USB PC Drivers Based on Generic HID Class

- Supported by Windows 98® SE or later
- Full Duplex Communication
- Send Commands Through the EP 0
- Dynamic Link Library Supported by any Compiler: VC++, JAVA, VB...
- Auto-detection of device for VC++ application
- Point-to-Point Communication

1. Introduction
This application note describes how to integrate the USB HID DLL in your application. The provided examples are based on VC++ and JAVA compilers, however the DLL can be used with any compiler (VB, Delphi, LabView...).

Simple code examples that demonstrate different types of implementation are given.
2. DLL functions

As specified in the USB HID specification, the Generic HID application uses two reports (IN/OUT) to send and receive data. The length of these reports is assigned in the firmware and automatically detected by the DLL following the firmware setting (please refer to the USB Generic HID Implementation application note to see how to modify these values if needed).

Please note that this DLL allows you to communicate with one Generic HID device and only one device at a time. You cannot manage several devices at the same time using this DLL.

Figure 2-1. DLL functions

2.1 findHidDevice

This function (BOOLEAN) allows to find the Generic HID device using the vendor ID (VID)/the product ID (PID) and open a handle if the device is connected.

Input

cost UINT VendorID: this is the vendor ID
const UINT ProductID: this is the product ID

Output

FALSE: if the device is not found. More information can be found using GetLastError().

GetLastError will return:

Find the HID Device
findHidDevice(VID, PID)

Send data to the device
writeData("buff")

Send commands to the device
setFeature("buff")

Get the IN report length
getInputReportLength()

Get the OUT report length
getOutputReportLength()

Get the Feature report length
getFeatureputReportLength()

Close all handles
closeDevice()
ERROR_USB_DEVICE_NOT_FOUND if Device is cannot be found.

ERROR_USB_DEVICE_NO_CAPABILITIES if device found but capabilities cannot be retrieved.

TRUE: if the connection has succeeded and the handle is opened.

2.2 closeDevice

Closes the communication with the USB device and all related handles.

2.3 writeData

This function (BOOLEAN) sends data to the device (OUT data). The maximum data length supported by this function must be lower or equal to the value given by the function getOutputReportLength (see Section 2.9 on page 5).

If the data length exceeds the maximum length specified in the firmware, the user has to send data in several packets.

When data length is lower than the maximum length, this function will complete the remaining bytes with zero (0x00).

**Input**

UCHAR* buffer: pointer of the packet to be sent.

2.3.1 Output

FALSE: if data transmission fails. GetLastError() will return ERROR_WRITE_FAULT code.

TRUE: if the packet was successfully transferred.

2.4 readData

This function (BOOLEAN) read the data packets sent by the device (IN data). To avoid data loss this function should be called in continuous mode (using a thread or a timer).

**Input**

UCHAR* buffer: Pointer to the buffer which will contain the received packet.

The buffer must have the length of the IN report given by the getInputReportLength function (see Section 2.10 on page 5).

2.4.1 Output

FALSE: if no data is available.

TRUE: if data are received and stored in the buffer.

2.5 setFeature

This function (BOOLEAN) allows the user to send a command data to control the HID device (i.e.: Start the bootloader, start a new task...). Data will be transmitted over the endpoint 0 as a “SetReport” request (Refer to the HID Specification for further information). The endpoints IN and OUT will be used for the applicative raw data transfer only.
The data length is fixed by the firmware and can be obtained using the function getFeatureReportLength (please refer to the Section 2.11 on page 5). The data length must not exceed the length returned by getFeatureReportLength function.

**Input**

UCHAR* buffer: Pointer to the buffer which contains the received packet.

**Output**

FALSE: if data transmission fails.

TRUE: if data are well transferred.

2.6 hidRegisterDeviceNotification

Please note that this function can be used only with VC++ project.

This function notifies the application if a new plug & play device has been plugged or unplugged.

**Input**

HWND hWnd - Handle to a window.

**Output**

FALSE: if the function fails. To get extended error information, call GetLastError.

TRUE: if the function succeeds.

2.7 hidUnregisterDeviceNotification

Please note that this function can be used only with VC++ project.

This function closes the specified device notification handle.

**Input**

HWND hWnd - Handle to a window.

**Output**

FALSE: if the function fails. To get extended error information, call GetLastError.

TRUE: if the function succeeds.

2.8 isMyDeviceNotification

Please note that this function can be used only with VC++ project.

This function allows to check if the new device (plugged or unplugged) notified by “hidRegisterDeviceNotification” is the used HID device or not.

**Input**

DWORD dwData, the value given by OnDeviceChange 2nd parameters

**Output**

TRUE: if the device connected/disconnected is the used HID device

FALSE: if this is another device
2.9 getOutputReportLength
This function allows the user to get the length of the OUT report (data packet sent from the PC to the device). This value is specified in the firmware.

2.10 getInputReportLength
This function allows the user to get the length of the IN report (data packet sent from the device to the PC). This value is specified in the firmware.

2.11 getFeatureReportLength
This function allows the user to get the length of the Feature report (Control data packet sent from the PC to the device). This value is specified in the firmware.

3. PC demos

3.1 VC++ demo
The VC++ demo allows the user to see how to load the AtUsbHid.dll in a project, and also how to use the plug & play notification.

3.1.1 Load the DLL in Visual C++ Application
The file AtUsbHid.h provides the macros which help to load and use the functions present in the Atmel USB HID DLL.

When designing an application using the DLL you need to do the following:

- create a handler for the DLL: HINSTANCE hLib = NULL;
- Load the DLL using the function hLib = LoadLibrary(AT_USB_HID_DLL);
- Load each DLL functions using loadFuncPointers(hLib)

Once these steps have been performed without error, the DLL and its functions are loaded in your application and can be called using the macro DYNCALL(DllFunction()).

When the application is stopped, it is convenient to free the DLL from memory using the function FreeLibrary(hLib).

You must ensure that USB device handle has been closed before freeing the DLL from memory.

3.1.2 Using Automatic Device Connection/Disconnection Feature
The DLL provides the functions which allow the user to detect the connection/disconnection of the device.

To perform this feature you have to do the following actions:

Register your application to get device change notification using:

DYNCALL(hidRegisterDeviceNotification)((m_hWnd)).

Add the function ON_WM_DEVICECHANGE() in your Message Map application.

Creates a function called OnDeviceChange(UINT nEventType, DWORD dwData) which will be called each time a device status changes.

In the function OnDeviceChange, call the function DYNCALL(isMyDeviceNotification(dwData)) to know if the status of your device has changed (connected or disconnected).

(See code demo code in UsbHidDemoCodeDlg.cpp)
When exiting the application, it is convenient to unregister it using the function: `DYNDCALL(hidUnregisterDeviceNotification(m_hWnd))`.

### 3.1.3 Using readData

As data can be sent continuously by the device. It’s interesting to read data using a timer base function. This allows you to poll continuously the readData function.

To do so, you have to do the following:

Add the function `ON_WM_TIMER()` in your Message Map application.

Create a function `OnTimer(UINT nIDEvent)` which will call the function `DYNDCALL(readData(sbuffer))`.

Now you can set the Timer for a specified interval using `SetTimer(n,x,y)`; to call the readData function each `x` ms when your device is connected.

Kill the timer using `KillTimer(n)` when your device is disconnected.

### 3.1.4 User Interface

Hereunder is a screen shot of the provided demo. Please note that the default PID is related to one specific demo (Atmel demos which have a Generic HID interface do not have the same PID). You may have to modify this PID parameter to match with the device you are using (refer to the firmware or the device manager to get the VID/PID used by your demo)

**Figure 3-1.** VC++ based demo

Hereunder is the description of the GUI components:

- The **Vendor ID, Product ID** box are used to specify the VID/PID of the device.  
- **OK** button should be pushed once the VID/PID are correctly set.  
- **LED 1..LED4** button are used to switch ON/OFF the LEDs of the board.  
- **Firmware Upgrade** button allows the user to start the bootloader to upgrade the firmware through the USB interface (Refer to the bootloader datasheet for further details).
3.1.5 DOS demo

This demo gives a simple console application example. This demo uses a fixed VID/PID and has to be recompiled to modify these parameters. The device have to be connected and running with the Generic HID firmware before performing this console application.

Figure 3-2. DOS Interface

Note: This project can be compiled using the MinGw (www.mingw.org). The command line is:

```bash
mingw32-g++ -O2 -Wall UsbHidSmallDemoCode.cpp -o AtUsbHidMinGw.exe -I.
```

3.2 JAVA demo

The JAVA demo allows the user to see how to integrate the AtUsbHid.dll in a JAVA project. The interface between the `AtUsbHid.dll` and the JAVA is done through the package `AtUsbHidJni.jar`.

3.2.1 AtUsbHid.dll integration

To integrate the `AtUsbHid.dll` you have to follow the steps below:

- Add the following code in the import section of your JAVA file:

  ```java
  import com.atmel.atusbhidjni.AtUsbHidJni;
  ```

- Create a new object to use the DLL:

  ```java
  AtUsbHidJni usbDevice = new AtUsbHidJni();
  ```
• Load the DLL:
  
  `usbDevice.loadLibraryUsbHid();`

• Now, the DLL is ready for use. Please refer to the DLL functions section for further details regarding the DLL functionalities.

• Before exiting the application, it is important to unload the DLL:

  `usbDevice.UnloadloadLibraryUsbHid();`

• To compile the project, please add to the class path of the `AtUsbHidJni.jar` package:

  `JAVAc userhid.JAVA -classpath AtUsbHidJni.jar`

Note: Please refer to the HTML documentation provided with the DLL package for further information.

### 3.2.2 User interface

The GUI source code is available in the JNICODEFORHIDDLL folder. Hereunder is the JAVA user interface:

**Figure 3-3. JAVA User interface**

![JAVA User interface](image)

The components have the same roles as described for the VC++ interface (refer to section 3.1.4). The Auto-Connect box is used to allow the application to detect automatically the connection/disconnection of the device.

### 3.2.3 DOS demo

This demo gives a simple console application example. The demo uses a fixed VID/PID and has to be recompiled to modify these parameters. The device has to be connected and running with the Generic HID firmware before performing this console application.
4. The package architecture

When you unzip the DLL package, you’ll find several folders. Hereunder is the content of each one:

4.1 AtUsbHid

This folder contains the AtUsbHid.dll and the AtUsbHid.h files.

4.2 ExeDemo

This folder contains the different executable demo examples.

4.3 JNICodeForHIDDLL

This folder contains the source code of the JAVA project.

4.4 UsbHidDemoCode

This folder contains the source code of the VC++ project.

4.5 UsbHidSmallDemoCode

This folder contains the source code of the VC++ small demo (DOS demo).