The website of Frank Zhao, electrical engineer and DIY hobbyist

Tutorial about USB HID Report Descriptors

This page is from my old website, and it is sort of popular, so I've moved it here.

A USB HID report descriptor is one of the descriptors that a USB host can request from a USB device. HID devices send data to the host using reports, and the descriptor tells the host how to interpret the data. I will try to show you how to write one of these descriptors.

First, go to this page http://www.usb.org/developers/hidpage/ and find the document titled "Device Class Definition for HID". What I will be talking about is essentially paraphrasing the important sections of that document.

Thanks for visiting! If you appreciate my content, please consider making a donation to a charity. Thank you ~ Frank

Second, go get the HID descriptor tool from the same page. You'll want to play with it as you go through this tutorial. It is an absolute headache to write the HID report descriptors manually (converting between binary and hex and looking up the meanings of the numbers) so this tool is essential.

What is a USB HID report descriptor?

The HID protocol makes implementation of devices very simple. Devices define their data packets and then present a "HID descriptor" to the host. The HID descriptor is a hard coded array of bytes that describe the device's data packets. This includes: how many packets the device supports, how large are the packets, and the purpose of each byte and bit in the packet. For example, a keyboard with a calculator program button can tell the host that the button's pressed/released state is stored as the 2nd bit in the 6th byte in data packet number 4 (note: these locations are only illustrative and are device specific). The device typically stores the HID descriptor in ROM and does not need to intrinsically understand or parse the HID descriptor. Some mouse and keyboard hardware in the market today are implemented using only an 8-bit CPU.

– Wikipedia on Human Interface Device

I'm going to try teaching you about USB HID report descriptors by walking you through writing a few.

For a simple starting point, let us make a standard mouse. Just three buttons, and movement on the X and Y axis. So we want to send data regarding the buttons and movement. It takes one bit to represent each button, and one byte to represent the movement on one axis as a signed integer. So we can say that we want the data structure to look something like this

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Useless	Useless	Useless	Useless	Useless	Left Button	Middle Button	Right Button
Byte 1	X Axis Relative Movement as Signed Integer							
Byte 2	Y Axis Relative Movement as Signed Integer							

And then we can say our data structure in C looks like

```
struct mouse_report_t
{
    uint8_t buttons;
    int8_t x;
    int8_t y;
}
```

So now in our descriptor, our first item must describe buttons, three of them

```
USAGE_PAGE (Button)
USAGE_MINIMUM (Button 1)
USAGE_MAXIMUM (Button 3)
```

each button status is represented by a bit, 0 or 1

```
LOGICAL_MINIMUM (0)
LOGICAL_MAXIMUM (1)
```

there are three of these bits

```
REPORT_COUNT (3)
REPORT_SIZE (1)
```

send this variable data to the computer

```
INPUT (Data,Var,Abs)
```

and the final result looks like

```
USAGE_PAGE (Button)
USAGE_MINIMUM (Button 1)
USAGE_MAXIMUM (Button 3)
LOGICAL_MINIMUM (0)
LOGICAL_MAXIMUM (1)
REPORT_COUNT (3)
REPORT_SIZE (1)
INPUT (Data,Var,Abs)
```

that will represent the buttons

but what about the five useless padding bits?

```
REPORT_COUNT (1)
REPORT_SIZE (5)
INPUT (Cnst,Var,Abs)
```

now we make the X axis movement

```
USAGE_PAGE (Generic Desktop)
USAGE (X)
```

we want it to be a signed integer that takes one byte, so it has a value between -127 and +127 (actually -128 and +127, but I want to keep things even)

```
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
```

we want it to take an entire byte which is 8 bits

```
REPORT_SIZE (8)
REPORT_COUNT (1)
```

and send it to the computer as a variable relative coordinate

```
INPUT (Data,Var,Rel)
```

you end up with something like this to represent the X axis movement

```
USAGE_PAGE (Generic Desktop)
USAGE (X)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (1)
INPUT (Data,Var,Rel)
```

How about Y axis? You can try

```
USAGE_PAGE (Generic Desktop)
USAGE (X)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (1)
INPUT (Data,Var,Rel)
USAGE_PAGE (Generic Desktop)
USAGE (Y)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (1)
INPUT (Data,Var,Rel)
```

Which will work, but to save memory, we can do this instead

```
USAGE_PAGE (Generic Desktop)
USAGE (X)
USAGE (Y)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (2)
INPUT (Data,Var,Rel)
```

So all your data will end up looking like

```
USAGE_PAGE (Button)
USAGE_MINIMUM (Button 1)
USAGE_MAXIMUM (Button 3)
LOGICAL_MINIMUM (0)
LOGICAL_MAXIMUM (1)
REPORT_COUNT (3)
REPORT_SIZE (1)
INPUT (Data,Var,Abs)
REPORT_COUNT (1)
REPORT_SIZE (5)
INPUT (Cnst,Var,Abs)
USAGE_PAGE (Generic Desktop)
USAGE (X)
USAGE (Y)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (2)
INPUT (Data,Var,Rel)
```

Ah but we are not done, in order to make the computer know that this is a mouse, we do

```
USAGE_PAGE (Generic Desktop)
USAGE (Mouse)
COLLECTION (Application)
    USAGE (Pointer)
    COLLECTION (Physical)

    ... What we wrote already goes here

    END COLLECTION
END COLLECTION
```

So in the end, this is the USB HID report descriptor for a standard mouse

```
0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
0x09, 0x02,                    // USAGE (Mouse)
0xa1, 0x01,                    // COLLECTION (Application)
0x09, 0x01,                    //   USAGE (Pointer)
0xa1, 0x00,                    //   COLLECTION (Physical)
0x05, 0x09,                    //     USAGE_PAGE (Button)
0x19, 0x01,                    //     USAGE_MINIMUM (Button 1)
0x29, 0x03,                    //     USAGE_MAXIMUM (Button 3)
0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
0x25, 0x01,                    //     LOGICAL_MAXIMUM (1)
0x95, 0x03,                    //     REPORT_COUNT (3)
0x75, 0x01,                    //     REPORT_SIZE (1)
0x81, 0x02,                    //     INPUT (Data,Var,Abs)
0x95, 0x01,                    //     REPORT_COUNT (1)
0x75, 0x05,                    //     REPORT_SIZE (5)
0x81, 0x03,                    //     INPUT (Cnst,Var,Abs)
0x05, 0x01,                    //     USAGE_PAGE (Generic Desktop)
0x09, 0x30,                    //     USAGE (X)
0x09, 0x31,                    //     USAGE (Y)
0x15, 0x81,                    //     LOGICAL_MINIMUM (-127)
0x25, 0x7f,                    //     LOGICAL_MAXIMUM (127)
0x75, 0x08,                    //     REPORT_SIZE (8)
0x95, 0x02,                    //     REPORT_COUNT (2)
0x81, 0x06,                    //     INPUT (Data,Var,Rel)
0xc0,                          //   END_COLLECTION
0xc0                           // END_COLLECTION
```

This is actually the example descriptor provided with the USB HID documentation, and you can also find this as an example provided with the HID tool.

Cool, at this point, you will have encountered some concepts that you may have questions about, you should research the following:

Usage Pages

There's one thing that I think isn't explained well in the documentation, USAGE, USAGE_PAGE, USAGE_MINIMUM and USAGE_MAXIMUM. In a descriptor, you first set a USAGE_PAGE, and certain USAGEs are available. In the mouse example, USAGE_PAGE (Generic Desktop) allowed you to use USAGE (Mouse), and when the usage page was changed to USAGE_PAGE (Button), then the USAGE_MINIMUM and USAGE_MAXIMUM allowed you to specify the buttons, and before you can use USAGE (X) and USAGE (Y), the usage page was changed back to USAGE_PAGE (Generic Desktop). The usage page is like a name-space, changing the usage page affects what "usages" are available. Read the documentation called " HID Usage Tables" for more info.

Collections

Read the documentation about the official proper use of collections. In my own words, collections can be used to organize your data, for example, a keyboard may have a built-in touchpad, then the data for the keyboard should be kept in one application collection while the touchpad data is kept in another. We can assign an "Report ID" to each collection, which I will show you later.

Hey here's something you can do, by turning "USAGE (Mouse)" into "USAGE (Gamepad)", you make the computer think that it's a game pad with one joystick and 3 buttons. How about converting a Playstation 2 controller into a USB gamepad? The controller has 16 buttons and two thumb sticks, so we want the data to look like

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Button	Button	Button	Button	Button	Button	Button	Button
Byte 1	Button	Button	Button	Button	Button	Button	Button	Button
Byte 2	Left X Axis as Signed Integer							
Byte 3	Left Y Axis as Signed Integer							
Byte 4	Right X Axis as Signed Integer							
Byte 5	Right Y Axis as Signed Integer							

So our data structure looks like

```c
struct gamepad_report_t
{
    uint16_t buttons;
    int8_t left_x;
    int8_t left_y;
    int8_t right_x;
    int8_t right_y;
}
```

We make the computer understand that it's a game pad

```
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)

    ...

    END COLLECTION
END COLLECTION
```

for the buttons

```
USAGE_PAGE (Button)
USAGE_MINIMUM (Button 1)
USAGE_MAXIMUM (Button 16)
LOGICAL_MINIMUM (0)
LOGICAL_MAXIMUM (1)
REPORT_COUNT (16)
REPORT_SIZE (1)
INPUT (Data,Var,Abs)
```

for the four thumb stick axis

```
USAGE_PAGE (Generic Desktop)
USAGE (X)
USAGE (Y)
USAGE (Z)
USAGE (Rx)
LOGICAL_MINIMUM (-127)
LOGICAL_MAXIMUM (127)
REPORT_SIZE (8)
REPORT_COUNT (4)
INPUT (Data,Var,Abs)
```

NOTE: Z is used to represent the right stick's X axis, Rx is used to represent the right stick's Y axis. This doesn't make sense but this is how most existing USB game pads work. I have tested this using Battlefield Bad Company 2, it works.

NOTE: Use "absolute" for something like joysticks, but "relative" for things like mouse.

So now you end up with

```
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        USAGE_PAGE (Button)
        USAGE_MINIMUM (Button 1)
        USAGE_MAXIMUM (Button 16)
        LOGICAL_MINIMUM (0)
        LOGICAL_MAXIMUM (1)
        REPORT_COUNT (16)
        REPORT_SIZE (1)
        INPUT (Data,Var,Abs)
        USAGE_PAGE (Generic Desktop)
        USAGE (X)
        USAGE (Y)
        USAGE (Z)
        USAGE (Rx)
        LOGICAL_MINIMUM (-127)
        LOGICAL_MAXIMUM (127)
        REPORT_SIZE (8)
        REPORT_COUNT (4)
        INPUT (Data,Var,Abs)
    END COLLECTION
END COLLECTION
```

Hey how about two players? Here’s where collections get handy

```
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        REPORT_ID (1)
        ...
    END COLLECTION
END COLLECTION
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        REPORT_ID (2)
        ...
    END COLLECTION
END COLLECTION
```

fill in the data areas and you end up with

```
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        REPORT_ID (1)
        USAGE_PAGE (Button)
        USAGE_MINIMUM (Button 1)
        USAGE_MAXIMUM (Button 16)
        LOGICAL_MINIMUM (0)
        LOGICAL_MAXIMUM (1)
        REPORT_COUNT (16)
        REPORT_SIZE (1)
        INPUT (Data,Var,Abs)
        USAGE_PAGE (Generic Desktop)
        USAGE (X)
        USAGE (Y)
        USAGE (Z)
        USAGE (Rx)
        LOGICAL_MINIMUM (-127)
        LOGICAL_MAXIMUM (127)
        REPORT_SIZE (8)
        REPORT_COUNT (4)
        INPUT (Data,Var,Abs)
    END COLLECTION
END COLLECTION
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        REPORT_ID (2)
        USAGE_PAGE (Button)
        USAGE_MINIMUM (Button 1)
        USAGE_MAXIMUM (Button 16)
        LOGICAL_MINIMUM (0)
        LOGICAL_MAXIMUM (1)
        REPORT_COUNT (16)
        REPORT_SIZE (1)
        INPUT (Data,Var,Abs)
        USAGE_PAGE (Generic Desktop)
        USAGE (X)
        USAGE (Y)
        USAGE (Z)
        USAGE (Rx)
        LOGICAL_MINIMUM (-127)
        LOGICAL_MAXIMUM (127)
        REPORT_SIZE (8)
        REPORT_COUNT (4)
        INPUT (Data,Var,Abs)
    END COLLECTION
END COLLECTION
```

This is really important: You must change your data structure to include the report ID

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Report ID							
Byte 1	Button	Button	Button	Button	Button	Button	Button	Button
Byte 2	Button	Button	Button	Button	Button	Button	Button	Button
Byte 3	Left X Axis as Signed Integer							
Byte 4	Left Y Axis as Signed Integer							
Byte 5	Right X Axis as Signed Integer							
Byte 6	Right Y Axis as Signed Integer							

```
struct multiplayer_gamepad_report_t
{
    uint8_t report_id;
    uint16_t buttons;
    int8_t left_x;
    int8_t left_y;
    int8_t right_x;
    int8_t right_y;
}
```

You must manually set the report ID before you send the data to the computer in order for the computer to understand which player the data belongs to.

```
multiplayer_gamepad_report_t player1_report;
multiplayer_gamepad_report_t player2_report;
player1_report.report_id = 1;
player2_report.report_id = 2;
```

You can also use collections and report IDs to make composite devices. So far I've shown you the keyboard, mouse, and gamepad. Here's something that describes a composite device that is a keyboard, mouse, and two player game pad.

```
USAGE_PAGE (Generic Desktop)
USAGE (Keyboard)
COLLECTION (Application)
    REPORT_ID (1)
    ...
END COLLECTION
USAGE_PAGE (Generic Desktop)
USAGE (Mouse)
COLLECTION (Application)
    USAGE (Pointer)
    COLLECTION (Physical)
        REPORT_ID (2)
        ...
    END COLLECTION
END COLLECTION
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
```

```
        REPORT_ID (3)
        ...
    END COLLECTION
END COLLECTION
USAGE_PAGE (Generic Desktop)
USAGE (Game Pad)
COLLECTION (Application)
    COLLECTION (Physical)
        REPORT_ID (4)
        ...
    END COLLECTION
END COLLECTION
```

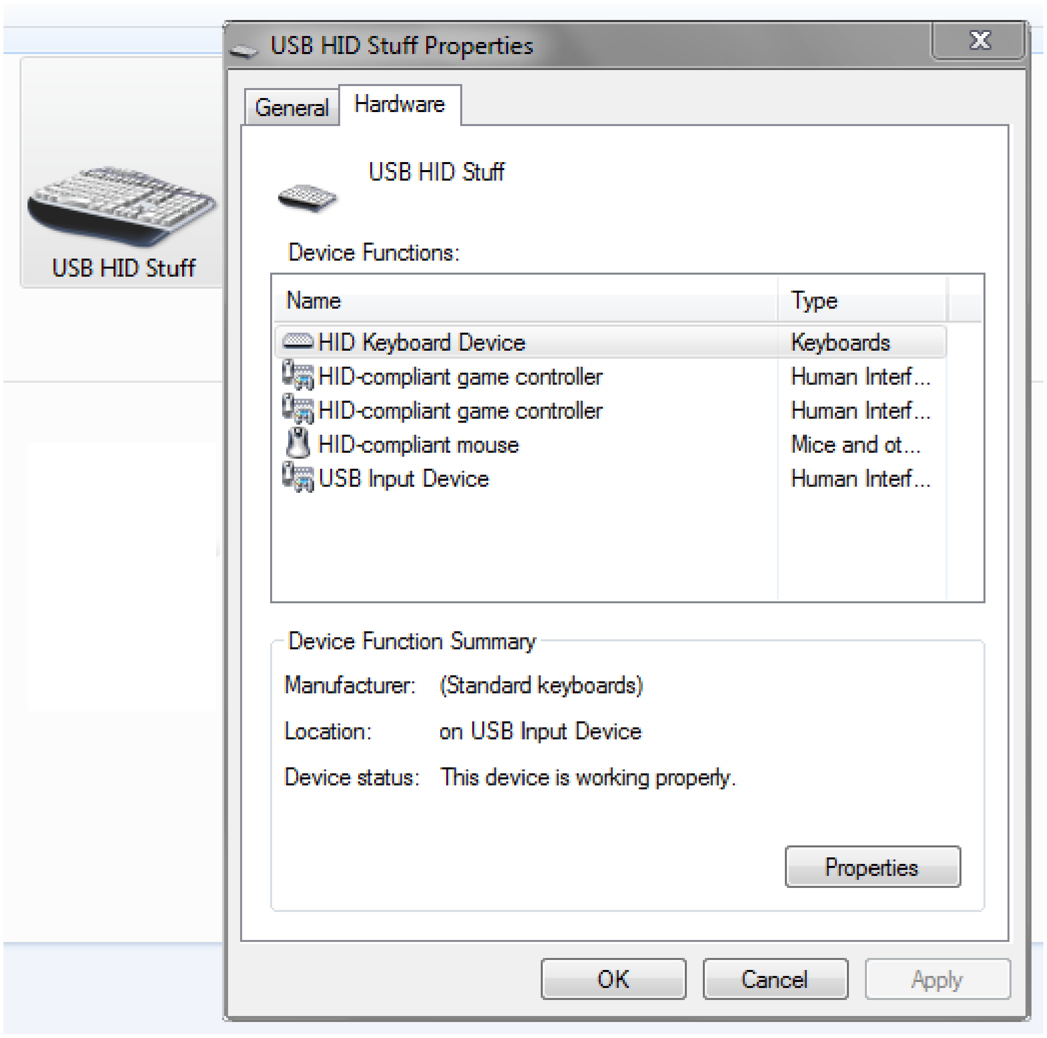
and of course, your data structures with the added report ID.

```
struct keyboard_report_t
{
    uint8_t report_id;
    uint8_t modifier;
    uint8_t reserved;
    uint8_t keycode[6];
}

struct mouse_report_t
{
    uint8_t report_id;
    uint8_t buttons;
    int8_t x;
    int8_t y;
}

struct gamepad_report_t
{
    uint8_t report_id;
    uint16_t buttons;
    int8_t left_x;
    int8_t left_y;
    int8_t right_x;
    int8_t right_y;
}
```

But since we changed the data structure, your device no-longer supports boot protocol, and hence you will not need to define a protocol. So make sure you change usbconfig.h accordingly.

Want to see this in action? Load up this example project into USnooBie and let Windows's "Devices and Printers" show you!

Example Project Files



USB HID Stuff

Device Functions:

Name	Type
HID Keyboard Device	Keyboards
HID-compliant game controller	Human Interf...
HID-compliant game controller	Human Interf...
HID-compliant mouse	Mice and ot...
USB Input Device	Human Interf...

Device Function Summary

Manufacturer:	(Standard keyboards)
Location:	on USB Input Device
Device status:	This device is working properly.

This entry was posted in Tutorial and tagged usb on January 1, 2013 [https://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/] .

133 thoughts on “Tutorial about USB HID Report Descriptors”

Happy Fellow
December 2, 2013 at 7:17 PM

God bless you 🙂

CT
December 25, 2013 at 3:20 PM

Awesome tutorial…Thanks. CT