Application Note

MSP430™ USB HID Windows API 编程人员指南

William Goh, Keith Quiring *MSP430 Applications*

摘要

API 使用数据结构来包含与 HID 器件（在 hiddevice.h 中定义）相关的所有信息（请参阅表 A-1）。该结构的一个实例会被传递到涉及该器件的任何 API 调用中。

表 A-1. strHidDevice 结构定义

字段	说明	数据源
HANDLE hndHidDevice	该 HID 器件的句柄。	由 HID_Open() 写入。在 HID_Close() 期间设置为 INVALID_HANDLE_VALUE。
BOOL bDeviceOpen	布尔值，表示器件已打开，并且 hndHidDevice 中的句柄有效。	由 HID_Open() 和 HID_Close() 写入。
UINT uGetReportTimeout	读取数据的超时值，以毫秒为单位。这是 HID_ReadFile() 在数据读取操作中等待器件提供任何给定报告的时间长度。	必须由应用程序进行初始化。演示应用程序在 UsbAppDlg.cpp 中执行此操作。
UINT uSetReportTimeout	写入数据的超时值，以毫秒为单位。这是 HID_WriteFile() 在数据发送操作中等待器件接收任何给定报告的时间长度。	必须由应用程序进行初始化。演示应用程序在 UsbAppDlg.cpp 中执行此操作。
OVERLAPPED oRead	异步 I/O 结构	由 HID_Init() 初始化
OVERLAPPED oWrite	异步 I/O 结构	由 HID_Init() 初始化
WORD wInReportBufferLength	输入报告的最大长度。	由 HID_Init() 初始化，派生自器件的 USB 描述符。
WORD wOutReportBufferLength	输出报告的最大长度。	由 HID_Init() 初始化，派生自器件的 USB 描述符。
BYTE inBuffer[256]	接收传入数据的缓冲区。	由 by HID_readFile() 写入，由应用程序读取。
WORD inBufferUsed	inBuffer[] 中的字节数。	由 by HID_readFile() 写入，由应用程序读取。

运行 MSP430 USB API 堆栈并创建 HID 数据管道接口的 USB 器件使用表 B-1 中所示格式的 HID 报告。这在物理 USB 器件中的 USB 描述符内定义。此 Windows API 使用相同的格式，并且实际上假设 HID 器件使用此格式。

表 B-1. HID 数据管道报告结构

字段	尺寸
报告 ID	1 字节 (0x3F)
尺寸	1 字节
数据	N-2 个字节，其中 N 是 USB 数据包的有效载荷大小（即 USB API 堆栈内 descriptors.h 文件中定义的 MAX_PACKET_SIZE 值）。

1 MSP430™ USB HID Windows API 编程人员指南

内容

1 MSP430™ USB HID Windows API 编程人员指南 2
2 引言 3
3 实现 4
3.1 概述 4
3.2 文件组织 5
3.3 系统要求 5
3.4 MSP430 USB API 堆栈 5
3.5 Windows 如何将物理 USB HID 器件映射到主机应用程序 5
3.6 在系统上查找特定 HID 器件/接口并将其打开 7
3.7 发送/接收数据 7
3.8 检测 HID 器件的动态连接/断开 7
4 函数调用参考 9
4.1 器件连接管理和初始化调用 9
4.2 发送/接收数据 11
4.3 即插即用管理 13
5 演示应用程序 14
6 MSP430 USB 工具套件 16
C 参考文献 17
C 修订历史记录 17

2 引言

USB 人机接口器件 (HID) 类是不同操作系统之间支持最广泛的器件类之一。虽然 HID 最初是为鼠标和键盘开发的，但它在通用用途方面具有许多优势。

通用 USB 应用的一个常见选择是虚拟 COM 端口，该端口可以使用通信器件类 (CDC) 来实现。COM 端口易于在主机平台上实现，并且为开发人员所熟知。缺点是 Windows 主机的 USB 虚拟 COM 端口解决方案需要最终用户完成器件安装过程。除了有点麻烦外，如果用户选择了错误的选项，这个过程可能会出错。此外，企业环境中的一些用户没有管理权限，因此，如果没有网络管理员的帮助，他们就无法进行器件安装。相比之下，HID 器件会以静默方式加载，绕过了所有这些问题。

HID 确实有一些缺点。与虚拟 COM 端口相比，许多软件工程师对其用法并不熟悉。HID 依靠“报告”来传输数据，并且这些复杂的格式通常不会为通用应用提供实际价值。基本 HID 实现的带宽限制为 64KB/s。

为了弥补这些缺点，TI 提供了一个 Windows HID API 和演示工程。它经过简化，无需创建 HID 报告，就能与可通过 MSP430 HID API 堆栈实现的*数据管道接口*配合使用。它支持复合 HID 器件，让开发人员可以轻松地将多个 HID 接口合并到一个 USB 器件内，从而使带宽倍增。

MSP430 HID API 堆栈和 Windows HID API 组合在一起构成了一个端到端解决方案，对于许多通用应用而言，该解决方案比虚拟 COM 端口更具优势。

有关 MSP430 MCU 的 USB 支持软件，请参阅 上的 MSP430 USB 开发套件。

3 实现

下文将此 Windows MSP430 HID API 简称为 *Windows API*。

3.1 概述

图 3-1 显示了此 API 在 Windows 软件栈内的位置以及与 MSP430 器件上 USB HID API 的关系。

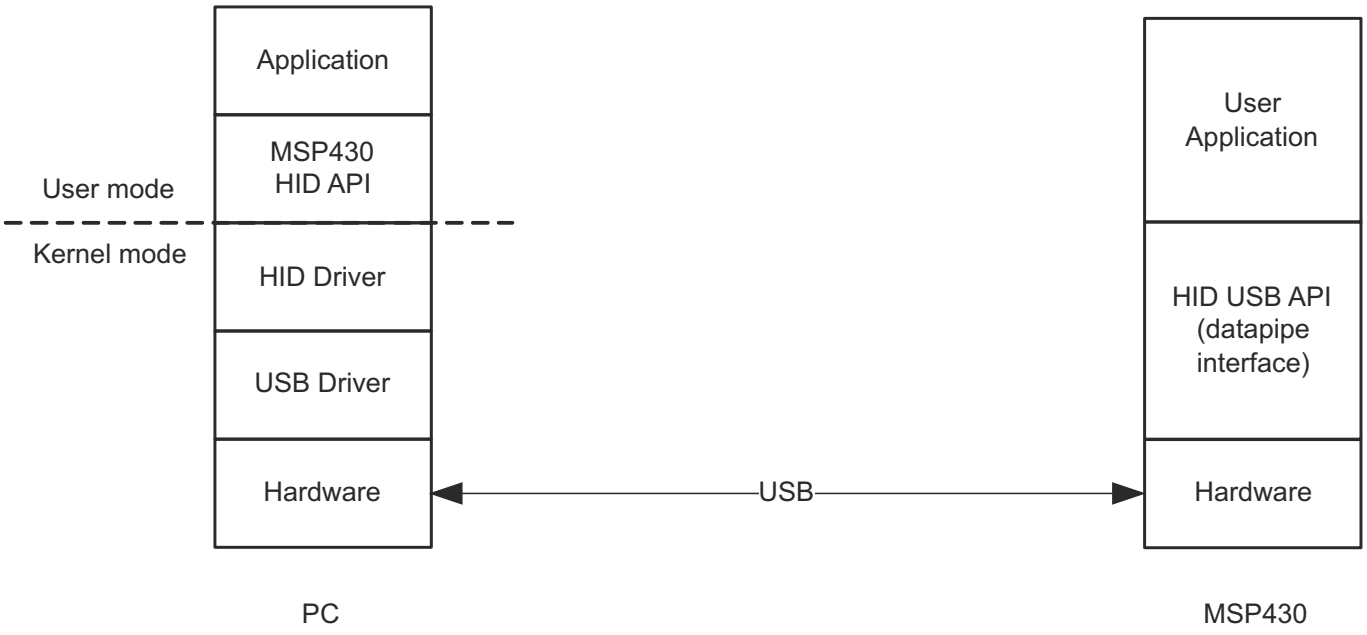

图 3-1. Windows 和 MSP430 软件栈

该软件旨在与 MSP430 的 USB HID API 堆栈固件（特别是该 API 堆栈提供的数据管道接口）配合使用。该 API 堆栈与 Windows API 结合使用，使软件开发人员与 HID 报告分离开来，让开发人员可以将数据接口视为未格式化的数据流，类似于 COM 端口。

为了实现这种分离，数据管道接口实现了一个非常简单的 HID 报告结构，开发人员无需修改该结构。开发人员使用简单的命令来发送或接收任何大小的数据块，而 API 使用报告作为数据包来传输此数据。（有关数据管道接口的更多信息，请参阅 *MSP430 USB CDC/HID API 堆栈编程人员指南*。）

3.2 文件组织

表 3-1 显示了 API 和演示应用程序中的文件。

表 3-1. 源代码文件

文件	说明
演示应用程序	
*.cpp	各种 C++ 文件
API	
hiddevice.c	API 函数调用和支持函数调用的代码实现。
hiddevice.h	可供应用程序使用的 API 函数调用的定义。

3.3 系统要求

分发的 Windows API 是使用 Microsoft 基础类 (MFC) 库为 Visual Studio 2008 编写的。因此，它可用于各种 Windows 平台，包括所有类型的 Windows Vista/XP/7。该 API 用 C 语言编写，而演示应用程序用 C++ 语言编写。Visual Studio 的 Express 版本足以编译该 API。但是，编译演示应用程序需要完整版本。

该 API 还需要安装 Windows 驱动程序套件 (WDK)。这可以从 Microsoft 免费获得。建议使用版本 7，因为相对于以前的版本，它可以提高速度。

要将该 API 链接到 WDK，请执行以下操作：

1. 从 Microsoft 下载并安装最新版本的 WDK。
2. 右键点击 HidDevice.c 文件 → “属性”
3. 导航至“C/C++”→“常规”→“附加包含目录”
4. 包括：
   a. c:\<WinDDK 安装位置>\<构建版本>\inc\api\
   b. c:\<WinDDK 安装位置>\<构建版本>\inc\crt\
5. 导航至“链接器”→“常规”→“附加包含目录”
6. 添加 c:\<WinDDK 安装位置>\<构建版本>\lib\wxp\i386

3.4 MSP430 USB API 堆栈

MSP430 USB API 旨在让开发者无需详细了解 USB，即可轻松创建 USB 器件。它随附了 MSP430 USB 描述符工具，后者能够自动完成以下操作：

- 针对 USB 接口的任意组合（单个或复合）配置 API 堆栈
- 创建首次就奏效的 USB 描述符

适用于 HID 的 API 支持两种 HID 实现。一种是针对传统 HID 器件，这需要详细了解 HID 报告。另一种是针对数据管道 HID 器件，这是 TI 的一种将 HID 用作简单数据载体的方法。这就让设计人员无需创建 HID 报告。

对于 MSP430 应用程序，HID 数据管道接口看起来和感觉都非常像 COM 端口：

- 它相对未格式化，允许设计人员应用所选的格式。
- 它可以处理任何大小的数据块（不受 USB 数据包大小的限制）。

实际上，适用于 HID 数据管道器件的 API 与适用于 CDC（用于在主机上生成虚拟 COM 端口）的 API 几乎完全相同。

有关 MSP430 USB API 堆栈的更多信息，请参阅 MSPUSBDEVPACK 文档，网址为 https://www.ti.com.cn/tool/cn/MSP430USBDEVPACK。

3.5 Windows 如何将物理 USB HID 器件映射到主机应用程序

每个 USB 器件都包含一个供应商 ID (VID) 和一个产品 ID (PID)。USB 主机使用 VID/PID 组合来识别产品类型。按理说，来自给定 VID 供应商且具有给定 PID 的所有产品在功能上都是相同的。主机会将具有该 VID/PID 的所有器件关联到特定的 USB 器件类别，例如 HID 或 CDC。每次遇到此 VID/PID 时，它都会加载该特定的驱动程序。

在 Windows HID 驱动程序之上，情况看起来略有不同。HID 驱动程序会维护一个“HID 器件”列表，其中包含总线的所有“HID 器件”，且器件可通过通用索引进行选择，并使该列表可供其上面的主机应用程序使用。此列表中的“HID 器件”是逻辑实体，而不是物理 USB 器件。这些逻辑实体不一定直接映射到总线上的物理器件。

例如，可能会向主机应用程序报告多个具有给定 VID/PID 的“HID 器件”。这可能意味着该主机上连接了多个给定产品类型的物理器件；也可能是一个复合器件具有多个 HID 接口（可被视为逻辑器件）。此外，如果有 4 个以上的“HID 器件”，则可能同时存在这两个情况，如图 3-2 所示。

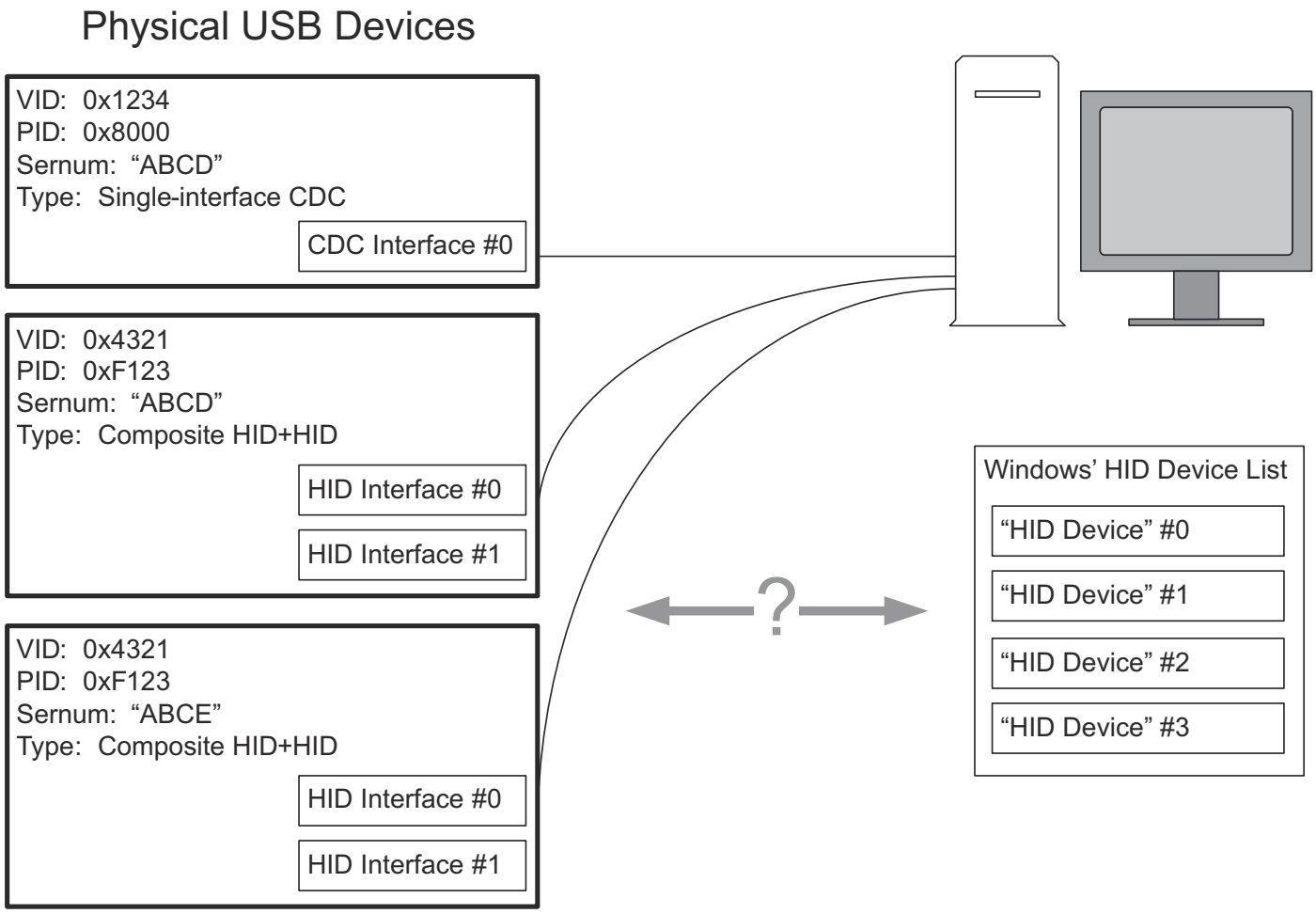

图 3-2. Windows 系统上“HID 接口”到“HID 器件”的映射

在此图中，Windows 应用程序无法立即知道这些“HID 器件”在现实世界中代表什么。

如果主机应用程序想要在这种多器件情况下正常工作，或者更进一步，想要充分利用物理器件上的多个 HID 数据管道，则需要进一步进行探索。它需要从每个 HID 器件收集的一个重要信息是序列号（在图 3-2 中标记为“sernum”）。USB 器件可以选择报告序列号，序列号用于标识一个唯一的物理设备，使其与包含相同 VID/PID 的所有其他物理设备区分开来。（如果选择使用描述符工具来实现这一点，MSP430 USB API 堆栈可以自动报告序列号。）

在获得总线上每个 HID 器件的 VID、PID 和序列号后，主机应用程序就可以开始了解总线上的情况。如果它找到两个具有给定 VID/PID 的 HID 器件，并且其序列号也相同，那么它就知道这是一个具有两个 HID 接口的复合 HID 器件。相反，如果这两个 HID 器件具有不同的序列号，应用程序就知道这是两个单接口 HID 物理器件。

借助这些技术，开发人员可以设计具有多个 HID 接口的器件，将每个接口用作特定类型信息的数据管道，相当于具有多个 COM 端口。除了其他优点外，这还有助于克服单个 HID 接口的带宽限制 (64kbps)。一个 MSP430 上可以并行运行多达 8 个 HID 接口，从而可以提供 512kbps 的带宽。

需要补充的是，如果 Windows 在初始枚举期间通过 USB 描述符识别出此器件是鼠标或键盘，则 Windows 本身将成为该 HID 器件的“主机应用程序”。它会发出报告请求并使用数据来控制屏幕上的鼠标或输入数据以进行文本输入。

3.6 在系统上查找特定 HID 器件/接口并将其打开

本文档的其余部分使用了以下术语：

- *HID 器件*：Windows HID 器件列表中的器件
- *物理 USB 器件*：一个实际的 USB 硬件，在 USB 总线上具有自己的唯一地址
- *HID 接口*：在物理 USB 器件的 USB 描述符中声明的接口。它可能是器件上唯一的接口，也可能是复合 USB 器件中的多个接口之一。HID 接口与 Windows 中的 HID 器件相关联。

主机应用程序开发人员在设置访问 USB 器件时，通常知道与其关联的器件的 VID/PID 对，因为两者通常由同一方设计。因此，该过程从调用 HID_GetSerNums() 开始。此函数接收 VID/PID 作为参数并返回一个序列号列表，其中每个序列号均代表总线上与 VID/PID 关联的一个物理 USB 器件。应用程序可以选择仅与其中一个物理器件交互，也可以为其提供与多个器件交互的功能。

应用程序开发人员通常还知道这些物理器件每个具有多少个接口。如果不知道这一点，可以调用函数 HID_GetNumOfInterfaces()。当调用 HID_Open() 来打开器件时，必须知道物理器件上的 HID 接口总数。

当使用 VID、PID 和序列号来筛选 HID 器件列表时，该列表中剩下的就是由这些参数描述的物理 USB 器件中的所有 HID 接口。此列表通常有一个或两个，但也可能像 MSP430 一样有多达 8 个。此列表的顺序与 HID 接口在物理器件的 USB 描述符中列出的顺序相同。这样，使用 API 堆栈的主机应用程序和 MSP430 应用程序可以相互“查找”，形成完整的数据链路。

知道 USB 器件内所需 HID 接口的 VID/PID、序列号和索引后，应用程序可以调用 HID_Open() 以打开与物理 USB 器件上 HID 接口的连接。

当应用程序使用完器件时（例如退出程序时），应用程序应使用 HID_Close() 关闭所有打开的 HID 器件。

3.7 发送/接收数据

在 HID 接口打开后，并且假定此接口设置为 MSP430 HID API 上的“数据管道”接口，可以使用 HID_writeFile() 和 HID_readFile() 调用来通过该 API 发送/接收任何大小的数据块。这些调用会自动处理打包成一个或多个 HID 报告。

如果是实现“传统”HID 器件而不是“数据管道”器件，这意味着已自定义报告格式并且在 MSP430 HID 应用程序中使用了 HID 传统函数调用，那么仍然可以使用此 API 作为示例，但需要自定义 API 代码。

3.8 检测 HID 器件的动态连接/断开

与传统的 COM 端口不同，USB 应用必须注意，用户可以随时断开（或连接）器件。这是 USB 的动态“即插即用”方面。Windows 通过在 MFC 库中实现的通知来处理此功能。这发生在 API 之外，因为通知直接发送到应用程序的窗口对象；但是，当接收到这些通知时，应用程序可以调用专为此目的提供的 API 处理程序。演示应用程序展示了如何做到这一点。

以下是应用程序处理这些通知所需的步骤：

1. 在开始时，应用程序应寄存其窗口，以使用 HID_RegisterForDeviceNotification() 接收器件更改通知。
2. 将 ON_WM_DEVICECHANGE 置于窗口的消息映射中。（确保这不在 AFX_MSG_MAP 注释部分中。）
3. 写入函数 OnDeviceChange() 作为窗口的公共函数。MFC 框架将调用此消息处理程序以响应 ON_WM_DEVICECHANGE 通知。

函数的定义应如下所示：

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

此函数需要确定是否从系统中移除了此应用程序使用的任何“HID 器件”。 为此，它可以为每个打开的 HID 器件调用 HID_IsDeviceAffected()，以确定它们是否已被移除。

退出应用程序时，应用程序应确保使用 HID_UnregisterForDeviceNotification() 取消寄存。

4 函数调用参考

4.1 器件连接管理和初始化调用

表 4-1 总结了器件连接管理调用。

表 4-1. 器件连接管理调用摘要

功能	说明
VOID HID_Init()	初始化用于存储 HID 器件/接口信息的 HID 器件数据结构。
DWORD HID_GetNumOfInterfaces()	返回系统上与特定 VID/PID 和序列号关联的 HID 器件数量。
BYTE HID_Open()	打开特定器件的句柄。
BYTE HID_Close()	关闭特定器件的句柄。
BYTE HID_GetSerNums()	获取与特定 VID/PID 关联的序列号。

4.1.1 VOID HID_Init(struct strHidDevice* pstrHidDevice)

说明

此函数用于初始化 pstrHidDevice。在使用之前，必须为每个 strHidDevice 结构实例调用一次该函数。

参数

表 4-2. HID_Init() 的参数

strHidDevice* pstrHidDevice	要初始化的结构。

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

说明

返回系统上与 vid 和 pid 组合关联的物理 USB 器件的数量。（这些器件可能是 HID 类型或其他类型。）如果未连接任何器件，则返回 0。如果连接了相关器件，则传递的 strTrackSerialNumbers 结构将填充与找到的物理器件相对应的序列号，该函数将返回列表中的序列号总数。

如果返回的结果不止一个，则表明有多个具有相应 vid/pid 组合的物理器件。

参数

表 4-3. HID_GetSerNums() 的参数

WORD vid	要查找的器件的 16 位供应商 ID。
WORD pid	要查找的器件的 16 位产品 ID。
struct strTrackSerialNumbers *serialNumList	包含与 vid/pid 关联的序列号列表的结构。 该函数会使用找到的序列号填充该结构。
返回值	0：未连接具有此 VID/PID 的物理 USB 器件时。 非 0：系统上具有此 VID/PID 的物理 USB 器件的数量。

表 4-4. strTrackSerialNumbers 结构定义

字段	说明
DWORD deviceNum	表示物理 USB 器件的索引号。
char serialNum[SERNUM_LEN];	包含已检测到物理器件的序列号的字符串。

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

说明

返回由 vid 和 pid 标识的每个物理 USB 器件中存在的 HID 接口数量。如果未连接此类器件，则该函数返回零。

numSerNums 是系统上与此 VID/PID 关联的物理器件的数量。此函数通过扫描 Windows HID 器件列表（即在系统上注册的逻辑 HID 器件）、统计与此 VID/PID 关联的器件，然后将其除以 numSerNums 来实现其目标。numSerNums 可以通过调用 HID_GetSerNums() 来找到。

单接口 HID 器件始终会返回值 1。

参数

表 4-5. HID_GetNumOfInterfaces() 的参数

UINT vid	要查找的器件的 16 位供应商 ID。
UINT pid	要查找的器件的 16 位产品 ID。
DWORD numSerNums	具有此 VID/PID 的已连接物理器件的总数
返回值	0：未连接具有此 VID/PID 的 USB 器件时。 非 0：系统上具有此 VID/PID 的 USB 器件的数量

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

说明

尝试在系统上与此 VID/PID 和序列号关联的器件列表中查找索引为 deviceIndex 的器件。如果找到，它会打开该器件的句柄，将其存储在 pstrHidDevice 结构中，并加载该结构中的其他字段。

为了实现其目标，HID_Open() 需要另外两个参数。一个参数是 totalDevNum，这是 vid 和 pid 描述的器件上的 HID 接口数量。开发人员通常已经知道这个数值。如果不知道，则可以调用 HID_GetNumOfInterfaces()。另一个必需的参数是 totalSerNum，它是系统上与此 VID/PID 匹配的序列号（物理器件）总数。

deviceIndex 是一个从 0 到 totalDevNum-1 的数字。使用 vid、pid 和 serialNumber 筛选 Windows HID 器件列表后，该列表实际上会简化为仅包含 HID 器件，这些器件代表着特定物理 USB 器件中包含的 HID 接口。剩下的就是选择其中一个接口，这就是 deviceIndex 的功能。这些 HID 器件的顺序（从 0 到 totalDevNum）与在器件的 USB 描述符中声明 HID 接口的顺序相同（请参阅 节 3.5）。

如果 vid 和 pid 描述的器件中只有一个 HID 接口，deviceIndex 就为 0。

该函数会假定此器件实际上是 HID 器件，而不是其他类型的 USB 器件。

参数

表 4-6. HID_Open() 的参数

strHidDevice* pstrHidDevice	包含新打开的器件的结构。
WORD vid	要打开的器件的 16 位供应商 ID。
WORD pid	要打开的器件的 16 位产品 ID。
DWORD deviceIndex	此 VID/PID 和序列号所对应可用器件内的索引。
char serialNumber[SERNUM_LEN]	要查找的器件的序列号的字符串，大小为 SERNUM_LEN (40)。
DWORD totalDevNum	此 VID/PID 和序列号描述的物理器件上的 HID 接口总数。
DWORD totalSerNum	此 VID/PID 对应的可用物理 USB 器件总数
返回值	HID_DEVICE_SUCCESS。已找到并打开器件。 HID_DEVICE_ALREADY_OPENED。已打开器件。 HID_DEVICE_NOT_FOUND。找不到此 VID/PID/索引标识的器件。

4.1.5 BYTE HID_Close(struct strHidDevice* pstrHidDevice)

说明

关闭与此 pstrHidDevice 结构关联的器件，并指示系统关闭该句柄。应用程序应始终在使用完器件时执行此操作。

参数

表 4-7. HID_Close() 的参数

strHidDevice* pstrHidDevice	包含新打开的器件的结构。

表 4-7. HID_Close() 的参数 (continued)

返回值	HID_DEVICE_HANDLE_ERROR。器件句柄无效。 HID_DEVICE_SUCCESS。器件句柄已成功关闭。 HID_DEVICE_NOT_OPENED。尝试了关闭以前未打开的器件句柄。

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

说明

将与 pstrHidDevice 关联的器件版本号放入 VersionNumber 中。这是物理 USB 器件在器件描述符的 bcdDevice 字段中报告的值。

参数

表 4-8. HID_GetVersionNumber() 的参数

strHidDevice* pstrHidDevice	包含 HID 器件信息的结构。
USHORT* VersionNumber	器件版本号。
返回值	HID_DEVICE_HANDLE_ERROR。器件句柄无效。 HID_DEVICE_SUCCESS

4.2 发送/接收数据

表 4-9 中显示的调用与发送/接收数据有关。

表 4-9. 器件连接管理调用摘要

功能	说明
BYTE HID_WriteFile()	将数据发送到器件。
BYTE HID_ReadFile()	从器件接收数据。

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

说明

将缓冲区中存储的 bufferSize 个字节的数据发送到 pstrHidDevice 指定的 HID 器件。

此函数将 HID 报告用作“数据包”发送数据。如果任何发送 HID 报告的尝试超时，该函数将返回 HID_DEVICE_TRANSFER_TIMEOUT。

除了 bufferSize 以 DWORD 值形式施加的限制外，可以发送的字节数没有内在限制。打包由系统自动处理。

参数

表 4-10. HID_WriteFile() 的参数

strHidDevice* pstrHidDevice	包含 HID 器件信息的结构。
BYTE* buffer	要发送的数据数组
DWORD bufferSize	要发送的字节数，从地址缓冲区开始。
DWORD bytesSent	实际发送的字节数（发生错误时）
返回值	HID_DEVICE_NOT_OPENED。HID 器件无法打开。 HID_DEVICE_TRANSFER_TIMEOUT。报告请求超时。 HID_DEVICE_TRANSFER_FAILED。传输因不明原因而失败。 HID_DEVICE_SUCCESS。所有数据均已成功发送。

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

说明

从 pStrHidDevice 指定的 HID 器件获取 bufferSize 个字节的有效负载数据，并将其存储在缓冲区中。它通过根据需要读取尽可能多的 HID 报告，直到返回 bufferSize 个字节，来实现这一点。

如果该函数返回 HID_DEVICE_SUCCESS，则 bytesReturned 将等于 bufferSize。只有出现其他返回代码之一时，bytesReturned 才会小于 bufferSize。

此函数通过将 HID 报告用作“数据包”来接收数据。根据 pStrHidDevice.uGetReportTimeout 的值，如果从 USB 器件获取数据包的任何尝试超时，该函数会返回 HID_DEVICE_TRANSFER_TIMEOUT。

除了 bufferSize 以 DWORD 值形式施加的限制外，可以接收的字节数没有内在限制。打包由系统自动处理。

参数

表 4-11. HID_ReadFile() 的参数

strHidDevice* pstrHidDevice	包含 HID 器件信息的结构。
BYTE* buffer	包含接收数据的数组
DWORD bufferSize	缓冲区大小，表示预期接收的最大字节数
DWORD* bytesReturned	实际接收到的字节数（发生错误时）
返回值	HID_DEVICE_NOT_OPENED。HID 器件无法打开。 HID_DEVICE_TRANSFER_TIMEOUT。报告请求超时。 HID_DEVICE_TRANSFER_FAILED。传输因不明原因而失败。 HID_DEVICE_SUCCESS。接收到正确数量的字节。

4.3 即插即用管理

表 4-12 中显示的调用可以帮助应用程序管理 USB 器件的动态连接/分离。

表 4-12. 器件连接管理调用摘要

功能	说明
BYTE HID_RegisterForDeviceNotification()	将数据发送到器件。
BYTE HID_UnregisterForDeviceNotification()	从器件接收数据。
BYTE HID_IsDeviceAffected()	确定特定器件是否仍在系统上。

4.3.1 BYTE HID_RegisterForDeviceNotification(HWND hWnd, HDEVNOTIFY* diNotifyHandle)

说明

寄存句柄 hWnd 指向的窗口，以便在器件添加到系统中或从系统中移除时接收通知。应采取步骤来使应用程序能够响应系统通知，如节 3.6 中所述。

diNotifyHandle 中会返回器件通知句柄。该句柄应由应用程序存储，以便在调用 HID_UnregisterForDeviceNotification() 时使用。

参数

表 4-13. HID_RegisterForDeviceNotification() 的参数

HWND hWnd	主窗口的句柄
HDEVNOTIFY* diNotifyHandle	器件通知句柄。
返回值	HID_DEVICE_HANDLE_ERROR。器件句柄无效。 HID_DEVICE_SUCCESS

4.3.2 BYTE HID_UnregisterForDeviceNotification(HDEVNOTIFY* diNotifyHandle)

说明

取消寄存器件通知句柄 diNotifyHandle 指向的窗口，该句柄由 HID_RegisterForDeviceNotification 返回。调用此函数后，当器件添加到系统或从系统中移除时，Windows 将不再通知应用程序。无论句柄是否有效，该函数都会返回 true。

参数

表 4-14. HID_UnregisterForDeviceNotification() 的参数

HDEVNOTIFY* diNotifyHandle	器件通知句柄。
返回值	HID_DEVICE_SUCCESS

4.3.3 BOOL IsDeviceAffected(struct strHidDevice* pstrHidDevice)

说明

应用程序应调用此函数来响应从系统接收到的 ON_WM_DEVICECHANGE 通知。它指示事件是否引用了与 pstrHidDevice 关联的器件。

参数

表 4-15. HID_IsThisDeviceAffected() 的参数

strHidDevice* pstrHidDevice	包含 HID 器件信息的结构。
返回值	TRUE 或 FALSE

5 演示应用程序

这里提供了一个简单的演示应用程序，展示了如何使用该 API。该应用程序类似于 Hyperterminal 等 COM 端口终端应用程序。它会以类似的方式发送和接收数据块，只不过使用的是 HID 接口。

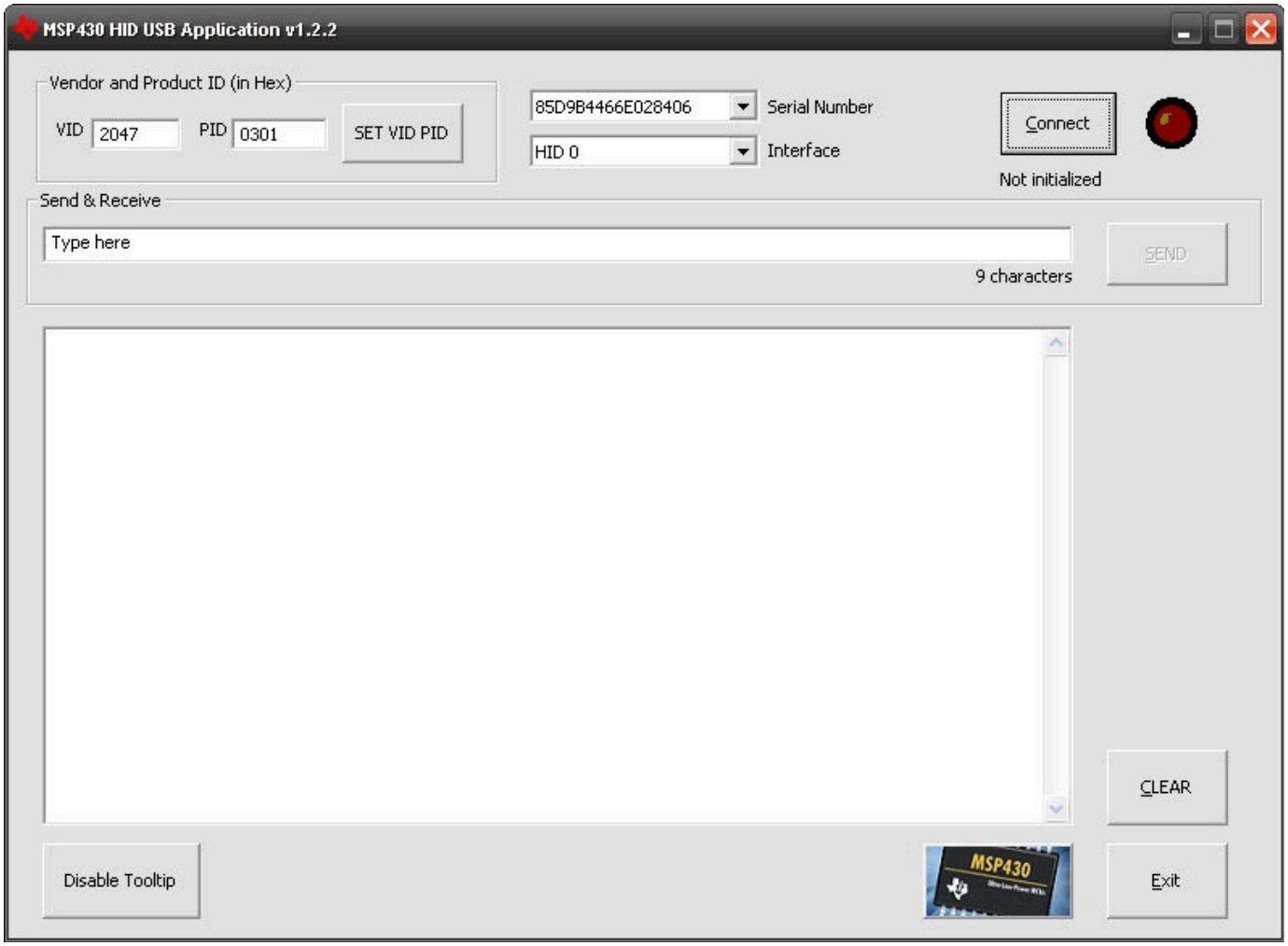

图 5-1. 演示应用程序窗口

要运行该程序，请连接一个运行 HID API 堆栈的 MSP430 并使用数据管道函数调用。API 堆栈分发中提供了几个这样的示例，并提供了单接口 HID、CDC+HID 复合器件和 HID+HID 复合器件例子。所有这些都可以与此演示应用程序配合使用。

GUI 中的 VID 和 PID 字段用于选择应用程序要搜索的 VID/PID。输入这些值后，按下“Set VID/PID”按钮。

设置 VID/PID 后，应用程序会确定连接到系统的物理器件中有多少器件与此 VID/PID 匹配。它会查找相关器件的序列号并将其发布在“Serial Number”组合框中。它会自动选择列表中的第一个序列号，并继续确定该器件上存在的 HID 接口数量。它会将这些接口载入“Interface”组合框中，其中每个接口对应一个字符串：“HID 0”、“HID 1”等等。如果找到了任何 HID 接口，则默认选择“HID 0”。

按“Connect”按钮。这将打开与相应 USB 器件的数据连接。如果成功，按钮下方的字符串会指示成功。如果失败，请尝试以下操作：

- 检查 Windows 器件管理器以确保该器件已经在系统上成功枚举为 HID 器件
- 确保器件配备了 MSP430 HID API 堆栈并运行 HID 数据管道应用程序
- 确保 descriptors.h 中显示的 VID/PID 与输入到演示应用程序字段中的值相同，并且按下了“Set VID/PID”按钮。

连接初始化后，可以通过输入文本并按“Send”来将数据发送到器件。应用程序只要从器件接收到数据，便会显示在大文本字段中。接收窗口可以使用“Clear”按钮清除。

如果在访问另一个器件时从总线上移除了该 HID 器件，或者添加了具有此 VID/PID 的另一个 HID 器件，应用程序会自动更新下拉菜单列表。

6 MSP430 USB 工具套件

此 API 是 TI 提供的一整套工具的一部分，旨在简化 MSP430 上的 USB，其中包括：

- MSP430 USB API 堆栈
  - CDC（通信器件类）
  - HID（人机接口器件类）
  - MSC（大容量存储类）
- MSP430 USB 描述符工具

  针对 USB 接口的任意组合（单个或复合 USB 器件）自动配置 USB API 堆栈，其中的 USB 描述符首次就奏效
- MSP430 USB 现场固件更新入门工程

  Windows Visual Studio Express 工程，用于使用 PC 通过 USB 现场更新 MSP430 上的固件

C 参考文献

1. *MSP430 USB API 堆栈编程人员指南* (随 API 堆栈下载；请参阅 MSP430 USB 开发套件的链接，位于 https://www.ti.com.cn/tool/cn/MSP430USBDEVPACK)
2. 适用于 HID 的 Microsoft 开发者网络参考：http://msdn.microsoft.com/en-us/library/dd446410.aspx
3. 如果需要支持，请访问 http://www.ti.com/msp430 并查看支持选项。

C 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (January 2011) to Revision A (February 2023)	Page
• 更新了整个文档中的表格、图和交叉参考的编号格式。	2
• 更新了链接	5
• 更新了 1 的链接。	17

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265