BLUETOOTH DOCUMENT	Date / Year-Month-Day 2005-11-24	Approved Approved	Revision V10r00	Document No HID Lite_WP
Prepared HID WG	e-mail address hid-main@bluetooth.org			N.B. Confidential

HID Lite:

Simplifying the *Bluetooth*® Human Interface Device Profile to Add Bluetooth Keyboard and Mouse Input Capability to Resource-Limited Devices

White Paper

Abstract: This white paper describes the recommended approach to supporting keyboards and mice on devices with limited memory and processing power resources.

Special Interest Group (SIG)

The following Promoter Companies are representatives in the *Bluetooth* Special Interest Group:

Agere Systems Inc.
Ericsson Technology Licensing AB
IBM Corporation
Intel Corporation
Microsoft Corporation
Motorola Inc.
Nokia Corporation
Toshiba Corporation

Revision History		
Revision	**Date**	**Description**
D05r00	May 12, 2005	Initial Draft
D05r01	June 9, 2005	2nd Draft
D05r02	June 14, 2005	Edits from Peter Hauser, Microsoft
D05r03	July 14, 2004	Removed recommendation to use Class of Device to identify boot-mode capable devices. Added more detail on device discovery, connection process, use of SDP.
D05r04	July 27, 2005	- Added example SDP transactions - Removed HIDBootDevice information from main text and placed in footnote.
D05r05	August 22, 2005	- Updated link to Assigned Numbers document for Peripheral Minor Class of Device fields - Incorporated comments from Logitech and Microsoft
D05r06	September 22, 2005	- Submitted to BARB
D05r07	October 27, 2005	- Incorporated BARB feedback - Submit for 1.o approval.
D10r00	November 1, 2005	- Approved by BARB
V10r00	November 24, 2005	- BoD review period for objections expired

Contributors	
Name	**Company**
Robert Hulvey (document owner)	Broadcom
Ken Steck	CSR
Pierre Chênes	Logitech
Rene Sommer	Logitech
Chris Dreher	Microsoft
Peter Hauser	Microsoft
Kanji Kerai	Nokia

Disclaimer and Copyright Notice

Contents

1 Introduction

The number of small, portable devices such as personal digital assistants (PDAs) and cell-phones being shipped each year is growing at a staggering pace.[1] The decreasing size and increasing capabilities of these devices is creating a demand for Human Interface Devices (HIDs) which facilitate user interaction with these devices. As small devices discourage the use of connectors, wireless HIDs are particularly attractive. Given the widespread adoption of Bluetooth in PDAs, cell-phones, and other such small devices, Bluetooth is an excellent candidate for the wireless protocol to connect wireless HIDs to such devices.

This whitepaper explains how a subset of the Bluetooth HID profile can be employed to greatly simplify the implementation of the Bluetooth HID profile on resource-limited devices.

[1] From a recent (as of this writing) Forbes article in April 2005: “This year will see the sale of 660 million phones worldwide, compared with a mere 195 million PCs, according to Merrill Lynch and IDC, respectively.”

2 Background

The Bluetooth HID Profile borrows heavily from and incorporates much of the USB HID Class specification. As a result this foundation, the Bluetooth HID Profile enables a wide variety of human interface devices to be wirelessly connected to host devices in a way which allows the re-use of many software drivers already developed for USB HID devices.

Most HID devices on the market today, whether they employ USB or Bluetooth to connect to the host, were designed first and foremost for use with personal computers (PCs). However, small devices such as cell-phones and PDAs are not often designed to be USB hosts, and hence USB HID software drivers were seldom developed for these devices. Furthermore, many of these devices have limited processing and memory resources. Fortunately, the designers of the USB HID Class specification provided a means to scale down the HID protocol in order to facilitate implementation on such platforms, and the Bluetooth HID Profile specification defines how this can be implemented on Bluetooth host platforms.

The scaled down HID protocol was primarily created to enable BIOS (Basic Input Output Subsystem) firmware in PCs to support USB HID devices with a minimal amount of resources, and to thereby hasten the adoption of USB and USB HID devices in PCs. Most PCs can support at least a keyboard, even while the system is booting and before an operating system is loaded. The PC's BIOS firmware generally supports input from a keyboard as the PC boots up and until the operating system (OS) takes over.[2] The BIOS firmware is typically kept small, and hence only limited support for HID devices is provided. It is for this reason the scaled down HID protocol was called the "boot protocol".

The complexity of the full implementation of the HID protocol stems largely from the fact that it was designed not only to support those devices which were known when the specification was developed, but also to support future devices without having to change the protocol specification. This adaptability is in large part due to the use of "report descriptors" which a host system may solicit from a device after the device is detected. As the name suggests, the report descriptor enables the HID to describe to the host what types of information the HID is capable of reporting to the host. The host software drivers typically comprise a component known as a "parser" which reads and interprets the report descriptor to determine what types of reports to expect from a device. While this mechanism makes the HID protocol highly "future proof", it is also one of the most resource-intensive aspects of a full HID protocol implementation in terms of code size, memory, processing power, and in terms of human resources required for development

[2] Some older operating systems actually continue to use the BIOS firmware to interface to HID devices such as the keyboard and mouse, even after the OS has fully loaded.

and testing. It is primarily this parser component which the “boot protocol” originally defined by the USB HID specification aims to eliminate.

As the most widely used HIDs are by far the familiar keyboard and mouse, the boot protocol was designed to support only these 2 devices. The report formats for these devices are fixed in the boot protocol definition and defined in the USB HID spec, essentially allowing the report descriptors for these devices to be implicit, and allowing the complex parser software element to be eliminated.

The name “boot protocol” is also somewhat misleading. While “boot protocol” is descriptive of its usage in PC BIOSes, what may not be obvious to many readers of the USB HID specification is that the boot protocol is also suitable for implementation in resource-limited devices. Hence, one might consider “boot protocol” to be a “lite” version of the HID protocol, or “HID Lite”. In the Bluetooth HID Profile, further simplifications are allowed to facilitate the implementation of non-HID portions of the Bluetooth software stack. In this document, the term “HID Lite” will be used to describe the application of the boot protocol to simplify the implementation of the Bluetooth HID profile on resource-limited host devices.

3 Device Discovery

This section will describe the procedures recommended to be used by a HID Lite host for discovering Bluetooth devices which support HID Lite. The general steps required are:

1. Host performs an Inquiry Operation using GIAC and LIAC.
2. For each device found:
    a. If CoD indicates device is a HID device, then proceed to step (e), otherwise continue to step (b)
    b. Host establishes a baseband connection to each device
    c. Host establishes an L2CAP connection on SDP PSM
    d. Host SDP client requests HIDDeviceSubclass
    e. Host determines that the device supports HID Lite if either:bit 7 or bit 6 of HIDDeviceSubclass (or least significant byte of CoD) is set indicating keyboard and/or pointing device respectively.

The Service Discovery Profile (SDP) is generally used by one Bluetooth device to discover the services provided by another Bluetooth device. The SDP client functionality is generally required on the host device, while the SDP server functionality is typically found on the HID.

Appendix B contains further information regarding the usage of SDP.

NOTE: In environments with many devices in the proximity, this process could take a while (paging + connection establishment overhead).

3.1 Class of Device and SDP Attribute Recommendations

It is recommended that devices which function primarily as a keyboard, pointing device, or hybrid keyboard/pointing device, report a Class of Device (CoD) field using the Peripheral Major CoD value in the Frequency Hopping Synchronization (FHS) packet sent as a response during the Inquiry process. This will ensure that host stack software which presents a graphical user interface to the user will display an appropriate iconized representation of the device when it is discovered or when it is displayed in a list of paired devices.

For devices for which the primary function does not fall under the Bluetooth HID profile, but which may implement the Bluetooth HID profile as a sub-function, other CoD values may be used as appropriate. For example, a laptop should use the CoD of a laptop computer.

Regardless of the CoD field used for a device, all devices implementing the Bluetooth HID Profile are required to support the HIDDeviceSubclass SDP attribute. This attribute can be used to identify all currently defined boot-mode capable devices.[3] The HIDDeviceSubclass attributes is described in the Bluetooth HID Profile version 1.0 document as follows:

7.11.2 HIDDeviceSubclass

Attribute Name	Attribute ID	Attribute Value Type
HIDDeviceSubclass	0x0202	8-bit unsigned integer

Description

The HIDDeviceSubclass attribute is an 8-bit integer, which identifies the type of device (keyboard, mouse, joystick, gamepad, remote control, sensing device, etc.). Keyboards and mice are required to support boot mode operation. In boot mode, a device presents a fixed report, thus negating the requirement for a HID parser.

The Attribute value is identical to the low-order 8 bits of the Class of Device/Service (CoD) field in the FHS packet, where bits 7-2 contain the 6 bit Minor Device Class value (defined in Section 1.2 of the Bluetooth Assigned Numbers document [8]) and bits 1-0 are set to zero. See Section 7.2.1 for more information about boot devices.

HIDDeviceSubclass can differ from the Class of Device field of an FHS packet if the device is a composite device that implements multiple Bluetooth profiles.

The format and contents of the CoD field, which applies in part to the HIDDeviceSubclass SDP attribute, is defined in the Bluetooth Assigned Numbers document accessible at the following URL:

https://www.bluetooth.org/foundry/assignnumb/document/baseband

The permissible values for the Major Device Class field are given in the section at the following URL and copied below:

https://www.bluetooth.org/foundry/assignnumb/document/baseband#pgfId-133823

[3] The Bluetooth HID Profile version 1.0 also provides the HIDBootDevice attribute in order to accommodate any additional boot-mode reports which may be defined in future revisions of the Bluetooth HID Profile or USB HID specifications. Since boot-mode reports are defined only for keyboards and pointing devices as of this writing, HIDDeviceSubclass provides sufficient and more detailed information to identify boot-mode capable devices, and hence HIDBootDevice is not required. Please refer to the the Bluetooth HID Profile specification for more information on the HIDBootDevice attribute.

Bluetooth keyboards and mice should always use the “Peripheral” Major Device Class code shown here:

12	11	10	9	8	Major Device Class
0	0	1	0	1	Peripheral (mouse, joystick, keyboards, ...)

The permissible values for the Minor Device Class field for the Peripheral Major Device Class are given at the following URL and copied below:

https://www.bluetooth.org/foundry/assignnumb/document/baseband#pgfId-133895

1.2.8 Minor Device Class field - Peripheral Major Class

The Minor Device Class field top bits specify whether the peripheral device is a pointing device, keyboard, combination keyboard/pointing, or other.

7	6	Minor Device Class bit no of CoD
0	0	Not Keyboard / Not Pointing Device
0	1	Keyboard
1	0	Pointing device
1	1	Combo keyboard/pointing device

Table 1.9: The Peripheral Major Class keyboard/pointing device field

Bits 6 and 7 independently specify mouse, keyboard or combo mouse/keyboard devices. These may both be set to indicate a hybrid keyboard/pointing device, and may also be combined with the lower bits to identify a multifunctional device.

5	4	3	2	**Minor Device Class bit no of CoD**
0	0	0	0	Uncategorized device
0	0	0	1	Joystick
0	0	1	0	Gamepad
0	0	1	1	Remote control
0	1	0	0	Sensing device
0	1	0	1	Digitizer tablet
0	1	1	0	Card Reader (e.g. SIM Card Reader)
X	X	X	X	All other values reserved

Table 1.10: Reserved sub-field for the device type

3.2 Minimal SDP Implementation

This section describes how to implement the SDP client on the HID Lite host to minimize the code and memory resources required. The only SDP operation required is to request the HIDDeviceSubclass attribute value from the HID device. This operation comprises the following steps:

1. Establish the baseband connection.
2. Open an L2CAP channel using the SDP PSM
3. Complete the bi-directional channel configuration for the SDP channel.
4. Send an SDP_ServiceSearchAttributeRequest packet on the newly opened SDP channel.
5. Process the returned SDP_ServiceSearchAttributesResponse packet.
6. Close the SDP L2CAP channel

These steps will be described in detail in the following sections. It is assumed that the platform has a basic implementation of the L2CAP (Logical Link Control and Adaptation Protocol) protocol stack component and supports a minimal SDP record. L2CAP serves as a multiplexing layer to allow multiple services to share the same radio link between two devices. Both SDP and HID are services which use L2CAP. Note that

SDP is a special type of service, since it allows devices to solicit and to advertise other services and that all HID devices must also support SDP.

3.2.1 Establishing the Baseband connection

The baseband connection is typically established via the baseband page procedure as described in the Bluetooth Core Specification. The Bluetooth Device Address of the device for which SDP information is sought is typically discovered during a previous inquiry process and is given as a parameter to the baseband connection process. In some implementations, the baseband connection may be established automatically as part of opening the L2CAP channel as described in the next section.

NOTE: In some Bluetooth stack implementations, the baseband connection may be established automatically as a result of a request to open an L2CAP channel to a device.

3.2.2 Opening the L2CAP Channel

Opening the L2CAP channel is typically accomplished by calling an L2CAP API function to open a channel to an already-connected device. A Protocol Service Multiplexer (PSM) is generally required as a parameter to such a function, and the value 0x0001 is the PSM used to denote an SDP channel. As mentioned in the previous section, the request to open an L2CAP channel may implicitly result in the creation of a baseband connection.

Upon a successful connection, the “open” function will at a minimum return two 2-byte channel ID parameters which the SDP implementation must record for use in later steps. One of the channel ID parameters identifies the L2CAP connection to the local L2CAP implementation on the HID Lite host, while the other channel ID identifies the channel to the L2CAP implementation on the remote HID device.

3.2.3 Configuration of the L2CAP Channel

After the L2CAP channel is opened, both of the HID host and HID device must complete the channel configuration negotiations Please refer to the L2CAP specification contain in the Bluetooth Core specification for additional details (as of this writing, L2CAP is defined in Volume 3, Part A of the Bluetooth Core v2.0+EDR specification)

3.2.4 Sending the SDP_ServiceSearchAttributeRequest

Following the opening of the L2CAP channel on the SDP PSM, a request for the HIDDeviceSubclass attribute can be constructed and sent. The construction of this request is as follows:

```
06    PDU ID (SDP_ServiceSearchAttributeRequest = 0x06)
00 00 Transaction ID (may be any number)
00 0d ParameterLength (13 bytes to follow)
35    Indicates that a Sequence follows, and length is in the next byte
03    Sequence Length = 3
19    UUID, length 2 bytes
11 24 UUID for the HID device class
00 0f MaximumAttributeByteCount; response could be as long as 15 bytes
35    Indicates that a Sequence follows, and length is in the next byte
03    Length 3
09    Unsigned Integer, length 2 bytes
02 02 This is the AttributeID of HIDDeviceSubclass = 0x0202
00    ContinuationState (this is not a continuation of a previous request)
```

This packet would be passed to L2CAP for transmission and the local channel ID recorded from the L2CAP channel open operation would be used to identify the logical channel on which to transmit this packet.

3.2.5 Processing the SDP_ServiceSearchAttributeResponse

If successful, the response to the previously sent SDP_ServiceSearchAttributeRequest will contain some nested sequences which must be parsed to extract the value for the HIDDeviceSubclass attribute. The first 7 bytes of the response should appear as follows:

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (2 bytes, should match what was sent in the request)
00 XX ParameterLength (2 bytes, value may vary depending on coding)
00 NN AttributeListsByteCount (2 bytes, value may vary depending on coding)
```

The remainder of the response is variable depending on how the SDP server chooses to encode the data element sequences. There are 2 data element sequences in the response, one embedded in the other. The length of each of the sequences may optionally be encoded as a 1-, 2- or 4-byte value. Due to the flexibility in the sequence length encoding, there are 9 possible response variations to the SDP query described in the previous section. The response may be vary in size from 17 bytes to 23 bytes, and may be any size in between except for 22 bytes.

The 9 possible responses are listed below. Note that the Attribute value is shown as 0x40 in all of these responses and indicates a keyboard class of device. However, other values are possible and only the 2 most significant bits should be considered in determining whether or not the device supports the boot-mode protocol and hence must also support HID Lite.

Response 1 (Length = 17)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0c ParameterLength
00 09 AttributeListsByteCount
35    Sequence, length in the next byte
07    Length 7
35    Sequence, length in the next byte
05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 2 (Length = 18)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0d ParameterLength
00 0a AttributeListsByteCount
35    Sequence, length in the next byte
08    Length 8
36    Sequence, length in the next word
00 05 Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 3 (Length = 20)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0f ParameterLength
00 0c AttributeListsByteCount
35    Sequence, length in the next byte
0a    Length 10
37    Sequence, length in the next d-word
00 00 00 05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 4 (Length = 18)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0d ParameterLength
00 0a AttributeListsByteCount
```

```
36    Sequence, length in the next word
00 07    Length 7
35    Sequence, length in the next byte
05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 5 (Length = 19)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0e ParameterLength
00 0b AttributeListsByteCount
36    Sequence, length in the next word
00 08    Length 8
36    Sequence, length in the next word
00 05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 6 (Length = 21)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 10 ParameterLength
00 0d AttributeListsByteCount
36    Sequence, length in the next word
00 0a    Length 10
37    Sequence, length in the next d-word
00 00 00 05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40              ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 7 (Length = 20)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 0f ParameterLength
00 0c AttributeListsByteCount
37    Sequence, length in the next d-word
00 00 00 07    Length 7
35    Sequence, length in the next byte
05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
```

```
40    Attribute Value 0x40                ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 8 (Length = 21)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 10 ParameterLength
00 0d AttributeListsByteCount
37    Sequence, length in the next d-word
00 00 00 08    Length 8
36    Sequence, length in the next word
00 05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40                ← Desired HIDDeviceSubclass value
00    ContinuationState
```

Response 9 (Length = 23)

```
07    PDU ID (SDP_ServiceSearchAttributeResponse)
00 00 Transaction ID (should match what was sent in the request)
00 12 ParameterLength
00 0f AttributeListsByteCount
37    Sequence, length in the next d-word
00 00 00 0a    Length 10
37    Sequence, length in the next d-word
00 00 00 05    Length 5
09    Unsigned Integer, length 2 bytes
02 02 AttributeID HID_DEVICE_SUBCLASS
08    Unsigned Integer, length 1 bytes
40    Attribute Value 0x40                ← Desired HIDDeviceSubclass value
00    ContinuationState
```

3.2.6 Closing the L2CAP Channel

After the HIDDeviceSubclass attribute has been obtained, the L2CAP channel to the remote SDP server on the HID device may be closed. This is generally accomplished through a call to an API which accepts the local channel ID of the channel to be closed as a parameter.

NOTE: Some HID devices may not support cases where the SDP and HID channels are simultaneously open. Such devices should contain the SDP attribute HIDSDPDisable with the value of “true”. However, to simplify the HID Lite implementation, it is recommended that the HID Lite host close the SDP channel before opening either of the HID L2CAP channels for all devices in order to avoid the need to retrieve the HIDSDPDisable attribute.

4 Security

The use of security is mandatory for all Bluetooth keyboards. It is recommended that random PIN codes of at least 8 decimal digits be required to ensure reasonable security.

- Authentication mandatory if pairing is supported
- Encryption mandatory for keyboards (if set in features)

5 Connection Establishment

The connection establishment process between a HID host from a HID peripheral is as follows:

1. User (or host in some cases) selects HID with which to connect
2. Host establishes a baseband connection with the HID
    a. Host performs baseband paging
    b. Features are exchanged
    c. Authentication performed (mandatory for keyboards)
    d. Encryption enabled (mandatory for keyboards, optional for mice)
3. Host opens HID Control L2CAP PSM (0x0011)
4. Host opens HID Interrupt L2CAP PSM (0x0013)
5. Host sends SET_PROTOCOL to select boot-protocol
6. Host waits for keyboard or mouse reports to be sent from HID interrupt channel

To disconnect from a device, a HID Lite Host must first close the HID Interrupt channel and then close the HID Control channel.

6 Boot Protocol

As previously mentioned, since a HID using the boot protocol uses fixed reports, it is not necessary for a host which supports only the boot protocol to have a HID parser. Instead, the host software drivers may be "hard-coded" to handle only those reports defined by the boot protocol. For keyboards, the report format is 9 bytes in length. For a mouse, the report format is 4 bytes in length.

NOTE: Boot protocol reports are allowed to be extended up to 9 bytes, including the Report ID. The extended data may be equivalent to data sent in a valid report protocol mode report, or may be vendor-specific. Since boot protocol keyboard reports are already 9 bytes in length, they may not be extended. However, there is 1 byte within the keyboard report which is reserved for vendor-specific use. In addition, up to 5 bytes may be appended to mouse reports. HID hosts are required to ignore any extended data when operating in boot protocol mode.

Device	Report ID	Report Size
Reserved	0	N/A
Keyboard	1	9 Bytes
Mouse	2	4 Bytes
Reserved	3-255	N/A

Table 1: Bluetooth HID Boot Report IDs

The following sections describe the report formats supported in the boot protocol.

6.1 Keyboard Boot Protocol Input Report

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
0	Report ID = 0x01							
1	Right GUI	Right Alt	Right Shift	Right Control	Left GUI	Left Alt	Left Shift	Left Control
2	Reserved (0x00)							
3	Key 1							
4	Key 2							
5	Key 3							
6	Key 4							
7	Key 5							
8	Key 6							

Byte 0 of the boot protocol keyboard report contains the Report ID and is fixed at a value of 0x01 for a keyboard.

Byte 1 is sometimes called the modifier status byte. It reports the status of the Shift, Alt, Control, and GUI keys present on many keyboards. For each of the 4 modifier key types, there is a bit for the key if it appears on the left side of the keyboard and a bit for the key if it appears on the right side. If a modifier key is pressed, the corresponding bit in the modifier status byte is set to 1. If the corresponding key is not pressed, or does not exist on the keyboard, the respective bit is set to 0.

Byte 2 is reserved for vendor-specific use. If this byte is used for vendor-specific information, it is recommended that the host implementation check vendor or version information for the keyboard before acting upon the information in this field. How this may be done is beyond the scope of this document. Refer to the CompID in the LMP version information fields as described in the Bluetooth Core v2.0+EDR specification, or the Device ID v1.0 specification for information on identification information which a vendor might use to identify devices.

Bytes 3 through 8 compose the 6-byte key status array. These bytes are used to indicate which keys are currently being held down on the keyboard. Up to 6 non-modifier keys may be reported simultaneously. To report a key, the keyboard places codes into the key status array which correspond to specific keys pressed on the keyboard. For a keyboard in boot protocol mode, the host can simply use the reported codes as indices into a look-up table to determine the keys pressed.[4]

The codes allowed in the boot mode keyboard are defined in the Keyboard/Keypad Page section of the HID Usage Table (HUT) specification. The HUT specification document may be downloaded from the USB Implementers' Forum (USB-IF) website here:

http://www.usb.org

If more than 6 keys are held down at a time, the keyboard is required to report.the special usage code ErrorRollOver which has a value of 0x01. This code must be reported in all 6 array bytes. The keyboard may also report the 0x01 value in all 6 array bytes is a phantom or ghost condition is present.

If less than 6 keys are down, the key codes reported should appear first in the buffer and the remaining bytes filled with the code 0x00.

[4] Note that the report codes used for a keyboard operating in report protocol mode are interpreted somewhat differently than codes reported when the keyboard operates in boot protocol mode. In report protocol mode, each codes reported is an intermediate "scan code" which in turn is used as an index to look up what is called a usage code. However, the boot protocol mode was designed such that the scan codes are equal to the usage codes to simplify the decoding of the report by a boot mode host, which of course also facilitates the implementation of HID Lite host devices.

6.2 Keyboard Boot Protocol Output Report

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
0	Report ID = 0x01							
1	0	0	0	Kana	Compose	Scroll-Lock	Caps-Lock	Num-Lock

Keyboards supporting boot-protocol may optionally support the above output report to control LEDs on the keyboard. The output report must be sent either over the HID Control channel using the SET_REPORT command or over the HID Interrupt channel as a DATA payload (see the Bluetooth HID Profile specification for an explanation of a DATA payload). Since the SET_REPORT method entails a handshake packet, it may simplify the HID Lite host implementation if the report be sent over the Interrupt channel. In this way, the host need not wait for a response nor take any special action if the device does not support the LED output report.

6.3 Mouse Boot Protocol Input Report

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
0	Report ID = 0x02							
1	0	0	0	0	0	M-Btn	R-Btn	L-Btn
2	X-Axis displacement (8 bits)							
3	Y-Axis displacement (8 bits)							

Byte 0 contains the Report ID which is fixed at 0x02 for a mouse in boot protocol mode.

Byte 1 contains the button status bits:

- L-Btn = Left Mouse Button (1 = pressed, 0 = not pressed)
- M-Btn = Middle Mouse Button (1 = pressed, 0 = not pressed)
- R-Btn = Right Mouse Button (1 = pressed, 0 = not pressed)

Bytes 2 and 3 contain the X- and Y-Axis displacement reports:

- X = X-Axis Data (2's complement format with valid range from -127 to +127)
- Y = Y-Axis Data (2's complement format with valid range from -127 to +127)

Appendix A: Pending Errata Impacting this Whitepaper

The Bluetooth HID Profile Specification and PICS documents version 1.0 contain inconsistencies which make it difficult or impossible to qualify hosts which support only boot protocol mode, a.k.a. "boot mode only hosts." Erratum 747 corrects this issue and as of this writing has been approved by the HID Profile Working Group, but has not been formerly incorporated into the HID Profile Specification. Detailed information on the status of erratum 747 is available at the following links, which should be accessible to all Bluetooth SIG members:

http://errata.bluetooth.org/dp_view_all.cfm?errata_id=747

http://errata.bluetooth.org/draft_files/Errata%20Proposal%20Boot%20Mode%20Only%20Hosts_5_18_2005.doc

It is expected that additional errata will be created to remove the text suggesting the use of the Class of Device field for identifying boot-mode capable Bluetooth HID devices. See Appendix B for more information.

Future Bluetooth HID devices will require the use of both Limited and General Inquiry Access Codes. This will mean that HID hosts will have the option of searching for devices using either the LIAC or the GIAC. In environments with many discoverable devices, this will provide an advantage for HID devices as hosts will be able to reduce inquiry times by using the LIAC.

Appendix B: Use of Class of Device to Identify Boot-Mode Capable Devices

At the time of this writing, the Bluetooth HID Profile version 1.0 defines a means of identifying Bluetooth HID keyboards and mice using the Class of Device (CoD) information obtained during device discovery (also known as the "Inquiry" process). The text of the current specification suggests that, for a host which only implements the boot protocol, the SDP client functionality is then not needed, as the Major and Minor Device Class fields of the of the CoD field obtained during Inquiry indicates whether the device is a mouse or keyboard.

However, it is recommended that the Class of Device field be used only for the purposes of selecting an icon to display for a remote device during the device discovery process. Users may wish to use multi-function devices which implement the Bluetooth HID specification, but for which the primary function may be something other than a keyboard or pointing device.

For example, some PDAs on the market today feature small "thumb" keyboards. A user might wish to use such a PDA as a keyboard to input text on another device. The PDA is likely to report a CoD appropriate for a PDA (e.g. 0x000110 or 0x000114). Hence, a HID Lite implementation which relies solely on the CoD obtained in the Inquiry response to identify Bluetooth keyboards and pointing devices will not support using multi-function devices such as the example PDA.

The following are references from the Bluetooth HID Profile version 1.0 specification where the use of the CoD is recommended to identify boot devices. However, this usage of CoD is no longer recommended.

§5.5.1 BIOS Requirements for Boot Device Support:

> *The PC BIOS may use the Class of Device bits in the FHS packet to discover a mouse and keyboard as an alternative to reading the SDP record of the device.*

§7.2.1 Boot Mode Operation:

> *Boot mode was originally defined by USB HID to simplify the design of PC BIOSs; however, it has proved useful for a variety of products with small, embedded operating systems. When a HID device is in Boot mode, a HID parser and SDP client is not required in the host system.*

Appendix C: SPP Compliant Input Device Implementations

As of this writing, there are reports of keyboards and pointing devices on the market which use Bluetooth as the over-the-air protocol, but which do not conform to the existing Bluetooth HID specifications. These implementations have been motivated by the existence of legacy platforms such as cellular phones and PDAs which do not implement the Bluetooth HID profile. Many of these platforms have limited Bluetooth stacks which often:

- Support only a fixed number of profiles
- Are not readily upgradeable by the end-user
- Support the Bluetooth Serial Port Profile (SPP)
- Offer SPP APIs to user-loadable applications
- Do not expose L2CAP APIs to user-loadable applications

Many of these platforms also support keyboard input via a wired serial port connection, or via an infrared (IrDA) interface which is also a serial interface. Utilizing SPP may be the only way to provide keyboard and pointing device input capability for such platforms.

However, since there is no Bluetooth profile or other Bluetooth specification defining how keyboards or point devices are to be implemented over SPP, such devices may not interoperate with other Bluetooth devices as expected, and hence may create confusion for end-users in the marketplace.

Furthermore, the SPP does not require security to be used, and hence sensitive data typed on a keyboard which uses SPP to transmit keystroke information may be vulnerable to eavesdropping. The Bluetooth HID Profile mandates the use of security for all keyboards.

For these reasons, the practice of using SPP for these implementations is strongly discouraged. For all new host platform development, it is recommended that the guidelines in this HID Lite whitepaper be followed at a minimum.

However, for legacy host platforms for which HID Lite cannot be implemented, it is recommended that the use of security is enforced for keyboard devices. In addition, for

HID devices designed to work with such host platforms, it is strongly recommended that such devices support both the Bluetooth HID Profile specification in addition to the SPP based input protocol so as to enable the input device to interoperate with host stacks which do support the Bluetooth HID Profile.