Create a HID USB Device (sample Joystick)
By Jared St.Clare, October 2009
Version 1

1. Use EasyHid to create the generic files and USB files. Select the correct pic chip that you plan to program. We're going to call this project **Racewheel**.

2. If this is not the first time using EasyHid – you need to generate a new PID serial number. Default values should be ok.

3. EasyHid will create files in two folders. We want all the files in the PicBasicPro folder. These can be moved to your working directory. The other folder (VisualBasic) can be deleted.

4. Of all the files you just copied, we're interested in just two:
   - **Racewheel.pbp** – main program (or hardware config)
   - **DescRacewheel.asm** – our descriptors (or software config).

5. At this point we need a plan on what our device will do (i.e. number of buttons, axis etc). For this joystick we'll use 4 buttons and 2 axis.

6. First, we will be using the **Descriptor Tool** to build our descriptor report. A report descriptor is built from scratch by adding Hid Items. Eventually we will copy the completed report descriptor into our DescRacewheel.asm file.

7. The report must always follow this general format:

   - Usage Page(Generic Desktop)
   - Usage (…type of hardware…)
   - Collection (Application)
   - Usage (…type of hardware…)
   - Collection (Physical)
   - Usage [Page] (…featured part of hardware…)
   - Logical Min
   - Logical Max
   - Usage Min
   - Usage Max
   - **Report Size**
   - **Report Count**
   - Input (data, Var, Abs)
   - Usage [Page) (…featured next part of hardware…)
   - … Same format etc etc …
   - End Collection
   - End Collection

I've found that it is possible to leave some items out and your device will still work, however, the more you describe about the device the better chances you have that Windows and more importantly other software can use it without problems.

Report count, report Size,…. Do yours add up?

The USB data is sent in (multiple) byte (8 bit) blocks. The byte must be completely filled in else Windows will tell you your device isn't working properly.

For example, if we have 8 buttons we would send the on/off status of these buttons in a single 1 byte report block.

For 8 button reports:

- Report Count = 8 (8 buttons)
- Report Size = 1 (each button only requires 1 bit)

Button 0 is LSB, Button 8 is MSB

For 4 button reports:

- Report Count = 8 (4 buttons + 4 spare)
- Report Size = 1 (each button only requires 1 bit)

All 8 bits are still used, because the report block must be complete. We just ignore the 4 wasted bits.

This is a descriptor report for a 4 button joystick.

Report Descriptor

USAGE_PAGE (Generic Desktop)	05 01
USAGE (Joystick)	09 04
COLLECTION (Application)	A1 01
USAGE (Joystick)	09 04
COLLECTION (Physical)	A1 00
USAGE_PAGE (Button)	05 09
USAGE_MINIMUM (Button 1)	19 01
USAGE_MAXIMUM (Button 4)	29 04
LOGICAL_MINIMUM (0)	15 00
LOGICAL_MAXIMUM (1)	25 01
REPORT_COUNT (8)	95 08
REPORT_SIZE (1)	75 01
INPUT (Data,Var,Abs)	81 02
END_COLLECTION	C0
END_COLLECTION	C0

What about the analog axis reports?

Like above, the report block is sent as a 1 byte block. This makes things pretty straight forward, esp. if the analog value of our pot (axis) is 8 bit (0 – 255 range).

For a x2 axis report:

- Report Count = 2 (2 axis inputs eg X+Y)
- Report Size = 8 (8bits = 256 analog value of pot)

All 8 bits are used per report block, and the PIC is sending two blocks.

The rest of the report is pretty much a no brainer. Study the included example for a better understanding of the different Hid items and their values.

Here is our completed descriptor file for 4 buttons and 2 axis Joystick:

Report Descriptor

```
USAGE_PAGE (Generic Desktop)          05 01
USAGE (Joystick)                      09 04
COLLECTION (Application)              A1 01
  USAGE (Joystick)                    09 04
  COLLECTION (Physical)               A1 00
    USAGE (X)                         09 30
    USAGE (Y)                         09 31
    REPORT_SIZE (8)                   75 08
    REPORT_COUNT (2)                  95 02
    PHYSICAL_MINIMUM (0)              35 00
    PHYSICAL_MAXIMUM (255)            46 FF 00
    INPUT (Cnst,Var,Abs)              81 03
    USAGE_PAGE (Button)               05 09
    USAGE_MINIMUM (Button 1)          19 01
    USAGE_MAXIMUM (Button 4)          29 04
    LOGICAL_MINIMUM (0)               15 00
    LOGICAL_MAXIMUM (1)               25 01
    REPORT_SIZE (1)                   75 01
    REPORT_COUNT (8)                  95 08
    INPUT (Cnst,Var,Abs)              81 03
    END_COLLECTION                    C0
END_COLLECTION                        C0
```

8. Save the report to a .hid file for later use and save the report as a .h (header text file) so that we can edit in MicroStudio. Open up the .h header text file (desc1.h).

9. Step 1: use Replace to add '**RETLW 0x'** to all of the fields starting with 0x.
   Step 2: use Replace to change the **'//'** comments out to a '**;**'
   Step 3: use Replace to change all commas to a space
   Step 4: put each RETLW on it's own line
   Step 5: delete any other junk at top and bottom of the report.

   I use a batch file to do most of the editing for me.

This is what part of the edited descriptor looks like.

```
RETLW 0x05
RETLW 0x01                  ; USAGE_PAGE (Generic Desktop)
RETLW 0x09
RETLW 0x04                  ; USAGE (Joystick)
RETLW 0xa1
RETLW 0x01                  ; COLLECTION (Application)
RETLW 0x09
RETLW 0x04                  ;   USAGE (Joystick)
RETLW 0xa1
RETLW 0x00                  ;   COLLECTION (Physical)
RETLW 0x09
RETLW 0x30                  ;     USAGE (X)
RETLW 0x09
```

10. Now cut and paste all of the descriptor into the **DescRacewheel.asm file**. This must go between the [ReportDescriptor1]…. [EndReportDescriptor1] section (replacing all the data that was already there). Save this file.

11. Time to **edit the main program Racewheel.pbp** file. As stated before, I like to think of this as hardware setup.

    All the basic USB stuff has already been configured by EasyHid. We need to do a few things to finalize our program settings.

    - Configure hardware (ports, clock freq etc)
    - Modify the USBBufferSize based on our USB report descriptor

Configuring hardware is no different to any other pic program. Our project requires four digital inputs and two analog inputs.

Since the joystick will only be sending data to the pc, we can delete all references to DoUSBin and RxBuffer.

USB buffer size, how big should it be?

We have 4 buttons, and 2 axis. The buttons require 1 bit of data, and each axis requires 8 bits.

That's a total of 8 + 8 + 4 = 20 bits.

As the data is sent in 8 bit blocks, the USBbuffer requires 3 bytes (3x8bits = 24bits). The last 4 bits won't be used, sound familiar?

Byte 0: Bits 0 – 7 for Axis X
Byte 1: Bits 8 – 15 for Axis Y
Byte 2: Bits 16 – 19 for the buttons

This is what our program looks like for setting the buffer size:

```
USBBufferSizeMax    CON 3   ' maximum buffer size
USBBufferSizeTX     CON 3   ' input

USBBuffer         VAR BYTE[USBBufferSizeMax]
USBBufferCount    VAR BYTE
```

USBBuffer is a 3 byte array.

Now all we need to edit is the main program loop. This is what it looks like:

```
ProgramStart:
   ADCIN 0, USBBuffer[0]          ' axis1
   ADCIN 1, USBBuffer[1]          ' axis 2

   USBBuffer.0(16) = porta.2      ' button 1
   USBBuffer.0(17) = porta.3      ' button 2
   USBBuffer.0(18) = porta.4      ' button 3
   USBBuffer.0(19) = porta.5      ' button 4

   GOSUB DoUSBOut
   GOTO ProgramStart
```

So our program loops through, reading the analog value of X axis, it's 8 bit value is stored into the first byte of our array, Y axis, stored in the second byte of the array, and the 4 buttons, stored the third byte of the array.

Finishing up….

Compile the program and run. Do Dee! Windows should detect and install the RaceWheel and you can monitor this via opening joysticks in the control panel.

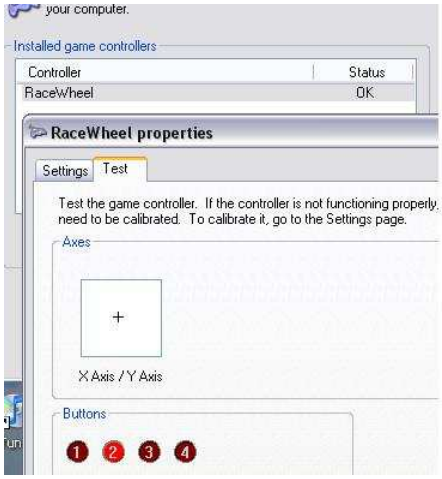
Disconnect the Pic and our RaceWheel is removed from the list of joysticks. Dee Do!

What if it doesn't work? Do Do! Or device not working properly.

If you get the dreaded Do Do! sound, leave the pic connected, open up windows device manager, expand the Human Interface Devices group and delete the Human Interface Device that corresponds to our joystick. Disconnect the pic and re-connect. This should fix it.

If you still get device not working properly, there is a problem with your descriptor settings or your pic isn't configured properly.

RaceWheel Connected and ready to use:



I hope this little demonstration gives you a foot in the right direction. Try adding a throttle axis or some more buttons. For you flight simmers, how about a 64 button overhead panel or make your own rudder pedals?

Enjoy!.

Jared St.Clare