Joystick and Gamepad Input

Rod

The port

Your PC supports quite a variety of input devices. In modern parlance they are called HIDs (Human Input Devices), keyboards, mice, graphic pads, touchscreens and an old stalwart, the joystick or game pad. Now there use to be a dedicated joystick port , latterly called a game port, on every PC.

That has fallen away and since Vista the game port is no longer supported. It matters not a whit to Liberty BASIC as it is perfectly capable of reading and dealing with HID devices which are most often connected via USB, (Universal Serial Bus).

In most cases the native command set is all that you will need. READJOYSTICK (n) will read the current status of joystick (n). (n) can be 1 or 2. So you can plug in two, USB based, standard two or three axis joysticks and use their input very easily in your Liberty BASIC programs.

But it does not end there. With API calls you can read in up to six axis of movement from up to sixteen devices and on top of that read in the status of up to sixteen buttons from each device!

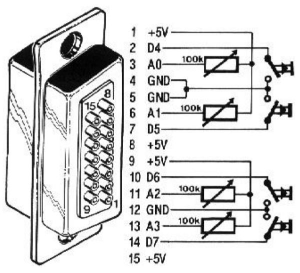
Simple Joystick

A simple joystick will usually support two axis of movement, X and Y and two buttons. The axis of movement are returned as numbers between 0 and 65535 (256*256). 0 means fully down, 32767 or thereabouts means centred and 65535 means fully up. I say thereabout because the game pad input is a little erratic.

It is erratic because it is cheap and cheerful but nevertheless extremely efficient and useful. Whatever hardware supplies the readings, be it on your PC, on a PCI card or on a USB based dongle, it operates essentially the same way. The stick moves a potentiometer, a pot, a variable resistor. This resistor controls how fast a capacitor discharges, the reading is taken by timing when the voltage across it falls from +5v to +0v, 1 or 0 in computer speak. The time interval will be short if the resistance is low and high if the resistance is high. This is what supplies the "analog" nature of the input.

It is a fast, repetitive process. The timing of the pulse to charge the capacitor, the temperature, mechanical variance and the graininess of the graphite on the pot all contribute to the erratic nature of the raw data. You can calm it down with algorithms.

The wiring of these devices is pretty simple no matter how complex the physical device actually looks. If you want some fun you can hack any old cheap device and get a myriad of robotic or measurement devices connected to Liberty BASIC, provided you measure resistance, (100k), or button closure.



Device capabilities

To see what your device can do simply plug it in and invoke the Windows control applet. Alyce shows us how.

```
'To Display the Joystick Settings
run "rundll32.exe shell32.dll Control_RunDLL joy.cpl"
```

Native command set

The native command set allows up to three axis of movement and two buttons to be read with the readjoystick (n) command. Once called, reference the dedicated system variables holding the x,y,z (z if it exists) and button conditions.

```
readjoystick 1
readjoystick 2
print Joy1x, Joy1y, Joy1z, Joy1button1, Joy1button2
print Joy2x, Joy2y, Joy2z, Joy2button1, Joy2button2
```

Enhanced API options

If we want to go further than that then we need to use API calls. There are two available. joyGetPos and joyGetPosEx. The first gets pretty much what Liberty BASIC achieves but does include more button info. The second reads up to six axis of movement and as many buttons as the device supports. Use which one suits your purpose best.

This code shows all three reading options in action.

```
    'define a struct for joyGetPos read, standard two axis joystick
    'for more info [[http://msdn.microsoft.com/en-
us/library/windows/desktop/dd757110(v=vs.85).aspx]]

    struct joy0,_
              x as long ,_ 'x axis 0 - 65535 with 32767 as centre
              y as long ,_ 'y axis
              z as long ,_ 'z axis if it exists
        buttons as long    'bits set by button press


'define a struct for joyGetPosEx read, gampads, car simulators and fli
ght sim yokes
    'for more info [[http://msdn.microsoft.com/en-
us/library/windows/desktop/dd757112(v=vs.85).aspx]]

    struct joyex0,_
                size as long,_ 'size of struct
               flags as long,_ 'what to check
                   x as long,_ 'x axis
                   y as long,_ 'y axis
                   z as long,_ 'z axis
                   r as long,_ 'r axis
                   u as long,_ 'u axis if it exists
                   v as long,_ 'v axis if it exists
             buttons as long,_ 'bits set by button press
        buttonNumber as long,_ ' number of buttons pressed
                 pov as long,_ 'point of view value
           reserved1 as long,_
           reserved2 as long


'store the size of the struct and set the flag value for the info we w
ant
    joyex0.size.struct=len(joyex0.struct)
    joyex0.flags.struct=1 or 2 or 4 or 8 or 128
'ie xyzr axis and buttons info returned

    'joysticks can be numbered 0-15
    'replicate the calls and structs for additional joysticks
    joy=0
    timer 250,[readit]
    wait
    [readit]
    cls

    'native read
    'restricted to three axis and buttons 1 and 2
    readjoystick 1 'actually joy 0
    print "joy0 X ";Joy1x,"Y ";Joy1y,"Z ";Joy1z,,
Joy1button1+Joy1button2

    'standard api read
    'restricted to three axis but can read all buttons
    calldll #winmm, "joyGetPos", joy as long, joy0 as struct ,
 result as long
    print "joy0 X ";joy0.x.struct,
    print "Y ";joy0.y.struct,
    print "Z ";joy0.z.struct,,
    print joy0.buttons.struct

    'extended info api read
    'up to six axis if they exist and all buttons
    calldll #winmm, "joyGetPosEx", joy as long, joyex0 as struct ,
 result as long
    print "joy0X ";joyex0.x.struct,
    print "Y ";joyex0.y.struct,
    print "Z ";joyex0.z.struct,
    print "R ";joyex0.r.struct,
    print joyex0.buttons.struct

    wait


'determine if a button is pressed by anding its value against the butt
on value.

'button 1=2^0, button 2=2^1, button 3=2^2, button 4=2^3, button 5=2^4,
 etc.
    'say button 2 and 5 are pressed thats 2+16=18
    print 18 and 2
    print 18 and 16
```

Joystick and Gamepad Input | The port | Simple Joystick | Device capabilities | Native command set | Enhanced API options