Hardware Control with the Serial Port and Propeller.

Richard Baggett, December 24, 2008 r2consult.net

In this article, we will examine the issues related to using the serial port with Liberty Basic. We will automatically scan for available ports, check those ports for our hardware, and then control that hardware.
This will work with any device that can appear as a Windows serial port. This includes real serial ports on the motherboard, real ports on added hardware, USB to serial converters, even Ethernet serial devices (Terminal servers) that allow the serial port to be anywhere in the world!
For our hardware, we will use the Parallax “Propeller” Demo board. It has a USB to serial converter built in, LEDs that we can turn on and off from our Liberty Basic program, and even comes with the USB cable.
My Main reason for using the Propeller, is that, like Liberty Basic, it is deceptively powerful for it's simplicity! You will be up and running in minutes, but the ultimate uses are limited only by your imagination. Liberty Basic users, accustomed to Excellent support, and a dedicated community eager to help, will feel right at home as the Propeller has this, also. The high-level SPIN language for the Propeller is somewhat Basic-like. Liberty Basic programmers will quickly be doing amazing things with it!
Download the most recent Propeller Tool Software from Parallax.com. Hook up your Propeller Demo board to power and USB. The SPIN code below may be copied and pasted into the Propeller Tool's editor. Then just press f11 to compile and download the code to your Propeller Demo board.
Note the lack of End If and other expected block closures.. In SPIN, indenting blocks of code isn't just for looks, it's the way the blocks are defined!
This is NOT Liberty Basic Code! It is SPIN for the Propeller Board

```
{{*************************************Propeller IO for LibertyBasic *
**********************************

       This is intended to run on the Demo Board. It will also run on
a ProtoBoard with a Prop Plug. If you go the ProtoBoard route,
       you will need to wire up your own LEDs if you want to see 'em g
o on and off from the LibertyBasic program.
}}
CON
    _clkmode = xtal1 + pll16x
    _xinfreq = 5_000_000


VAR
Byte    Buffer[32]                                          'My serial buf
fer

OBJ
  Serial : "FullDuplexSerial"

PUB Start  |idx
  Serial.start(31,30,0,9600)                                'Start the ser
```

```
ial port through the programming adapter.
  Dira[16..23]~~
  repeat
    If GetWord
      If Strcomp(@buffer,@MyName) == -1
        Serial.str(@Respond)
        Buffer.byte[0] := 0                             'Clear the buf
fer, Propeller strings are zero terminated.
      If Buffer.Byte[0] == "l"                          ' 'l'ed comman
d?
        Repeat idx from 16 to 23
          If Buffer.Byte[idx-15] == "1"
            Outa[idx] ~~
          Else
            Outa[idx] ~
        Buffer.Byte[0] := 0
    Serial.rxflush

PRI GetWord  |idx, tmp
  idx := 0                  ''This object attempts to receive a retur
n terminated string, or a full buffer.
    repeat                  ''The string is put into Buffer, and the
function returns the length of the string.
      tmp := Serial.rxcheck
      case tmp
        13:                     'Return encountered
          Buffer.byte[idx] := 0
        "0".."z":               'Characters only
          Buffer.Byte[idx++] := tmp
          if idx > 32
            tmp := 13
    while tmp <> 13
   Result := idx

DAT
MyName  Byte  "Propeller?",0
Respond Byte  "Present!",13,0      'Zero terminated strings
```

All our hardware setup is done! Now lets look at the Liberty Basic code!
First, We need a way to determine if a particular COM port is available. Most of the following routine is straight from the Yahoo Liberty Basic group.
By using an API call to open the unknown port, we avoid triggering an error when the port doesn't exist. We just get _INVALID_HANDLE_VALUE. Then it's a simple matter to return true or false for the existence of the port.

```
Function CheckCom(cp)                       'Checks for COM port avail
able. Thanks LB group members!
    lpFileName$ = "COM"; cp                 'Returns true if port can
be opened
    dwCreationDistribution = _OPEN_EXISTING
    hTemplateFile = _NULL
    CallDll #kernel32, "CreateFileA", _     'This won't halt the progr
am if the port doesn't exist.
        lpFileName$ as ptr, _
        dwDesiredAccess as ulong, _
        dwShareMode as ulong, _
        lpSecurityAttributes as ulong, _
        dwCreationDistribution as ulong, _
        dwFlagsAndAttributes as ulong, _
        hTemplateFile as ulong, _
        hFileHandle as ulong

    CallDll #kernel32,"CloseHandle",_       'Close the port, so we don
't get an error later
        hFileHandle as ulong,_
        ret as long
    If hFileHandle = _INVALID_HANDLE_VALUE Then
        CheckCom = 0
    Else
        CheckCom = 1
    End If
End Function
```

Now that we've found an active port, how do we know if it's the one where our hardware is attached?
We design a protocol. Protocol is simply the process of presenting information in an agreed-upon way. Since we will be using a serial port, we must have some way to find our place in the stream of data. For this, we will use a special character. Since Liberty Basic automatically adds a after each print statement, (With notable exceptions, like the trailing ';' .) we will use it as our 'end of data' character.
Our protocol will work like this:

- No data is longer than 32 characters
- All data ends with a (Chr$(13))
- Unrecognized data will be ignored.
- Data packet 'Propeller?' will be understood as a query to our hardware. 'Are you there?'
- Hardware will return 'Present!' in response. (Just like attendance at school!)
- Data packet nine characters long beginning with 'l' is the command for the LEDs.
- We could just add more stuff forever....almost..

As you can see, we can make our protocol carry any kind of data, as long as our Liberty Basic program and our

hardware agree on what it represents. That's all there is to it.
One important serial port quirk is that the data printed to the port is not sent until the Liberty Basic program pauses at a Wait statement, or an Input. Scan won't work. This is the reason behind the unorthodox use of Timer in this function. We print Challenge$, and then immediately invoke the timer and hit a wait statement. This allows our data to be sent, as well as providing time for the reply to arrive. When the timer fires, we disable the timer, and continue. The length of the timer may be adjusted according to the speed of your computer, and any delays caused by your serial connection.
So, Here it is:

```
Function DevCheck(cp,Challenge$,Expect$)    'Checks for device present
 by sending Challenge$, then
        Port$ = "COM"; cp; ":"; Baud; ",n,8,1,ds0,cs0,rs"  'examining
the reply for Expect$
        Com = 1024
        Reply$ = ""
        Open Port$ for random as #io
        oncomerror [ErrNotify]
        #io, chr$(13)
        #io, Challenge$             'An unusual way to use the Timer.
It will be available afterward.
        Timer 80, [IoWait]          'Things sent to the COM port will
not actually be sent out
        Wait                        'until we hit a wait statement! Th
en the wait must be long
        [IoWait]                    'enough to allow a reply. If funny
 things happen, try more time.
        Timer 0
        NumBytes = lof(#io)
        If NumBytes > 0 Then                      'We have a possible
reply..
            Reply$ = input$(#io, lof(#io))
            If trim$(Reply$) = Expect$ Then       'Is it the response
we expect?
                DevCheck = 1                      'Trim$ removes the t
erminating zero
            Else                                  'from the Propeller
string
                DevCheck = 0
            End If
        End If
        Close #io                'We should leave the computer the way
 we found it.
End Function
```

Using a Timer inside a function like this is risky. If any other activity causes our program to branch or call a sub or

function while we are paused at our Wait statement, our program will crash! If this were a program for general release, It would be best to do our search before we open any windows at all.
In our demo program, we will simply cover the entire window (Including the title bar with it's pesky buttons.) with a buttonless dialog. For simplicity's sake, our demo won't do anything special to ensure that the original window hasn't been moved or resized, so just don't do it. Of course, if we make a version for non-programmers to use, we should do a much better job.
So here's the rest of the demo.

```
'*************************** Propeller IO *********************
'Demonstrating functions and techniques for successful serial
'port projects

'****************** Window Setup**********************
    NOMAINWIN
    WindowWidth = 250 : WindowHeight = 150
    UpperLeftX = INT((DisplayWidth-WindowWidth)/2)
    UpperLeftY = INT((DisplayHeight-WindowHeight)/2)

    StaticText  #main.st, "Using COM port number:", 10, 60, 150, 20
    TextBox     #main.CP, 165, 55, 55, 25
    CheckBox    #main.P16, "P16", LED, LED,  5, 5, 50, 16
    CheckBox    #main.P17, "P17", LED, LED,  65, 5, 50, 16
    CheckBox    #main.P18, "P18", LED, LED,  125, 5, 50, 16
    CheckBox    #main.P19, "P19", LED, LED,  185, 5, 50, 16
    CheckBox    #main.P20, "P20", LED, LED,  5, 30, 50, 16
    CheckBox    #main.P21, "P21", LED, LED,  65, 30, 50, 16
    CheckBox    #main.P22, "P22", LED, LED,  125, 30, 50, 16
    CheckBox    #main.P23, "P23", LED, LED,  185, 30, 50, 16
    Button      #main.rsc  "Re Scan COM ports", [Rescan],ul,  5, 85, 2
30, 24

'****************** Window *********************
    Open "IO with the Propeller" for Window as #main
    #main "trapclose [quit]"
    #main "font ms_sans_serif 10"

'**************** Find Propeller ***************
[Rescan]
WindowWidth = 255 : WindowHeight = 180
statictext  #scn.static1, "scanning for hardware,", 55, 30, 145, 16
statictext  #scn.static2, "Please wait.", 80, 65, 85, 16
Open "scanning" for Dialog_popup as #scn        'We'll cover the entir
e window with this
                                                'popup to prevent any
events during the scan.
```

```
    If ComOpen = 1 Then
        Close #io
        ComOpen = 0
    End If
    Global Baud, PortNumber, ComOpen
    Baud = 9600
    For a = 1 to 30                     'USB serial adapters can have
some crazy numbers!
        If CheckCom(a) = 1 Then
            If DevCheck(a,"Propeller?","Present!") = 1 Then
                #main.CP, str$(a)
                Port$ = "COM"; a; ":"; Baud; ",n,8,1,ds0,cs0,rs"
                Com = 1024
                Open Port$ for random as #io
                oncomerror [ErrNotify]
                ComOpen = 1
            End If
        End If
    Next
    If ComOpen = 0 Then
        #main.CP, "None"
    End If
    Close#scn
    Wait
[ErrNotify]
    Notice "Unexpected error. Com";Str$(ComPortNumber);" ";ComError$
  'Report error and close
[quit]
    Close #main
    If ComOpen = 1 Then Close #io
    End

Sub LED handle$                             'Handle the checkboxes. No
te that the click does not
    If ComOpen = 1 Then                     'happen until the mouse bu
tton is released.
        Send$ = "l"                         '"l" for led.
        For a =  16 to 23                   'The LED i/o addresses on
the Propeller Demo Board
            CbHandl$ = "#main.P"; a
            #CbHandl$, "value? V$"
            If V$ = "set" Then
                Send$ = Send$ + "1"         'We want 'checked' leds ON
            Else
                Send$ = Send$ + "0"         'We want 'unchecked' leds
OFF
```

```
            End If
        Next
    #io, Send$                              'Just send the string, tha
t's all there is to it!
    End If
End Sub
```

The only thing specific to hardware here is the LED subroutine. It builds our 'l' command by testing all of the checkboxes, adding a '1' for checked or a '0' for unchecked. Then it sends it. Since this sub is called from the 'wait' state by checking or clearing a checkbox, we immediately return to the 'wait' state afterward and our data gets sent right away

Have some fun turning the LEDs on and off. Then take a careful look at the hardware.

Did you notice all the other stuff on your Propeller Demo board? Yes, that is a VGA monitor port on there! And actual mouse and keyboard ports! What about the microphone? And that video or antenna looking RCA socket.. It'll do video OR broadcast tv channels! Yes, that other doohickie is a jack for headphones! Even some I/O left over to use with that solderless breadboard!

Yes, we will have lots of fun with all of these in future articles! Be sure to check out the Propeller forum and Propeller Object Exchange at Parallax.com. The Propeller data sheet has an excellent SPIN programming tutorial. You will quickly see that there is nothing you can imagine that Liberty Basic, and the Propeller, can't do!