11

Human Interface Devices: Using Control and Interrupt Transfers

The human interface device (HID) class was one of the first USB classes supported under Windows. On PCs running Windows 98 or later, applications can communicate with HIDs using the drivers built into the operating system. For this reason, many vendor-specific USB devices use the HID class.

Chapter 7 introduced the class. This chapter shows how to determine whether a specific device can fit into the human-interface class, details the firmware requirements that define a device as a HID and enable it to exchange data with its host, introduces the six HID-specific control requests, and presents an example of HID firmware. Chapter 12 describes the reports that HIDs use to exchange information and Chapter 13 shows how to access HIDs from applications.

What is a HID?

The *human interface* in the name suggests that HIDs interact directly with people, and many HIDs do. A mouse may detect when someone presses a key or moves the mouse, or the host may send a message that translates to a joystick effect that the user experiences. Besides keyboards, mice, and joysticks, the HID class encompasses front panels with knobs, switches, buttons, and sliders; remote controls; telephone keypads; and game controls such as data gloves and steering wheels.

But a HID doesn't have to have a human interface. The device just needs to be able to function within the limits of the HID class specification. These are the major abilities and limitations of HID-class devices:

- All data exchanged resides in structures called reports. The host sends and receives data by sending and requesting reports in control or interrupt transfers. The report format is flexible and can handle just about any type of data, but each defined report has a fixed size.
- A HID interface must have an interrupt IN endpoint for sending Input reports.
- A HID interface can have at most one interrupt IN endpoint and one interrupt OUT endpoint. If you need more interrupt endpoints, you can create a composite device that contains multiple HIDs. An application must obtain a separate handle for each HID in the composite device.
- The interrupt IN endpoint enables the HID to send information to the host at unpredictable times. For example, there's no way for the computer to know when a user will press a key on the keyboard, so the host's driver uses interrupt transactions to poll the device periodically to obtain new data.
- The rate of data exchange is limited, especially at low and full speeds. As Chapter 3 explained, a host can guarantee a low-speed interrupt endpoint no more than 800 bytes/sec. For full-speed endpoints, the maxi-

mum is 64 kilobytes/sec., and for high-speed endpoints, the maximum is about 24 Megabytes/sec. if the host supports high-bandwidth endpoints and about 8 Megabytes/sec. if not. Control transfers have no guaranteed bandwidth except for the bandwidth reserved for all control transfers on the bus.

- Windows 98 Gold (original edition) supports USB 1.0, so interrupt OUT transfers aren't supported and all host-to-device reports must use control transfers.

Any device that can live within the class's limits is a candidate to be a HID. The HID specification mentions bar-code readers, thermometers, and voltmeters as examples of HIDs that might not have a conventional human interface. Each of these sends data to the computer and may also receive requests that configure the device. Examples of devices that mostly receive data are remote displays, control panels for remote devices, robots, and devices of any kind that receive occasional or periodic commands from the host.

A HID interface may be just one of multiple USB interfaces supported by a device. For example, a USB speaker that uses isochronous transfers for audio may also have a HID interface for controlling volume, balance, treble, and bass. A HID interface is often cheaper than traditional physical controls on a device.

Hardware Requirements

To comply with the HID specification, the interface's endpoints and descriptors must meet several requirements.

Endpoints

All HID transfers use either the control endpoint or an interrupt endpoint. Every HID must have an interrupt IN endpoint for sending data to the host. An interrupt OUT endpoint is optional. Table 11-1 shows the transfer types and their typical uses in HIDs.

Table 11-1: The transfer type used in a HID transfer depends on the chip's abilities and the requirements of the data being sent.

Transfer Type	Source of Data	Typical Data	Required Pipe?	WIndows Support
Control	Device (IN transfer)	Data that doesn't have critical timing requirements.	yes	Windows 98 and later
	Host (OUT transfer)	Data that doesn't have critical timing requirements, or any data if there is no OUT interrupt pipe.		
Interrupt	Device (IN transfer)	Periodic or low-latency data.	yes	
	Host (OUT transfer)	Periodic or low-latency data.	no	Windows 98 SE and later

Reports

The requirement for an interrupt IN endpoint suggests that every HID must have at least one Input report defined in the HID's report descriptor. Output and Feature reports are optional.

Control Transfers

The HID specification defines six class-specific requests. Two requests, Set_Report and Get_Report, provide a way for the host and device to transfer reports to and from the device using control transfers. The host uses Set_Report to send reports and Get_Report to receive reports. The other four requests relate to configuring the device. The Set_Idle and Get_Idle requests set and read the Idle rate, which determines whether or not a device resends data that hasn't changed since the last poll. The Set_Protocol and Get_Protocol requests set and read a protocol value, which can enable a device to function with a simplified protocol when the full HID drivers aren't loaded on the host, such as during boot up.

Interrupt Transfers

Interrupt endpoints provide an alternate way of exchanging data, especially when the receiver must get the data quickly or periodically. Control transfers can be delayed if the bus is very busy, while the bandwidth for interrupt transfers is guaranteed to be available when the device is configured.

The ability to do Interrupt OUT transfers was added in version 1.1 of the USB specification, and the option to use an interrupt OUT pipe was added to version 1.1 of the HID specification. Windows 98 SE was the first Windows edition to support USB 1.1 and HID 1.1.

Firmware Requirements

The device's firmware must also meet class requirements. The device's descriptors must include an interface descriptor that specifies the HID class, a HID descriptor, and an interrupt IN endpoint descriptor. An interrupt OUT endpoint descriptor is optional. The firmware must also contain a report descriptor that contains information about the contents of a HID's reports.

A HID can support one or more reports. The report descriptor specifies the size and contents of the data in a device's reports and may also include information about how the receiver of the data should use the data. Values in the descriptor define each report as an Input, Output, or Feature report. The host receives data in Input reports and sends data in Output reports. A Feature report can travel in either direction.

Every device should support at least one Input report that the host can retrieve using interrupt transfers or control requests. Output reports are optional. To be compatible with Windows 98 Gold, devices that use Output reports should support sending the reports using control transfers. Using interrupt transfers for Output reports is optional. Feature reports always use control transfers and are optional.

Identifying a Device as a HID

As with any USB device, a HID's descriptors tell the host what it needs to know to communicate with the device. Listing 11-1 shows example device, configuration, interface, class, and endpoint descriptors for a vendor-specific HID. The host learns about the HID interface during enumeration by send-

```
{
// Device Descriptor

0x12,       // Descriptor size in bytes
0x01,       // Descriptor type (Device)
0x0200,     // USB Specification release number (BCD) (2.00)
0x00,       // Class Code
0x00,       // Subclass code
0x00,       // Protocol code
0x08,       // Endpoint 0 maximum packet size
0x0925,     // Vendor ID (Lakeview Research)
0x1234,     // Product ID
0x0100,     // Device release number (BCD)
0x01,       // Manufacturer string index
0x02,       // Product string index
0x00,       // Device serial number string index
0x01        // Number of configurations

// Configuration Descriptor

0x09,       // Descriptor size in bytes
0x02,       // Descriptor type (Configuration)
0x0029,     // Total length of this and subordinate descriptors
0x01,       // Number of interfaces in this configuration
0x01,       // Index of this configuration
0x00,       // Configuration string index
0xA0,       // Attributes (bus powered, remote wakeup supported)
0x50,       // Maximum power consumption (100 mA)

// Interface Descriptor

0x09,       // Descriptor size in bytes
0x04,       // Descriptor type (Interface)
0x00,       // Interface Number
0x00,       // Alternate Setting Number
0x02,       // Number of endpoints in this interface
0x03,       // Interface class (HID)
0x00,       // Interface subclass
0x00,       // Interface protocol
0x00,       // Interface string index
```

Listing 11-1: Descriptors for a vendor-specific HID (Sheet 1 of 2)

```
// HID Descriptor

0x09,        // Descriptor size in bytes
0x21,        // Descriptor type (HID)
0x0110,      // HID Spec. release number (BCD) (1.1)
0x00,        // Country code
0x01,        // Number of subordinate class descriptors
0x22,        // Descriptor type (report)
002F,        // Report descriptor size in bytes

// IN Interrupt Endpoint Descriptor

0x07,        // Descriptor size in bytes
0x05,        // Descriptor type (Endpoint)
0x81,        // Endpoint number and direction (1 IN)
0x03,        // Transfer type (interrupt)
0x40,        // Maximum packet size
0x0A,        // Polling interval (milliseconds)

// OUT Interrupt Endpoint Descriptor

0x07,        // Descriptor size in bytes
0x05,        // Descriptor type (Endpoint)
0x01,        // Endpoint number and direction (1 OUT)
0x03,        // Transfer type (interrupt)
0x40,        // Maximum packet size
0x0A         // Polling interval (milliseconds)
}
```

Listing 11-1: Descriptors for a vendor-specific HID (Sheet 2 of 2)

ing a Get_Descriptor request for the configuration containing the HID interface. The configuration's interface descriptor identifies the interface as HID-class. The HID class descriptor specifies the number of report descriptors supported by the interface. During enumeration, the HID driver requests the report descriptor and any physical descriptors.

The HID Interface

In the interface descriptor, bInterfaceclass = 3 to identify the interface as a HID. Other fields that contain HID-specific information in the interface descriptor are the subclass and protocol fields, which can specify a boot interface.

If bInterfaceSubclass = 1, the device supports a boot interface. A HID with a boot interface is usable when the host's HID drivers aren't loaded. This situation might occur when the computer boots directly to DOS, or when viewing the system setup screens that you can access on bootup, or when using Windows' Safe mode for system troubleshooting. A keyboard or mouse with a boot interface can use a simplified protocol supported by the BIOS of many hosts. The BIOS loads from ROM or other non-volatile memory on bootup and is available in any operating-system mode. The HID specification defines boot-interface protocols for keyboards and mice. If a device has a boot interface, the bInterfaceProtocol field indicates if the device supports the keyboard (1) or mouse (2) interface.

The HID Usage Tables document defines the report format for keyboards and mice that use the boot protocol. The BIOS knows what the boot protocol is and assumes that a boot device will support this protocol, so there's no need to read a report descriptor from the device. Before sending or requesting reports, the BIOS sends the HID-specific Set_Report request to request to use the boot protocol. When the full HID drivers have been loaded, the driver can use Set_Protocol to cause the device to switch from the boot protocol to the report protocol, which uses the report formats defined in the report descriptor

The bInterfaceSubclass field should equal zero if the HID doesn't support a boot protocol.

HID Class Descriptor

The HID class descriptor identifies additional descriptors for HID communications. The class descriptor has seven or more fields depending on the number of additional descriptors. Table 11-2 shows the fields. Note that the

Table 11-2: The HID class descriptor has 7 or more fields in 9 or more bytes.

Offset (decimal)	Field	Size (bytes)	Description
0	bLength	1	Descriptor size in bytes.
1	bDescriptorType	1	This descriptor's type: 21h to indicate the HID class.
2	bcdHID	2	HID specification release number (BCD).
4	bCountryCode	1	Numeric expression identifying the country for localized hardware (BCD).
5	bNumDescriptors	1	Number of subordinate report and physical descriptors.
6	bDescriptorType	1	The type of a class-specific descriptor that follows. (A report descriptor (required) is type 22h.)
7	wDescriptorLength	2	Total length of the descriptor identified above.
9	bDescriptorType	1	Optional. The type of a class-specific descriptor that follows. A physical descriptor is type 23h.
10	wDescriptorLength	2	Total length of the descriptor identified above. Present only if bDescriptorType is present immediately above. May be followed by additional wDescriptorType and wDescriptorLength fields to identify additional physical descriptors.

descriptor has two or more bDescriptorType fields. One identifies the HID descriptor and the other(s) identify the type of a subordinate descriptor.

The Descriptor

bLength. The length in bytes of the descriptor.

bDescriptorType. The value 21h indicates a HID descriptor.

The Class

bcdHID. The HID specification number that the interface complies with. In BCD format. Version 1.0 is 0100h; Version 1.1 is 0110h.

bCountryCode. If the hardware is localized for a specific country, this field is a code identifying the country. The HID specification lists the codes. If the hardware isn't localized, this field is 00h.

bNumDescriptors. The number of class descriptors that are subordinate to this descriptor.

bDescriptorType. The type of a descriptor that is subordinate to the HID class descriptor. Every HID must contain a report descriptor. One or more physical descriptors are optional.

wDescriptorLength. The length of the descriptor described in the previous field.

Additional bDescriptorType, wDescriptorLength (optional). If there are physical descriptors, the descriptor type and length for each follow in sequence.

Report Descriptors

A report descriptor defines the format and use of the data in the HID's reports. If the device is a mouse, the data reports mouse movements and button clicks. If the device is a relay controller, the data specifies which relays to open and close.

A report descriptor needs to be flexible enough to handle devices with different purposes. The data should use a concise format to keep from wasting storage space in the device or bus time when the data transmits. HID report descriptors achieve both goals by using a format that's more complex and less readable than a more verbose format might be.

A report descriptor is a class-specific descriptor. The host retrieves the descriptor by sending a Get_Descriptor request with the wValue field containing 22h in the high byte.

Listing 11-2 is a bare-bones report descriptor that describes an Input report, an Output report, and a Feature report. The device sends two bytes of data in the Input report. The host sends two bytes of data in the Output report. The Feature report is two bytes that the host can send to the device or request from the device.

Each item in the report descriptor consists of a byte that identifies the item and one or more bytes containing the item's data. The HID class specifica-

```
0x06  0xFFA0        Usage Page (vendor-defined)
0x09  0x01          Usage (vendor-defined)
0xA1  0x01          Collection (Application)

0x09  0x03          Usage (vendor-defined)
0x15  0x00          Logical Minimum (0)
0x26  0x00FF        Logical Maximum (255)
0x95  0x02          Report Count (2)
0x75  0x08          Report Size (8 bits)
0x81  0x02          Input (Data, Variable, Absolute)

0x09  0x04          Usage (vendor-defined)
0x15  0x00          Logical Minimum (0)
0x26  0x00FF        Logical Maximum (255)
0x75  0x08          Report Count (2)
0x95  0x02          Report Size (8 bits)
0x91  0x02          Output (Data, Variable, Absolute)

0x09  0x05          Usage (vendor-defined)
0x15  0x00          Logical Minimum (0)
0x26  0x00FF        Logical Maximum (255)
0x75  0x08          Report Count (2)
0x95  0x02          Report Size (8 bits)
0xB1  0x02          Feature (Data, Variable, Absolute)

0xC0                End Collection
```

Listing 11-2: This report descriptor defines an Input report, an Output report, and a Feature report. Each report transfers two vendor-defined bytes.

tion defines items that a report can contain. Here is what each item in the example descriptor specifies:

The **Usage Page** item is identified by the value 06h and specifies the general function of the device, such as generic desktop control, game control, or alphanumeric display. In the example descriptor, the Usage Page is the vendor-defined value FFA0h. The HID specification lists values for different Usage Pages and values reserved for vendor-defined Usage Pages.

The **Usage** item is identified by the value 09h and specifies the function of an individual report in a Usage Page. For example, Usages available for generic desktop controls include mouse, joystick, and keyboard. Because the example's Usage Page is vendor-defined, all of the Usages in the Usage Page are vendor-defined also. In the example, the Usage is 01h.

The **Collection (Application)** item begins a group of items that together perform a single function, such as keyboard or mouse. Each report descriptor must have an application collection.

The Collection contains three reports. Each report has these items:

A vendor-defined Usage applies to the data in the report.

A Logical Minimum and Logical Maximum specify the range of values that the report can contain.

The Report Count item indicates how many data items the report contains. In the example, each report contains two data items.

The Report Size item indicates how many bits are in each reported data item. In the example, each data item is eight bits.

The final item specifies whether the report is an Input report (81h), Output report (91h), or Feature report (B1h). The bits provided with the item contain additional information about the report data.

The End Collection item closes the Application Collection.

Chapter 12 has more about report formats.

HID-specific Requests

The HID specification defines six HID-specific requests. Table 11-3 lists the requests, and the following pages describe each request in more detail. All HIDs must support Get_Report, and boot devices must support Get_Protocal and Set_Protocol. The other requests (Set_Report, Get_Idle, and Set_Idle) are optional except that a keyboard using the boot protocol must support Set_Idle. If a HID doesn't have an Interrupt OUT endpoint or if the HID is communicating with a 1.0 host such as Windows 98 Gold, a HID that wants to receive reports from the host must support Set_Report.

Table 11-3: The HID class defines six HID-specific requests.

Request Number	Request	Data Source (Data stage)	wValue (high byte, low byte)	wIndex	Data Length (bytes) (wLength)	Data Stage Contents	Required?
01h	Get_ Report	device	report type, report ID	interface	report length	report	yes
02h	Get_ Idle	device	0, report ID	interface	1	idle duration	no
03h	Get_ Protocol	device	0	interface	1	protocol	required for HIDs that support a boot protocol
09h	Set_ Report	host	report type, report ID	interface	report length	report	no
0Ah	Set_ Idle	no Data stage	idle duration, report ID	interface	–	–	no, except for keyboards using the boot protocol
0Bh	Set_ Protocol	no Data stage	0, protocol	interface	–	–	required for HIDs that support a boot protocol

Get_Report

Purpose: The host requests an Input or Feature report from a HID using a control transfer.

Request Number (bRequest): 01h

Source of Data: device

Data Length (wLength): length of the report

Contents of wValue field: The high byte contains the report type (1=Input, 3=Feature), and the low byte contains the report ID. The default report ID is zero.

Contents of wIndex field: the number of the interface the request is directed to.

Contents of data packet in the Data stage: the report

Comments: All HIDs must support this request. See also Set_Report

Get_Idle

Purpose: The host reads the current Idle rate from a HID.

Request Number (bRequest): 02h

Source of Data: device

Data Length (wLength): 1

Contents of wValue field: The high byte is zero. The low byte indicates the report ID that the request applies to. If the low byte is zero, the request applies to all of the HID's Input reports.

Contents of wIndex field: the number of the interface that supports this request.

Contents of data packet in the Data stage: the Idle rate, expressed in units of 4 milliseconds.

Comments: See Set_Idle for more details. HIDs aren't required to support this request.

Get_Protocol

Purpose: The host learns whether the boot or report protocol is currently active in the HID.

Request Number (bRequest): 03h

Source of Data: device

Data Length (wLength): 1

Contents of wValue field: 0

Contents of wIndex field: the number of the interface that supports this request.

Contents of data packet in the Data stage: The protocol (0 = boot protocol, 1 = report protocol).

Comments: Boot devices must support this request. See also Set_Protocol.

Set_Report

Purpose: The host sends an Output or Feature report to a HID using a control transfer.

Request Number (bRequest): 09h

Source of Data: host

Data Length (wLength): length of the report

Contents of wValue field: The high byte contains the report type (2=Output, 3=Feature), and the low byte contains the report ID. The default report ID is zero.

Contents of wIndex field: the number of the interface the request is directed to.

Contents of data packet in the Data stage: the report

Comments: If a HID interface doesn't have an Interrupt OUT endpoint or if the host complies only with version 1.0 of the HID specification, this request is the only way the host can send data to the HID. HIDs aren't required to support this request. See also Get_Report.

Set_Idle

Purpose: Saves bandwidth by limiting the reporting frequency of an interrupt IN endpoint when the data hasn't changed since the last report.

Request Number (bRequest): 0Ah

Source of Data: no Data stage

Data Length (wLength): no Data stage

Contents of wValue field: The high byte sets the duration, or the maximum amount of time between reports. A value of zero means that the HID will send a report only when the report data has changed. The low byte indicates the report ID that the request applies to. If the low byte is zero, the request applies to all of the HID's Input reports.

Contents of wIndex field: the number of the interface that supports this request.

Contents of data packet in the Data stage: no Data stage.

Comments: The duration is in units of 4 milliseconds, which gives a range of 4 to 1,020 milliseconds. No matter what the duration value is, if the report data has changed since the last Input report sent, on receiving an interrupt IN token packet, the HID sends a report. If the data hasn't changed and the duration time hasn't elapsed since the last report, the HID returns NAK. If the data hasn't changed and the duration time has elapsed since the last report, the HID sends a report. A duration value of zero indicates an infinite duration: the HID sends a report only if the report data has changed and responds to all other interrupt IN requests with NAK.

If the HID returns a STALL in response to this request, the HID can send reports whether or not the data has changed. On enumerating a HID, the Windows HID driver attempts to set the idle rate to zero. The HID should Stall this request if an infinite Idle duration isn't wanted! HIDs aren't required to support this request except for keyboards using the boot protocol.

See also Get_Idle.

Set_Protocol

Purpose: The host specifies whether the HID should use the boot or report protocol.

Request Number (bRequest): 0Bh

Source of Data: no Data stage

Data Length (wLength): no Data stage

Contents of wValue field: the protocol (0 = boot protocol, 1 = report protocol).

Contents of wIndex field: the number of the interface that supports this request.

Contents of data packet in the Data stage: no Data stage

Comments: Boot devices must support this request. See also Get_Protocol

Transferring Data

When enumeration is complete, the host has identified the device interface as a HID and has established pipes with the interface's endpoints and learned what report formats to use to send and receive data.

The host can then request reports using either interrupt IN transfers and/or control transfers with Get_Report requests. The device also has the option to support receiving reports using interrupt OUT transfers and/or control transfers with Set_Report requests.

If you don't have example firmware for Get_Report and Set_Report, enumeration code can serve as a model. Get_Report is a control Read transfer that behaves in a similar way to Get_Descriptor except that the device returns a report instead of a descriptor. Set_Report is a control Write transfer. Unfortunately, the only standard USB request with a host-to-device Data stage is the rarely supported Set_Descriptor, so example code for control Write transfers is harder to find.

About the Example Code

The example code in this chapter is written for the Microchip PIC18F4550 introduced in Chapter 6. The code is based on Microchip's USB Firmware Framework. The Framework code is for the PIC18 family of microcontrollers but can give an idea of how to structure code for other CPUs. My code is adapted from Microchip's mouse code and implements a generic HID device that exchanges reports in both directions.

Portions of the code can be useful even if your device isn't a HID. The control-transfer examples can serve as models for responding to other class-specific and vendor-specific requests. From the firmware's point of view, bulk and interrupt transfers are identical, so the interrupt-transfer code can serve as a model for any firmware that uses bulk or interrupt transfers.

In the Framework code, a group of system files handles general USB tasks and class-specific tasks. These files typically require no changes or only minor changes and additions for specific applications. The system files include these:

usbmmap.c allocates memory for variables, endpoints, and other buffers used in USB communications.

usbdrv.c contains functions to detect device attachment and removal, check and respond to USB hardware interrupts, enter and exit the Suspend state, and respond to bus resets.

usbctrltrf.c contains functions for handling transactions in control transfers. The functions decode received Setup packets, manage the sending and receiving of data in the Data stage, and manage the sending and receiving of status information in the Status stage.

usb9.c contains functions that manage responding to the requests defined in Chapter 9 of the USB specification. The functions decode received requests, provide pointers to descriptors and other requested data to return, and take requested actions such as selecting a configuration or setting an address.

hid.c contains functions that manage tasks that are specific to the HID class. The functions decode received Setup data directed to the HID interface, define pointers to data to be sent and locations to store received data in control and interrupt transfers, and respond to other HID-specific requests such as Get_Idle and Set_Idle.

Additional files handle other tasks:

usbdesc.c contains structures that hold the device's descriptors, including the device descriptor, configuration and subordinate descriptors, string descriptors, and report descriptor. This information will of course differ for every device.

user_generic_hid.c contains most of the code that is specific to the generic HID application. The functions obtain the data that the device will send in reports and use the data received in reports.

main.c initializes the system and executes a loop that checks the bus status and calls functions in the *user_generic_hid* file to carry out the device's purpose.

The sections that follow contain excerpts from this code. The excerpts concentrate on the application-specific code used to send and receive reports.

The complete device firmware and host applications to communicate with the device are available from *www.Lvr.com*.

See Chapter 6 for an introduction to the '18F4550's architecture.

Sending Reports via Interrupt Transfers

When the HID driver is loaded on the host, the host controller begins sending periodic IN token packets to the HID's interrupt IN endpoint. The endpoint should NAK these packets until the HID has an Input report to send. To send data in an interrupt transfer, device firmware typically places the data in the endpoint's buffer and configures the endpoint to send the data to the host on receiving an IN token packet.

The code below executes after the host has configured the HID. The *usbmmap.c* file declares hid_report_in as a char array whose length equals the endpoint's wMaxPacketSize:

```
extern volatile far unsigned char
  hid_report_in[HID_INT_IN_EP_SIZE];
```

The code initializes the interrupt IN endpoint by setting values in the endpoint's buffer descriptor (HID_BD_IN):

```
// Set the endpoint's address register to the address
// of the hid_report_in buffer.

HID_BD_IN.ADR = (byte*)&hid_report_in;

// Set the status register bits:
// _UCPU = the CPU owns the buffer.
// _DAT1 = a DATA1 data toggle is expected next.
// (Before sending data, call the mUSBBufferReady
// macro to toggle the data toggle.)

HID_BD_IN.Stat._byte = _UCPU|_DAT1;
```

Listing 11-3 is a function that accepts a pointer to a buffer containing report data (*buffer) and the number of bytes to send (len) and makes the data available in the endpoint's buffer.

```
void HIDTxReport(char *buffer, byte len)
{
    byte i;

    // len can be no larger than the endpoint’s wMaxPacketSize
    // (HID_INT_IN_EP_SIZE). Trim len if necessary

    if(len > HID_INT_IN_EP_SIZE)
        len = HID_INT_IN_EP_SIZE;

    // Copy data from the passed buffer to the endpoint’s buffer.

    for (i = 0; i < len; i++)
        hid_report_in[i] = buffer[i];

    // To send len bytes, set the buffer descriptor’s count
    // register to len.

    HID_BD_IN.Cnt = len;

    // Toggle the data toggle and transfer ownership of the
    // buffer to the SIE, which will send the data on receiving
    // an interrupt IN token packet.

    mUSBBufferReady(HID_BD_IN);

}//end HIDTxReport
```

Listing 11-3: The HIDTxReport function provides data to send in an Input report at the HID’s interrupt IN endpoint and prepares the endpoint to send the report.

This code calls the function:

```
char transmit_buffer[2];

// Place report data in transmit_buffer:

transmit_buffer[0] = 104;
transmit_buffer[1] = 105;

// If necessary, wait until the CPU owns the interrupt
// IN endpoint's buffer.

while(mHIDTxIsBusy())
{
    // Service USB interrupts.

    USBDriverService();
}

// Make the report data available to send in the next
// interrupt IN transaction.

HIDTxReport(transmit_buffer, 2);
```

Before calling the function, the firmware places the report data in a char array (transmit_buffer) and waits if necessary for the mHIDTxIsBusy macro to return false, indicating that the CPU owns the HID's interrupt IN endpoint buffer. While waiting, the loop calls the USBDriverService function to service any USB interrupts that occur.

In the HIDTxReport function, if the len value passed to the function is greater than the endpoint's wMaxPacketSize value, len is trimmed to wMaxPacketSize. The function copies the report data from the passed buffer to the endpoint's buffer (hid_report_in).

The endpoint's byte-count register (HID_BD_IN.Cnt) holds the number of bytes to send. The macro mUSBBufferReady toggles the data-toggle bit and transfers ownership of the endpoint buffer to the SIE. On receiving an interrupt IN token packet, the SIE sends the packet and returns ownership of the buffer to the CPU.

Receiving Reports via Interrupt Transfers

If the HID interface has an interrupt OUT endpoint and the operating system supports USB 1.1 or later, the host can send Output reports to the device using interrupt transfers.

To receive data in an interrupt transfer, device firmware typically configures the endpoint to receive data from the host on receiving an OUT token packet, and an interrupt or polled register announces that data has been received.

The code below executes after the host has configured the HID. The *usbmmap.c* file declares hid_report_out as a char array whose length equals the endpoint's wMaxPacketSize:

```
extern volatile far unsigned char
  hid_report_out[HID_INT_OUT_EP_SIZE];
```

The code initializes the interrupt OUT endpoint by setting values in the endpoint's buffer descriptor (HID_BD_OUT):

```
// Set the endpoint's byte-count register to the
// length of the output report expected.

HID_BD_OUT.Cnt = sizeof(hid_report_out);

// Set the endpoint's address register to the address
// of the hid_report_out buffer.

HID_BD_OUT.ADR = (byte*)&hid_report_out;

// Set the status register bits:

// _USIE =  The SIE owns the buffer.
// _DATA0 = a DATA0 data toggle is expected next.
// _DTSEN = Enable data-toggle synchronization.

HID_BD_OUT.Stat._byte = _USIE|_DAT0|_DTSEN;
```

The endpoint is then ready to receive a report. Listing 11-4 is a function that retrieves data that has arrived at an interrupt OUT endpoint. The function accepts a pointer to a buffer to copy the report data to (*buffer) and the

```
byte HIDRxReport(char *buffer, byte len)
{
    // hid_rpt_rx_len is a byte variable declared in hid.c
    hid_rpt_rx_len = 0;

    if(!mHIDRxIsBusy())
    {
        // If necessary, trim len to equal
        // the actual number of bytes received.

        if(len > HID_BD_OUT.Cnt)
            len = HID_BD_OUT.Cnt;

        // The report data is in hid_report_out.
        // Copy the data to the user's buffer (buffer).

        for(hid_rpt_rx_len = 0;
            hid_rpt_rx_len < len;
            hid_rpt_rx_len++)

            buffer[hid_rpt_rx_len] =
                hid_report_out[hid_rpt_rx_len];

        // Prepare the endpoint buffer for next OUT transaction.

        // Set the endpoint’s count register to the length of
        // the report buffer.

        HID_BD_OUT.Cnt = sizeof(hid_report_out);

        // The mUSBBufferReady macro toggles the data toggle
        // and transfers ownership of the buffer to the SIE.

        mUSBBufferReady(HID_BD_OUT);

    }//end if

        return hid_rpt_rx_len;

}//end HIDRxReport
```

Listing 11-4: The HIDRxReport function retrieves data received in an Output report at the HID’s interrupt OUT endpoint.

number of bytes expected (len). The function returns the number of bytes copied or zero if no data is available.

This code calls the function:

```
byte number_of_bytes_read;
char receive_buffer[2];

number_of_bytes_read =
    HIDRxReport(receive_buffer, 2);
```

In the HIDRxReport function, the mHIDRxIsBusy macro checks to see if the CPU has ownership of the buffer. If not, the function returns zero. Otherwise, if necessary, the function trims the expected number of bytes (len) to match the number of bytes received as reported in the endpoint's byte-count register (HID_BD_OUT.Cnt). The received report is in the buffer hid_report_out. The function copies the received bytes from hid_report_out to the buffer that was passed to the function (receive_buffer). To prepare for a new transaction, the function sets the endpoint's byte-count register to the size of the hid_report_out buffer and calls the mUSBBufferReady macro, which toggles the data toggle and transfers ownership of the buffer to the SIE. The endpoint can then receive new data. When the function returns, the report data is available in receive_buffer. The hid_rpt_rx_len variable contains the number of bytes received (zero if no bytes were received).

Sending Reports via Control Transfers

To send an Input or Feature report using a control transfer, device firmware must detect the Get_Report code in the bRequest field of the Setup stage of a control request directed to the HID interface. The firmware then configures Endpoint 0 to send the report in the Data stage and receive the host's response in the Status stage.

Microchip's Framework HID example includes code that detects the request and calls the HIDGetReportHandler function in Listing 11-5.

The function examines the high byte of the wValue field of the Setup stage's data packet (MSB(SetupPkt.W_Value)) to determine if the host is request-

```
void HIDGetReportHandler(void)
{
    // The report type is in the high byte of the setup packet's
    // wValue field. 1 = Input; 3 = Feature.

    switch(MSB(SetupPkt.W_Value))
    {
        byte count;

        case 1: // Input report

            // Find out which report ID was specified.
            // The report ID is in the low byte (LSB) of the
            // wValue field. This example supports Report ID 0.
            switch(LSB(SetupPkt.W_Value))
            {
                case 0: // Report ID 0

                // The HID class code will handle the request.
                ctrl_trf_session_owner = MUID_HID;

                // Provide the data to send.
                GetInputReport0();

                break;

                case 1: // Report ID 1
                 // Add code to handle Report ID 1 here.
                break;

            } // end switch(LSB(SetupPkt.W_Value))

            break;

        case 3: // Feature report

            // Add code to handle Feature reports here.

    } // end switch(MSB(SetupPkt.W_Value))
}//end HIDGetReportHandler
```

Listing 11-5: The HIDGetReportHandler function is called on receiving a Get_Report request.

ing an Input or Feature report. The low byte of the wValue field (LSB(SetupPkt.W_Value)) is the report ID that names the specific report requested.

If the code supports the requested report, the ctrl_trf_session_owner variable is set to MUID_HID. The firmware handling other stages of the transfer can use this variable to detect who is handling the request.

To send Input report 0, the function calls the GetInputReport0 function (Listing 11-6). This function sets pSrc.bRam, a pointer to data in RAM, to the location of the report's data (hid_report_in).

The Framework firmware prepares Endpoint 0 to send the data that pSrc.bRam points to on receiving an IN token packet and accepts the host's zero-length packet in the Status stage of the transfer.

The code for sending other reports, including Input reports with other report IDs and Feature reports, is much the same except that the source of the report's contents will change depending on the report.

Receiving Reports via Control Transfers

To receive an Output or Feature report in a control transfer, the firmware must detect the Set_Report request in the bRequest field of a request directed to the HID interface. The device must also receive the report data in the Data stage and return a handshake in the Status stage. Microchip's Framework HID example includes code that detects the request and calls the HIDSetReportHandler function in Listing 11-7.

The function examines the high byte of the wValue field of the Setup stage's data packet (MSB(SetupPkt.W_Value)) to determine if the host is sending an Output or Feature report. The low byte of the wValue field (LSB(SetupPkt.W_Value)) is the report ID that names the specific report being sent.

If the code supports the requested report, the ctrl_trf_session_owner variable is set to MUID_HID so the firmware that handles other stages of the transfer can detect who is handling the request.

The Framework firmware prepares Endpoint 0 to receive the report data on receiving an OUT token packet.

```
void GetInputReport0(void)
{
    byte count;

    // Set pSrc.bRam to point to the report.

    pSrc.bRam = (byte*)&hid_report_in;
}
```

Listing 11-6: The GetInputReport0 function handles report-specific tasks.

When Output report 0 arrives in the Data stage of the transfer, the Framework firmware makes the data available at the RAM location pointed to by pDst.bRam and manages the sending of a zero-length packet in the transfer's Status stage. In this example, pDst.bRam has been set to point to the char array hid_report_out. The firmware can call a function that uses the report data:

```
if (ctrl_trf_session_owner == MUID_HID)
{
  // Call a function that uses the report data.

  HandleControlOutReport();
}
```

The code for receiving other reports, including Output reports with other report IDs and Feature reports, is much the same except that the firmware's use of the report's contents will change depending on the report.

```
void HIDSetReportHandler(void)
{
    // The report type is in the high byte of the Setup packet's
    // wValue field. 2 = Output; 3 = Feature.

    switch(MSB(SetupPkt.W_Value))
    {
        case 2: // Output report

            // The report ID is in the low byte of the Setup
            // packet's wValue field.
            // This example supports Report ID 0.

            switch(LSB(SetupPkt.W_Value))
            {
                case 0: // Report ID 0

                  // The HID-class code will handle the request.
                   ctrl_trf_session_owner = MUID_HID;

                   // When the report arrives in the Data stage,
                   // the report data will be available in
                   // hid_report_out.
                   pDst.bRam = (byte*)&hid_report_out;
                   break;

                case 1: // Report ID 1
              // Place code to handle Report ID 1 here.

            } // end switch(LSB(SetupPkt.W_Value))

            break;

        case 3: // Feature report

            // Place code to handle Feature reports here.

    } // end switch(MSB(SetupPkt.W_Value))

} //end HIDSetReportHandler
```

Listing 11-7: The HIDSetReportHandler function prepares to receive an Output or Feature report from the host.

12

Human Interface Devices: Reports

Chapter 11 introduced the reports that HIDs use to exchange data. A report can be a basic buffer of bytes or a complex assortment of items, each with assigned functions and units. This chapter shows how to design a report to fit a specific application.

Report Structure

A report descriptor may contain any of dozens of items arranged in various combinations. The advantage of a more complex descriptor is that the device can provide detailed information about the data it sends and expects to receive. The descriptor can specify the values' uses and what units to apply to the raw data. From a report descriptor, an application can learn whether a device supports a particular feature, such as force feedback on a joystick.

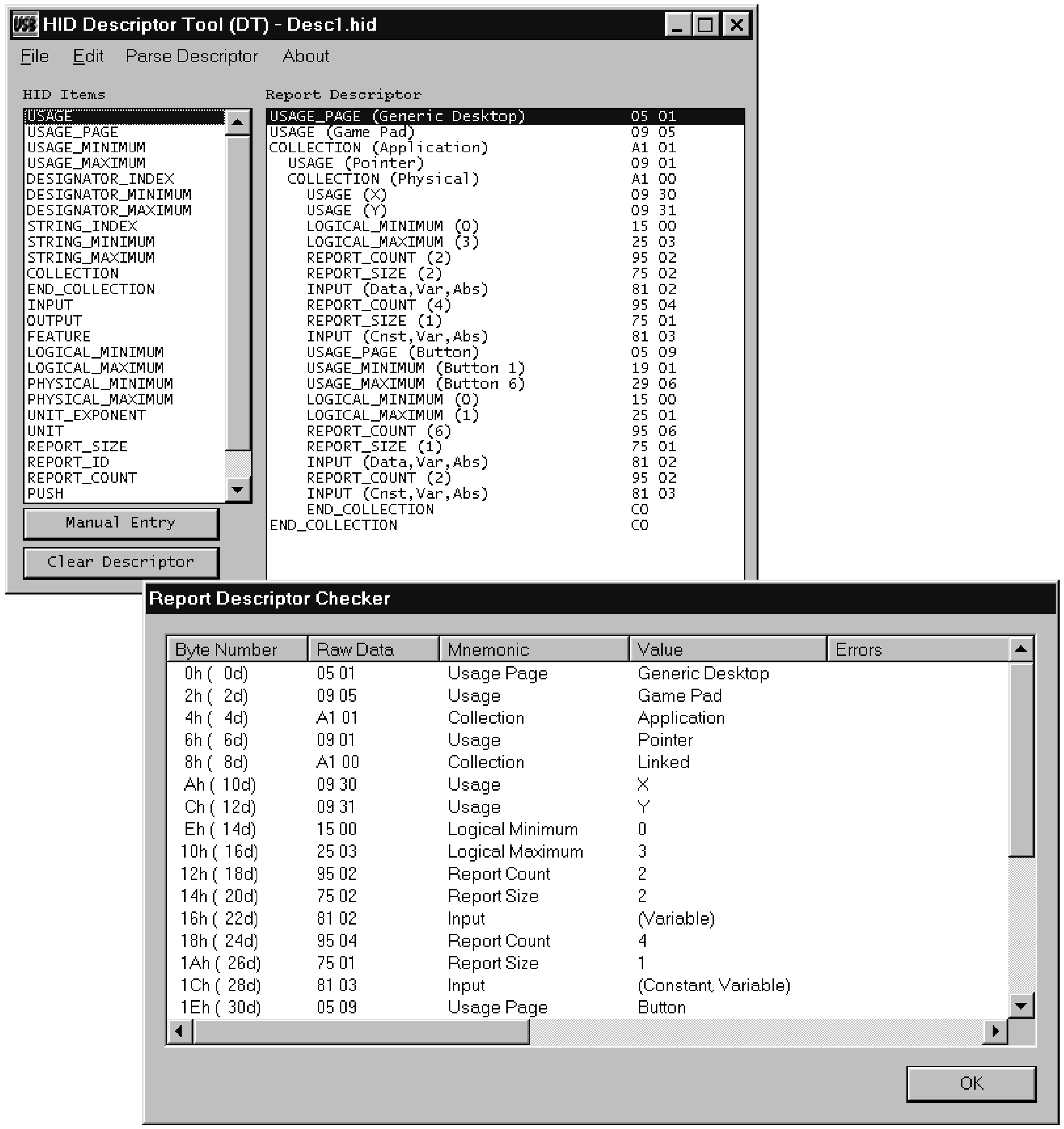
But just because the specification defines an item that could apply to a device's data doesn't mean that the report descriptor has to use that item. For vendor-specific devices that are intended for use with a single application, the application often knows in advance the type, size, and order of the data in a report, so there's no need to obtain this information from the device. For example, when the vendor of a data-acquisition unit creates an application for use with the unit, the vendor already knows the data format the device uses in its reports. At most, the application might check the product ID and release number from the device descriptor to learn whether the application can request a particular setting or action.

Some of the details about report structures can get tedious, and it's rarely necessary to understand every nuance about every item type. So feel free to skim through the details in this chapter and come back to them later if you need to.

A report descriptor contains one or more controls and data items that describe values to be transferred in one or more reports. A control is a button, switch, or other physical entity that operates or regulates an aspect of a device. Data items describe any other data the report transfers. Each control or data item has a defined scope. Some items can apply to multiple values, eliminating the need for repetition.

Using the HID Descriptor Tool

The HID Descriptor Tool (Figure 12-1) is a free utility available from the USB-IF. The tool helps in creating report descriptors, and will also check your descriptor's structure and report errors. Instead of having to look up the values that correspond to each item in your report, you can select the item from a list and enter the value you want to assign to it, and the tool adds the item to the descriptor. You can also add items manually. The Parse Descriptor function displays the raw and interpreted values in your descriptor and comments on any errors found. When you have a descriptor with no errors, you can convert it to the syntax required by your firmware. The tool has limited support for vendor-specific items, however, and may flag these as errors.


HID Descriptor Tool (DT) - Desc1.hid
File
Edit
Parse Descriptor
About
Manual Entry
Clear Descriptor
Report Descriptor Checker
OK

HID Items

```
USAGE
USAGE_PAGE
USAGE_MINIMUM
USAGE_MAXIMUM
DESIGNATOR_INDEX
DESIGNATOR_MINIMUM
DESIGNATOR_MAXIMUM
STRING_INDEX
STRING_MINIMUM
STRING_MAXIMUM
COLLECTION
END_COLLECTION
INPUT
OUTPUT
FEATURE
LOGICAL_MINIMUM
LOGICAL_MAXIMUM
PHYSICAL_MINIMUM
PHYSICAL_MAXIMUM
UNIT_EXPONENT
UNIT
REPORT_SIZE
REPORT_ID
REPORT_COUNT
PUSH
```

Report Descriptor

```
USAGE_PAGE (Generic Desktop)          05 01
USAGE (Game Pad)                      09 05
COLLECTION (Application)              A1 01
  USAGE (Pointer)                     09 01
  COLLECTION (Physical)               A1 00
    USAGE (X)                         09 30
    USAGE (Y)                         09 31
    LOGICAL_MINIMUM (0)               15 00
    LOGICAL_MAXIMUM (3)               25 03
    REPORT_COUNT (2)                  95 02
    REPORT_SIZE (2)                   75 02
    INPUT (Data,Var,Abs)              81 02
    REPORT_COUNT (4)                  95 04
    REPORT_SIZE (1)                   75 01
    INPUT (Cnst,Var,Abs)              81 03
    USAGE_PAGE (Button)               05 09
    USAGE_MINIMUM (Button 1)          19 01
    USAGE_MAXIMUM (Button 6)          29 06
    LOGICAL_MINIMUM (0)               15 00
    LOGICAL_MAXIMUM (1)               25 01
    REPORT_COUNT (6)                  95 06
    REPORT_SIZE (1)                   75 01
    INPUT (Data,Var,Abs)              81 02
    REPORT_COUNT (2)                  95 02
    INPUT (Cnst,Var,Abs)              81 03
    END_COLLECTION                    C0
END_COLLECTION                        C0
```

Byte Number	Raw Data	Mnemonic	Value	Errors
0h (0d)	05 01	Usage Page	Generic Desktop	
2h (2d)	09 05	Usage	Game Pad	
4h (4d)	A1 01	Collection	Application	
6h (6d)	09 01	Usage	Pointer	
8h (8d)	A1 00	Collection	Linked	
Ah (10d)	09 30	Usage	X	
Ch (12d)	09 31	Usage	Y	
Eh (14d)	15 00	Logical Minimum	0	
10h (16d)	25 03	Logical Maximum	3	
12h (18d)	95 02	Report Count	2	
14h (20d)	75 02	Report Size	2	
16h (22d)	81 02	Input	(Variable)	
18h (24d)	95 04	Report Count	4	
1Ah (26d)	75 01	Report Size	1	
1Ch (28d)	81 03	Input	(Constant, Variable)	
1Eh (30d)	05 09	Usage Page	Button	

Figure 12-1: The HID Descriptor Tool helps in creating and testing HID report descriptors.

Control and Data Item Values

Several documents define values that reports may contain. The first place to look is the *HID Usage Tables* document, which defines values for generic desktop controls, simulation controls, game controls, LEDs, buttons, telephone devices, and more. The document also tells you where to find values defined elsewhere. Some are in the HID specification, while others are in the class specifications for specific device functions such as monitor, power, and point-of-sale devices.

Item Types

The HID specification defines two report item types: short items and long items. As of HID 1.1, there are no defined Long items, and the type is just reserved for future use.

Short Items

A Short item's 1-byte prefix specifies the item type, item tag, and item size. These are the elements that make up the prefix byte:

Bit Number	Contents	Description
0	Item Size	Number of bytes in the item
1		
2	Item Type	Item scope: Main, Global, or Local
3		
4	Item Tag	Numeric value that indicates the item's function
5		
6		
7		

The item size (bits 1 and 0) indicates how many data bytes the item contains. Note that an item size of 3 (11 in binary) corresponds to 4 data bytes:

Item Size (binary)	**Number of Data Bytes**
00	0
01	1
10	2
11	4

The item type (bits 3 and 2) describes the scope of the item: Main (00), Global (01), or Local (10). Main items define or group data fields in the descriptor. Global items describe data. Local items define characteristics of individual controls or data. (This chapter has more information about these item types.)

The item tag (bits 4-7) indicates the item's function.

Long Items

A Long item uses multiple bytes to store the same information as the Short item's 1-byte prefix stores. A Long item's 1-byte prefix (FEh) identifies the item as a Long item. In addition, the item has a byte that specifies the number of data bytes, a byte containing the item tag, and up to 255 bytes of data.

The Main Item Type

A Main item defines or groups data items within a report descriptor. There are five Main item types. Input, Output, and Feature items each define fields in a type of report. Collection and End Collection items group related items within a report. The default value for all Main items is zero.

Input, Output, and Feature Items

Table 12-1 shows supported values for the Input, Output, and Feature items, including the item prefix and the meanings of the bits in the data that follows the prefix.

An Input item applies to data a device sends to the host. An Input report contains one or more Input items. The host uses interrupt IN transfers or Get_Report requests to request Input reports.

An Output item applies to information that the host sends to the device. An Output report contains one or more Output items. Output reports contain data that reports the states of controls, such as whether to open or close a switch or the intensity to apply to an effect. As explained earlier, if the HID has an interrupt OUT endpoint, a HID 1.1-compliant host can use interrupt OUT transfers to send Output reports. All hosts can also use Set_Report requests to send Output reports.

A Feature item typically applies to information that the host sends to the device. However, it's also possible for the host to read Feature items from a device. A Feature report contains one or more Feature items. Feature reports often contain configuration settings that affect the overall behavior of the device or one of its components. The reports often control settings that you might otherwise adjust in a physical control panel. For example, the host may have a virtual (on-screen) control panel to enable users to select and control a device's capabilities and settings. The host uses control transfers with Set_Report and Get_Report requests to send and receive Feature reports.

Following each Input, Output, or Feature item prefix are up to 9 bits that describe the item's data. (An additional 23 bits are reserved.) For example, an Input item prefix followed by 9 bits of data has the value 82h. The prefix's high four bits equal 08h to indicate an Input item, and the low four bits equal 02h to indicate that the data's 9 bits require 2 bytes. An Input item prefix followed by 8 bits of data has the value 81h. The prefix's high four bits equal 08h to indicate an Input item, and the low four bits equal 01h to because the data's 8 bits fit in 1 byte. The device firmware and host software may use or ignore the information in these items.

Table 12-1: The bits that follow Input, Output, and Feature Item prefixes describe the data in report items.

Main Item Prefix	Bit Number	Meaning if bit = 0	Meaning if bit = 1
Input (100000nn, where nn=the number of bytes in the data following the prefix) Use 81h for 1 byte of item data or use 82h for 2 bytes of item data.	0	Data	Constant
	1	Array	Variable
	2	Absolute	Relative
	3	No wrap	Wrap
	4	Linear	Non-linear
	5	Preferred state	No preferred state
	6	No null position	Null state
	7	Reserved	
	8	Bit field	Buffered bytes
	9-31	Reserved	
Output (100100nn, where nn=the number of bytes in the data following the prefix) Use 91h for 1 byte of item data or use 92h for 2 bytes of item data.	0	Data	Constant
	1	Array	Variable
	2	Absolute	Relative
	3	No wrap	Wrap
	4	Linear	Non-linear
	5	Preferred state	No preferred state
	6	No null position	Null state
	7	Non-volatile	Volatile
	8	Bit field	Buffered bytes
	9-31	Reserved	
Feature (101100nn, where nn=the number of bytes in the data following the prefix) Use B1h for 1 byte of item data or use B2h for 2 bytes of item data.	0	Data	Constant
	1	Array	Variable
	2	Absolute	Relative
	3	No wrap	Wrap
	4	Linear	Non-linear
	5	Preferred state	No preferred state
	6	No null position	Null state
	7	Non-volatile	Volatile
	8	Bit field	Buffered bytes
	9-31	Reserved	

The bit functions are the same for Input, Output, and Feature items, except that Input items don't support the volatile/non-volatile bit. These are the uses for each bit:

Data | Constant. Data means that the contents of the item are modifiable (read/write). Constant means the contents are not modifiable (read-only).

Array | Variable. This bit specifies whether the data reports the state of every control (Variable) or just reports the states of controls that are asserted, or active (Array). Reporting only the asserted controls results in a more compact report for devices such as keyboards, where there are many controls (keys) but only one or a few are asserted at the same time.

For example, if a keypad has eight keys, setting this bit to Variable would mean that the keypad's report would contain a bit for each key. In the report descriptor, the report size would be one bit, the report count would be eight, and the total amount of data sent would be eight bits. Setting the bit to Array would mean that each key has an assigned index, and the keypad's report would contain only the indexes of keys that are pressed. With eight keys, the report size would be three bits, which can report a key number from 0 through 7. The report count would equal the maximum number of simultaneous keypresses that could be reported. If the user can press only one key at a time, the report count would be 1 and the total amount of data sent would be just 3 bits. If the user can press all of the keys at once, the report count would be 8 and the total amount of data sent would be 24 bits.

An out-of-range value reported for an Array item indicates that no controls are asserted.

Absolute | Relative. Absolute means that the value is based on a fixed origin. Relative means that the data indicates the change from the last reading. A joystick normally reports absolute data (the joystick's current position), while a mouse reports relative data (how far the mouse has moved since the last report).

No Wrap | Wrap. Wrap indicates that the value rolls over to the minimum if the value continues to increment after reaching its maximum and that the value rolls over to the maximum if the value continues to decrement after reaching its minimum. An item specified as No Wrap that exceeds the specified limits may report a value outside the limits. This bit doesn't apply to Array data.

Linear | Non-linear. Linear indicates that the measured data and the reported value have a linear relationship. In other words, a graph of the reported data and the property being measured forms a straight line. In non-linear data, a graph of the reported data and the property being measured forms a curve. This bit doesn't apply to Array data.

Preferred State | No Preferred State. Preferred state indicates that the control will return to a particular state when the user isn't interacting with it. A momentary pushbutton has a preferred state (out) when no one is pressing the button. A toggle switch has no preferred state and remains in the last state selected by a user. This bit doesn't apply to Array data.

No Null Position | Null State. Null state indicates that the control supports a state where the control isn't sending meaningful data. A control indicates that it's in the null state by sending a value outside the range defined by its Logical Minimum and Logical Maximum. No Null Position indicates that any data sent by the control is meaningful data. A hat switch on a joystick is in a null position when it isn't being pressed. This bit doesn't apply to Array data.

Non-volatile | Volatile. The Volatile bit applies only to Output and Feature report data. Volatile means that the device can change the value on its own, without host interaction, as well as when the host sends a report requesting the device to change the value. For example, a control panel may have a control that users can set in two ways. A user may use a mouse to click a button on the screen to cause the host to send a report to the device, or a user may press a physical button on the device. Non-volatile means that the device changes the value only when the host requests a new value in a report.

When the host is sending a report and doesn't want to change a volatile item, the value to assign to the item depends on whether the data is defined as relative or absolute. If a volatile item is defined as relative, a report that assigns a value of 0 should result in no change. If a volatile item is defined as absolute, a report that assigns an out-of-range value should result in no change.

This bit doesn't apply to Array data.

Bit Field | Buffered Bytes. Bit Field means that each bit or a group of bits in a byte can represent a separate piece of data and the byte doesn't represent a single quantity. The application interprets the contents of the field. Buffered Bytes means that the data consists of one or more bytes. The report size for Buffered Byte items must be eight. This bit doesn't apply to Array data. Note that this bit is bit 8 in the item's data so using this bit requires two data bytes in the item.

Collection and End Collection Items

All of the report types can use Collection and End Collection items to group related items.

The three defined types of collections are application, physical, and logical. Vendors can also define collection types. Collections can be nested. Table 12-2 shows the values of the Collection and End Collection tags and the defined values for the different collection types.

An application collection contains items that have a common purpose or that together carry out a single function. For example, the boot descriptor for a keyboard groups the keypress and LED data in an application collection. All report items must be in an application collection.

A physical collection contains items that represent data at a single geometric point. A device that collects a variety of sensor readings from multiple locations might group the data for each location in a physical collection. The boot descriptor for a mouse groups the button and position indicators in a physical collection.

A logical collection forms a data structure consisting of items of different types that are linked by the collection. An example is the contents of a data buffer and a count of the number of bytes in the buffer.

Each collection begins with a Collection item and ends with an End Collection item. All Main items between the Collection and End Collection items are part of the collection. Each collection must have a Usage tag (described below).

Table 12-2: Data values for the Collection and End Collection Main Item Tags.

Main Item Type	Value	Description
Collection (A1h)	00h	Physical
	01h	Application
	02h	Logical
	03h-7Fh	Reserved
	80h-FFh	Vendor-defined
End Collection (C0h)	None	Closes a collection

A top-level collection is a collection that isn't nested within another collection. A HID interface can have more than one top-level collection, with each top-level collection representing a different HID. For example, a keyboard with an embedded pointing device can have a HID interface with two top-level collections, one for the pointing-device's reports and one for the keyboard's reports. Unlike HIDs in separate interfaces, these HIDs share interrupt endpoints.

If a report contains an unknown vendor-defined collection type, the host should ignore all Main items in the collection. If a known collection type has an unknown Usage, the host should ignore all items in the collection.

The Global Item Type

Global items identify reports and describe the data in them, including characteristics such as the data's function, maximum and minimum allowed values, and the size and number of report items. A Global item tag applies to every item that follows until the next Global tag. To save storage space in the device, the report descriptor doesn't have to repeat values that don't change from one item to the next. There are 12 defined Global items, shown in Table 12-3. The following sections describe the items in more detail.

Identifying the Report

Report ID. This value can identify a specific report. A HID can support multiple reports of the same type, with each report having its own report

Table 12-3: There are twelve defined Global items.

Global Item Type	Value (nn indicates the number of bytes that follow)	Description
Usage Page	000001nn	Specifies the data's usage or function.
Logical Minimum	000101nn	Smallest value that an item will report.
Logical Maximum	001001nn	Largest value that an item will report.
Physical Minimum	001101nn	The logical minimum expressed in physical units.
Physical Maximum	010001nn	The logical maximum expressed in physical units.
Unit exponent	010101nn	Base 10 exponent of units.
Unit	011001nn	Unit values.
Report Size	011101nn	Size of an item's fields in bits.
Report ID	100001nn	Prefix that identifies a report.
Report Count	100101nn	The number of data fields for an item.
Push	101001nn	Places a copy of the global item state table on the stack.
Pop	101101nn	Replaces the item state table with the last structure pushed onto the stack.
Reserved	110001nn to 111101nn	For future use.

ID, contents, and format. This way, a transfer doesn't have to include every piece of data every time. Often, however, the simplicity of having a single report is more important than the need to reduce the bandwidth used by longer reports.

In a report descriptor, a Report ID item applies to all items that follow until the next Report ID. If there is no Report ID item, the default ID of zero is assumed. A descriptor should not declare a Report ID of zero. The Report IDs are specific to each report type, so a HID can have one report of each type with the default ID. However, if at least one report type uses multiple report IDs, every report in the HID must have a declared ID. For example, if an interface supports Report ID 1 and Report ID 2 for Feature reports, any Input or Output reports must also have a Report ID greater than zero.

In a transfer that uses a Set_Report or Get_Report request, the host specifies a report ID in the Setup transaction, in the low byte of the wValue field. In an interrupt transfer, if the interface supports more than one report ID, the report ID precedes the report data on the bus. If the interface supports only the default report ID of zero, the report ID isn't sent with the report when using interrupt transfers.

Under Windows, the report buffer provided to an API call must be large enough to hold the report plus one byte for the report ID. When a HID supports multiple report IDs for Input reports of different sizes, the Windows HID driver requires applications to use a buffer large enough to hold the longest report. When the HID supports multiple reports of the same type and different sizes and the HID is sending a report whose data is a multiple of the endpoint's maximum packet size, the HID indicates the end of the report data by sending a zero-length data packet

For Input reports when there are multiple Input Report IDs, the host's driver has no way to request a specific report from the HID. On receiving the IN token packet, the device returns whatever report is in its buffer. In other words, the device firmware decides which report to send. At the host, the HID driver stores the received report and its report ID in a buffer.

Describing the Data's Use

The Global items that describe the data and how it will be used are Usage Page, Logical and Physical Maximums and Minimums, Unit, and Unit Exponent. Each of these items helps the receiver of the report interpret the report's data. All but the Usage Page are involved with converting raw report data to values with units attached. The items make it possible for a report to contain data in a compact form, with the receiver of the data responsible for converting the data to meaningful values. However, the sender of the report data may instead choose to do some or all of the converting.

Usage Page. An item's Usage is a 32-bit value that identifies a function that a device performs. A Usage contains two values: the upper 16 bits are a Global Usage Page item and the lower 16 bits are a Local Usage item. The value in the Local Usage item is a Usage ID. The term *Usage* can refer to either the 32-bit value or the 16-bit Local value. To prevent confusion, some sources use the term Extended Usage to refer to the 32-bit value. In Microsoft's documentation, the USAGE type is a 16-bit value that can contain a Usage Page or a Usage ID.

Multiple items can share a Usage Page while having different Usage IDs. After a Usage Page appears in a report, all Usage IDs that follow use that Usage Page until a new Usage Page is declared.

The HID Usage Tables document lists the defined Usage Pages and their values and also names the section or other document that describes each page and its indexes. There are Usage Pages for many common device types, including generic desktop controls (mouse, keyboard, joystick), digitizer, bar-code scanner, camera control, and various game controls. In specialized devices that don't have a defined Usage Page, a vendor can define the Usage Page using values from FF00h to FFFFh.

Logical Minimum and Logical Maximum. The Logical Minimum and Logical Maximum define the limits for reported values. The limits are expressed in "logical units," which means that they use the same units as the values they describe. For example, if a device reports readings of up to 500 milliamperes in units of 2 milliamperes, the Logical Maximum is 250.

If the most significant bit of the highest byte is 1, the value is negative, expressed as a two's complement. (To find the negative value represented by a two's complement, complement each bit and add 1 to the result.) Using 1-byte values, 00h to 7Fh are the positive decimal values 0 through 127, and FFh to 80h are the negative decimal values -1 through -128.

The HID specification says that if both the Logical Minimum and Logical Maximum are considered positive, there's no need for a sign bit. But the report-descriptor test in the USB-IF Compliance Tool assumes that if the most-significant bit is 1, the value is negative. These values will fail the compliance test because the Logical Minimum (0) is greater than the Logical Maximum (-1):

```
0x15 0x00     // Logical Minimum
0x25 0xFF     // Logical Maximum WRONG!
```

If the desired result is a minimum of zero and a maximum of 255, the solution is to use a 2-byte value for the maximum:

```
0x15 0x00     // Logical Minimum
0x26 0x00FF   // Logical Maximum
```

Note that the Logical Maximum item tag is now 26h to indicate that the data that follows the tag is two bytes. Because the most-significant bit of the Logical Maximum is zero, the value is assumed to be positive and the compliance test accepts the values as valid.

Converting Units

The Physical Minimum, Physical Maximum, Unit Exponent, and Unit items define how to convert reported values into more meaningful units.

Physical Minimum and Physical Maximum. The Physical Minimum and Physical Maximum define the limits for a value when expressed in the units defined by the Units tag. In the earlier example of values of 0 through 250 in units of 2 milliamperes, the Physical Minimum is 0 and the Physical Maximum is 500. The receiving device uses the logical and physical limit values to obtain the value in the desired units. In the example, reporting the data in units of 2 milliamperes means that the value can transfer in a single byte, with the receiver of the data using the Physical Minimum and Maximum values to translate to milliamperes. The price is a loss in resolution, compared to reporting 1 bit per milliampere. If the report doesn't specify the values, they default to the same as the Logical Minimum and Logical Maximum.

Unit Exponent. The Unit Exponent specifies what power of 10 to apply to the value obtained after using the logical and physical limits to convert the value into the desired units. The exponent can range from -8 to +7. A value of 0 causes the value to be multiplied by 10^0, or 1, which is the same as applying no exponent. These are the codes:

Exponent	0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
Code	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh

For example, if the value obtained is 1234 and the Unit Exponent is 0Eh, the final value is 12.34.

Unit. The Unit tag specifies what units to apply to the report data after the value is converted using the Physical and Unit Exponent items. The HID specification defines codes for the basic units of length, mass, time, temperature, current, and luminous intensity. Most other units can be derived from these.

Specifying a Unit value can be more complicated than you might expect. Table 12-4 shows values you can work from. The value can be as long as four bytes, with each nibble having a defined function. Nibble 0 (the least significant nibble) specifies the measurement system, either English or SI (International System of Units) and whether the measurement is in linear or angular units. Each of the nibbles that follow represents a quality to be measured, with the value of the nibble representing the exponent to apply to the value. For example, a nibble with a value of 2 means that the corresponding value is in units squared. A nibble with a value of Dh, which represents -3, means that the units are expressed as $1/\text{units}^3$. These exponents are separate from the Unit Exponent value, which is a power of ten applied to the data, rather than an exponent applied to the units.

Converting Raw Data

To convert raw data to values with units attached, three things must occur. The firmware's report descriptor must contain the information needed for the conversion. The sender of the data must send data that matches the specification in the descriptor. And the receiver of the data must apply the conversions specified in the descriptor.

Below are examples of descriptors and raw and converted data. Remember that just because a tag exists in the HID specification doesn't mean you have to use it. If the application knows what format and units to use for the values it's going to send or receive, the firmware doesn't have to specify these items.

To specify time in seconds, up to a minute, the report descriptor might include this information:

Logical Minimum: 0
Logical Maximum: 60

Table 12-4: The units to apply to a reported value are a function of the measuring system and exponent values specified in the Unit item

Nibble Number	**Quality Measured**	**Measuring System (Nibble 0 value)**				
		None (0)	**SI Linear (1)**	**SI Rotation (2)**	**English Linear (3)**	**English Rotation (4)**
1	**Length**	None	Centimeters	Radians	Inches	Degrees
2	**Mass**	None	Grams		Slugs	
3	**Time**	None	Seconds			
4	**Tempera-ture**	None	Kelvin		Fahrenheit	
5	**Current**	None	Amperes			
6	**Luminous Intensity**	None	Candelas			
7	**Reserved**	None				

Physical Minimum: 0

Physical Maximum: 60

Unit: 1003h. Nibble 0 = 3 to select the English Linear measuring system (though in this case, any value from 1 to 4 would work). Nibble 3 = 1 to select time in seconds.

Unit Exponent: 0

With this information, the receiver knows that the value sent equals a number of seconds.

Now, what if instead you want to specify time in tenths of seconds, again up to a minute? You would need to increase the Logical and Physical Maximums and change the Unit Exponent:

Logical Minimum: 0

Logical Maximum: 600

Physical Minimum: 0

Physical Maximum: 600

Unit: 1003h. Nibble 0 = 3 to select the English Linear measuring system. Nibble 3 = 1 to select time in seconds.

Unit Exponent: 0Fh. This represents an exponent of -1, to indicate that the value is expressed in tenths of seconds rather than seconds.

Sending values as large as 600 will require 3 bytes, which the firmware specifies in the Report Size tag.

To send a temperature value using one byte to represent temperatures from -20 to 110 degrees Fahrenheit, the report descriptor might contain the following:

> Logical Minimum: -128 (80h when expressed in hexadecimal as a two's complement)
>
> Logical Maximum: 127 (7Fh)
>
> Physical Minimum: -20 (ECh when expressed in hexadecimal as a two's complement)
>
> Physical Maximum: 110 (6Eh)
>
> Unit: 10003h. Nibble 0 is 3 to select the English Linear measuring system, though in this case, any value from 1 to 4 is OK. Nibble 4 is 3 to select degrees Fahrenheit.
>
> Unit Exponent: 0

These values ensure the highest possible resolution for a single-byte report item, because the transmitted values can span the full range from 0 to 255.

In this case the logical and physical limits differ, so converting is required. This Visual-Basic code finds the resolution, or number of bits per unit:

```
Resolution = _
  (Logical_Maximum - Logical_Minimum) / _
  ((Physical_Maximum - Physical_Minimum) * _
  (10 ^ Unit_Exponent))
```

With the example values, the resolution is 1.96 bits per degree, or 0.51 degree per bit.

This Visual-Basic code converts a value to the specified units:

```
Value = _
  Value_In_Logical_Units * _
  ((Physical_Maximum - Physical_Minimum) * _
  (10 ^ Unit_Exponent )) / _
  (Logical_Maximum - Logical_Minimum)
```

If the value in logical units (the raw data) is 63, the converted value in the specified units is 32 degrees Fahrenheit.

As another example, specifying velocity in centimeters per second requires a Unit value that contains units of both centimeters and seconds. From Table 12-4, the Unit value to use is 1011h. Nibble 0 = 1 to select the SI measuring system, nibble 1 = 1 to select length in centimeters, and nibble 3 = 1 to select time in seconds.

To show how complicated it can get, the Unit value for volts is F0D121h, which indicates the SI Linear measuring system in units of $(cm^{2})(gm)/(sec^{-3})(amp^{-1})$. However, remember that the Unit value only specifies the units. All the receiver has to do is identify the Units value and assign the units to received data; there's no need to do the calculations implied in the Units value.

Describing the Data's Size and Format

Two Global items describe the size and format of the report data.

Report Size specifies the size in bits of a field in an Input, Output, or Feature item. Each field contains one piece of data.

Report Count specifies how many fields an Input, Output, or Feature item contains.

For example, if a report has two 8-bit fields, Report Size is 8 and Report Count is 2. If a report has one 16-bit field, Report Size is 16 and Report Count is 1.

A single Input, Output, or Feature report can have multiple items, each with its own Report Size and Report Count.

Saving and Restoring Global Items

The final two Global items enable saving and restoring sets of Global items. These items allow flexibility in the report formats while using minimum storage space in the device.

Push places a copy of the Global-item state table on the CPU's stack. The Global-item state table contains the current settings for all previously defined Global items.

Pop is the complement to Push. It restores the saved states of the previously pushed Global item states.

The Local Item Type

Local items specify qualities of the controls and data items in a report. A Local item's value applies to all items that follow within a Main item until a new value is assigned. Local items don't carry over to the next Main item; each Main item begins fresh, with no Local items defined.

Local items relate to general usages, body-part designators, and strings. A Delimiter item enables grouping sets of Local items. Table 12-5 shows the values and meaning of each of the items.

Usage. The Local Usage item is the Usage ID that works together with the Global Usage Page to describe the function of a control, data, or collection.

As with the Usage Page item, the HID Usage Tables document lists many Usage IDs. For example, the Buttons Usage Page uses Local Usage IDs from 1 to FFFFh to identify which buttons in a set is pressed, with a value of 0 meaning no button pressed.

A report may assign one Usage to multiple items. If a report item is preceded by a single Usage, that Usage applies to all of the item's data. If a report item is preceded by more than one Usage and the number of controls or data items equals the number of Usages, each Usage applies to one control or data item, with the Usages and controls/data items pairing up in sequence. In the following example, the report contains two bytes. The first byte's Usage is X, and the second byte's Usage is Y.

```
Usage (X),
Usage (Y),
Report Count (2),
Report Size (8),
Input (Data, Variable, Absolute),
```

If a report item is preceded by more than one Usage and the number of controls or data items is greater than the number of Usages, each Usage pairs up with one control or data item in sequence, and the final Usage applies to all

Table 12-5: There are ten defined Local items.

Local Item Type	Value (nn indicates the number of bytes that follow)	Description
Usage	000010nn	An index that describes the use for an item or collection.
Usage Minimum	000110nn	The starting Usage associated with the elements in an array or bitmap.
Usage Maximum	001010nn	The ending Usage associated with the elements in an array or bitmap.
Designator Index	001110nn	A Designator value in a physical descriptor. Indicates what body part applies to a control.
Designator Minimum	010010nn	The starting Designator associated with the elements in an array or bitmap.
Designator Maximum	010110nn	The ending Designator associated with the elements in an array or bitmap.
String Index	011110nn	Associates a string with an item or control.
String Minimum	100010nn	The first string index when assigning a group of sequential strings to controls in an array or bitmap.
String Maximum	100110nn	The last string index when assigning a group of sequential strings to controls in an array or bitmap.
Delimiter	101010nn	The beginning (1) or end (0) of a set of Local items.
Reserved	101011nn to 111110nn	For future use.

of the remaining controls/data items. In the following example, the report is 16 bytes. Usage X applies to the first byte, Usage Y applies to the second byte, and a vendor-defined Usage applies to the third through 16th bytes.

```
Usage (X)
Usage (Y)
Usage (vendor defined)
Report Count (16),
Report Size (8),
Input (Data, Variable, Absolute)
```

Usage Minimum and Maximum. The Usage Minimum and Usage Maximum can assign a series of Usage IDs to the elements in an array or bitmap. The following example describes a report that contains the state (0 or 1) of each of three buttons. The Usage Minimum and Usage Maximum specify that the first button has a Usage ID of 1, the second button has a Usage ID of 2, and the third button has a Usage ID of 3:

```
Usage Page (Button Page)
Logical Minimum (0)
Logical Maximum (1)
Usage Minimum (1)
Usage Maximum (3)
Report Count (3)
Report Size (1)
Input (Data, Variable, Absolute)
```

Designator Index. For items with a physical descriptor, the Designator Index specifies a Designator value in a physical descriptor. The Designator specifies what body part the control uses.

Designator Minimum and Designator Maximum. When a report contains multiple Designator Indexes that apply to the elements in a bitmap or array, a Designator Minimum and Designator Maximum can assign a sequential Designator Index to each bit or array item.

String Index. An item or control can include a String Index to associate a string with the item or control. The strings are stored in the same format described in Chapter 4 for product, manufacturer, and serial-number strings.

String Minimum and Maximum. When a report contains multiple string indexes that apply to the elements in a bitmap or array, a String Minimum and String Maximum can assign a sequential String Index to each bit or array item.

Delimiter. A Delimiter defines the beginning (1) or end (0) of a local item. A delimited local item may contain alternate usages for a control. Different applications can thus define a device's controls in different ways. For example, a button may have a generic use (Button1) and a specific use (Send, Quit, etc.).

Physical Descriptors

A physical descriptor specifies the part or parts of the body intended to activate a control. For example, each finger might have its own assigned control. Similar physical descriptors are grouped into a physical descriptor set. A set consists of a header, followed by the physical descriptors.

A physical descriptor is a HID-specific descriptor. The host can retrieve a physical descriptor set by sending a Get_Descriptor request with 23h in the high byte of the wValue field and the number of the descriptor set in the low byte of the wValue field.

Physical descriptors are optional. For most devices, these descriptors either don't apply at all or the information they could provide has no practical use. The HID specification has more information on how to use physical descriptors, for those devices that need them.

Padding

To pad a descriptor so it contains a multiple of eight bits, a descriptor can include a Main item with no assigned Usage. The following excerpt from a keyboard's report descriptor specifies an Output report that transfers five bits of data and three bits of padding:

```
Usage Page (LEDs)
Usage Minimum (1)
Usage Maximum (5)
Output (Data, Variable, Absolute) (5 1-bit LEDs)

Report Count (1)
Report Size (3)
Output (Constant) (3 bits of padding)
```