連しています。

表 4-9. デバイス接続管理のコールの概要

機能	概要
BYTE HID_WriteFile()	デバイスにデータを送信します。
BYTE HID_ReadFile()	デバイスからデータを受信します。

4.2.1 BYTE HID_WriteFile(struct strHidDevice* pstrHidDevice, BYTE* buffer, DWORD bufferSize, DWORD* bytesSent)

概要

pstrHidDevice で指定された HID デバイスにバッファに、バッファに格納されているデータの bufferSize バイトを送信します。

この関数は、HID レポートを使用してデータを「パケット」として送信します。HID レポートの送信試行のいずれかがタイムアウトした場合、この関数は HID_DEVICE_TRANSFER_TIMEOUT を返します。

bufferSize が DWORD 値であることによる制限以外には、送信できるバイト数に固有の制限はありません。パケット化は自動的に処理されます。

パラメータ

表 4-10. HID_WriteFile() のパラメータ

strHidDevice* pstrHidDevice	HID デバイス情報を含む構造体。
BYTE* buffer	送信するデータの配列。
DWORD bufferSize	アドレス・バッファから開始して、送信するバイト数。
DWORD bytesSent	実際に送信されたバイト数 (エラー発生時)。
戻り値	HID_DEVICE_NOT_OPENEDHID デバイスを開けませんでした。 HID_DEVICE_TRANSFER_TIMEOUT レポート要求がタイムアウトしました。 HID_DEVICE_TRANSFER_FAILED 不明な理由により送信に失敗しました。 HID_DEVICE_SUCCESS すべてのデータが正常に送信されました。

4.2.2 BYTE HID_ReadFile(struct strHidDevice* pStrHidDevice, BYTE* buffer, DWORD bufferSize, DWORD* bytesReturned)

概要

pStrHidDevice で指定された HID デバイスから、bufferSize バイトのペイロード・データを取得し、バッファに格納します。これは、bufferSize バイトが返されるまで、必要なだけ HID レポートを読み取ることによって行われます。

この関数が HID_DEVICE_SUCCESS を返す場合、bytesReturned は bufferSize と等しくなります。他のリターン・コードのいずれかが発生した場合にのみ、bytesReturned は bufferSize より小さくなります。

この関数は、HID レポートを「パケット」として使用してデータを受信します。pStrHidDevice.uGetReportTimeout の値に従って、USB デバイスからパケットを読み取る試みがタイムアウトした場合、この関数は HID_DEVICE_TRANSFER_TIMEOUT を返します。

bufferSize が DWORD 値であることによる制限以外に、受信できるバイト数に固有の制限はありません。パケット化は自動的に処理されます。

パラメータ

表 4-11. HID_ReadFile() のパラメータ

strHidDevice* pstrHidDevice	HID デバイス情報を含む構造体。
BYTE* buffer	受信データを格納するための配列。
DWORD bufferSize	バッファのサイズ。受信しようとする最大バイト数を示します。
DWORD* bytesReturned	実際に受信したバイト数 (エラー発生時)
戻り値	HID_DEVICE_NOT_OPENEDHID デバイスを開けませんでした。 HID_DEVICE_TRANSFER_TIMEOUT レポート要求がタイムアウトしました。 HID_DEVICE_TRANSFER_FAILED 不明な理由により送信に失敗しました。 HID_DEVICE_SUCCESS 正しいバイト数を受信しました。

4.3 プラグ・アンド・プレイ管理

表 4-12 に示されているコールは、アプリケーションが USB デバイスの動的な取り付け / 取り外しを管理するのに役立ちます。

表 4-12. デバイス接続管理のコールの概要

機能	概要
BYTE HID_RegisterForDeviceNotification()	デバイスにデータを送信します。
BYTE HID_UnregisterForDeviceNotification()	デバイスからデータを受信します。
BYTE HID_IsDeviceAffected()	特定のデバイスがまだシステム上にあるかどうかを判断します。

4.3.1 BYTE HID_RegisterForDeviceNotification(HWND hWnd, HDEVNOTIFY* diNotifyHandle)

概要

デバイスがシステムに追加または削除されたときに通知を受け取るために、ハンドル hWND が指すウィンドウを登録します。セクション 3.6 で説明されているように、アプリケーションがシステム通知に応答できるようにするための手順を実行する必要があります。

デバイス通知ハンドルは diNotifyHandle に返されます。これは、HID_UnregisterForDeviceNotification() を呼び出すときに使用するために、アプリケーションによって保存される必要があります。

パラメータ

表 4-13. HID_RegisterForDeviceNotification() のパラメータ

HWND hWND	メイン・ウィンドウのハンドル。
HDEVNOTIFY* diNotifyHandle	デバイス通知ハンドル。
戻り値	HID_DEVICE_HANDLE_ERROR デバイス・ハンドルが無効です。 HID_DEVICE_SUCCESS

4.3.2 BYTE HID_UnregisterForDeviceNotification(HDEVNOTIFY* diNotifyHandle)

概要

HID_RegisterForDeviceNotification によって返されたデバイス通知ハンドル diNotifyHandle が指すウィンドウの登録を解除します。この呼び出しの後、Windows はデバイスがシステムに追加または削除されたときにアプリケーションに通知しなくなります。この関数は、ハンドルが有効であるかどうかにかかわらず、真を返します。

パラメータ

表 4-14. HID_UnregisterForDeviceNotification() のパラメータ

HDEVNOTIFY* diNotifyHandle	デバイス通知ハンドル。
戻り値	HID_DEVICE_SUCCESS

4.3.3 BOOL IsDeviceAffected(struct strHidDevice* pstrHidDevice)

概要

アプリケーションは、システムから受信した ON_WM_DEVICECHANGE 通知に応答して、この関数を呼び出す必要があります。イベントが、pstrHidDevice に関連するデバイスを参照しているかどうかを示します。

パラメータ

表 4-15. HID_IsThisDeviceAffected() のパラメータ

strHidDevice* pstrHidDevice	HID デバイス情報を含む構造体。
戻り値	TRUE または FALSE

5 デモ・アプリケーション

API の使用方法を示す簡単なデモ・アプリケーションが提供されています。このアプリケーションは、ハイパーターミナルのような COM ポート・ターミナル・アプリケーションと似ていると考えることができます。HID インターフェイスを経由すること以外は、同様の方法でデータのチャンクを送受信します。

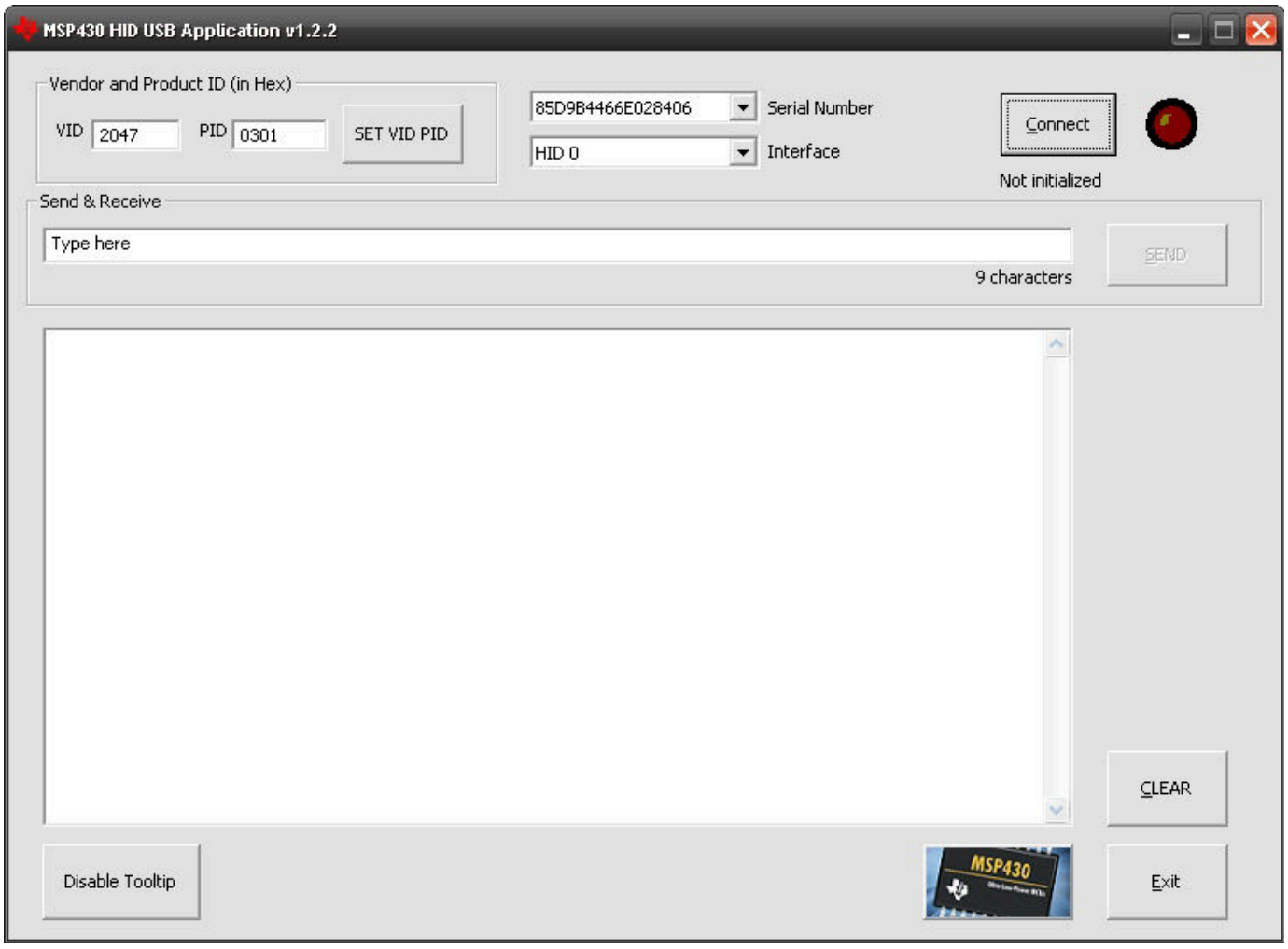

図 5-1. デモ・アプリケーション・ウィンドウ

このプログラムを動作させるには、datapipe 関数呼び出しを使用して HID API スタックを実行する MSP430 を接続します。API スタック・ディストリビューションに、いくつかの例が提供されています。シングル・インターフェイス HID、CDC+HID 複合デバイス、および HID+HID 複合デバイスの例があります。これらはすべて、このデモ・アプリケーションで使用できます。

GUI の VID および PID フィールドに、アプリケーションで検索する VID/PID を選択します。入力後、「Set VID/PID」ボタンを押します。

VID/PID が設定されている場合、アプリケーションは、この VID/PID に一致する物理デバイスがいくつシステムに接続されているかを決定します。シリアル番号を見つけ、「Serial Number」コンボ・ボックスに入力します。リスト内の最初のシリアル番号が自動的に選択され、そのデバイスに存在する HID インターフェイスの数を決定します。「Interface」コンボ・ボックスに、それぞれのインターフェイスに対応する文字列がロードされます。「HID 0」、「HID 1」など。HID インターフェイスが見つかった場合、デフォルトで「HID 0」が選択されます。

「Connect」ボタンを押します。これにより、USB デバイスとのデータ接続が開きます。成功した場合は、ボタンの下の文字列にそのことが示されます。成功しない場合は、以下をお試しください。

- Windows デバイス・マネージャをチェックして、デバイスが HID デバイスとしてシステムに正常に列挙されていることを確認
- デバイスに MSP430 HID API スタックが搭載され、HID-datapipe アプリケーションが実行されていることを確認
- descriptors.h に表示されている VID/PID が、デモ・アプリのフィールドに入力されたのと同じペアであること、および「Set VID/PID」ボタンが押されていることを確認

接続が初期化されたら、テキストを入力して「Send」を押すと、データをデバイスに送信できます。アプリケーションがデバイスから随時受信したデータは、大きなテキスト・フィールドに表示されます。「クリア」ボタンで受信ウィンドウをクリアすることができます。

別のデバイスへのアクセス中に HID デバイスがバスから削除された場合、またはこの VID/PID の別の HID デバイスが追加された場合、アプリケーションはプルダウン・メニュー・リストを自動的に更新します。

6 MSP430 USB ツール・スイート

この API は、MSP430 で USB の使用をより簡単にするために、テキサス・インスツルメンツが提供する完全なツール・スイートの一部です。以下のようなツールが含まれています。

- MSP430 USB API スタック
  - CDC (通信デバイス・クラス)
  - HID (ヒューマン・インターフェイス・デバイス・クラス)
  - MSC (マス・ストレージ・クラス)
- MSP430 USB ディスクリプタ・ツール

  最初に動作する USB ディスクリプタを使用して、USB インターフェイス (単一または複合 USB デバイス) の任意の組み合わせに対して USB API スタックを自動的に構成
- MSP430 USB フィールド・ファームウェア・アップデート・スタータ・プロジェクト

  PC から USB 経由で MSP430 のフィールドでファームウェアを更新するための Windows Visual Studio Express プロジェクト

C 関連資料

1. 『MSP430 USB API スタック・プログラマ・ガイド』(API スタックのダウンロードに付属。https://www.tij.co.jp/tool/MSP430USBDEVPACK にある MSP430 USB 開発パッケージへのリンクを参照)
2. HID に関する Microsoft Developers Network のリファレンス:http://msdn.microsoft.com/en-us/library/dd446410.aspx
3. サポートが必要な場合は、http://www.tij.co.jp/msp430 にアクセスしてサポート・オプションを参照。

C 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

Changes from Revision * (January 2011) to Revision A (February 2023)	Page
• 文書全体にわたって表、図、相互参照の採番方法を更新。	2
• リンクを更新	3
• リンクを更新	5
• 1 のリンクを更新。	17

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス· デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件、または ti.com やかかる TI 製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。

お客様がいかなる追加条項または代替条項を提案した場合でも、TI はそれらに異議を唱え、拒否します。

郵送先住所：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265