Wirebridge Example Code for PowerBasic

There are currently 3 models of Wirebridge available.

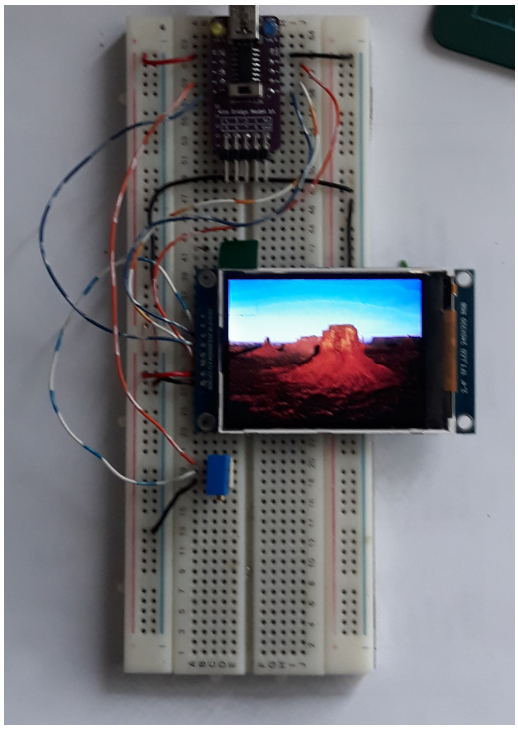
Model 1A with 8 I/O pins, 3 channels of PWM output, SPI and I2C connectivity. This is 3.3Volt or 5V switched.

Model 1B with 8 I/O pins, 3 channels of PWM output, SPI and I2C connectivity. This is 5V only.

Model 2 with 6 Output pins, 2 input Pins and 2 PWM channels. This has open drain mosfet outputs which can tolerate upto 24v. The inputs can also handle high voltages if needed.

Using the SPI interface with PowerBasic

SPI (Serial Peripheral Interface) is used primarily for short-distance wired communication between integrated circuits. And can be found on various items like temperature measurement and displays like this one.

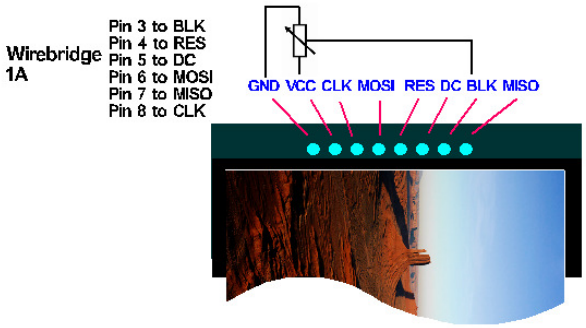


The particular screen used here is 3.3V. In this case to avoid the use of buck converters and level shifters the Wirebridge 1A with the 5v<>3v switch is the bridge of choice.
All that's needed is a few wires and a small pot to set the brightness.

You will need the Chipset info to make it work. This one uses the 9431 chipset and suitable codes are included in the file "pb_ili9431.inc" enclosed which contains equates for most of the commands available, this demo program just uses a subset of them.

The ability to load, rotate / resize images to fit the display is provided by the DeViL project which works well with PB and an include file and DLL are included with the source to enable this. More info here https://openil.sourceforge.net/

Wiring Diagram
The Variable resistor is 5K as in the LCD project the value is not cast in stone and 1k-10k should be fine.



At the start are a few useful equates – basically to remove magic numbers

```
%MOSI = 10  ' WB PCB pin 6
%MISO =  9  ' WB PCB pin 7
%CLK  =  8  ' WB PCB pin 8
%DC   = 11  ' WB PCB pin 5
%RES  = 12  ' WB PCB pin 4
%BLK  = 13  ' WB PCB pin 3  Used to turm on and off the backlight
%BLK_Toggle = 1  'used for backlight toggle (its a flag)
```

These additional includes are used.

#INCLUDE ONCE "pb_ili9431.inc" 'Contains the Equates for the TFT chipset
#INCLUDE ONCE "wirebridge.inc" 'Contains the Equates, Types & Functions for the WireBridge
#INCLUDE ONCE "de1.inc" 'Contains the Equates, types and Functions for the DeVil DLL

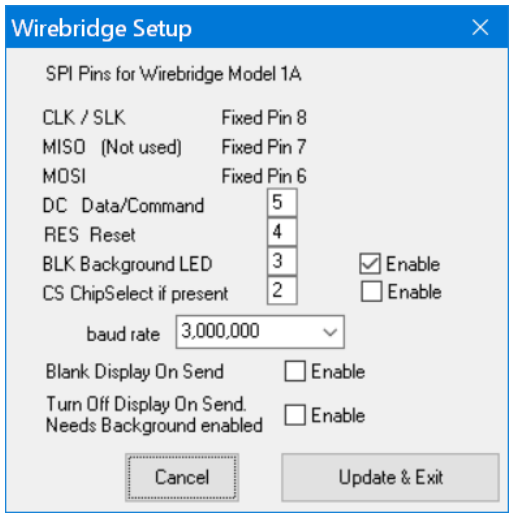
This starts off by enumerating the Wirebridge's but this time restricting to just the Model 1A for the 3.3V possibility. To use a 1B then additional circuitry may be needed unless a 5V display is used.

The Subroutine EnumWB Is basically the same as the LCD one but only the 1A is accepted.

```
        wb_outcome = WB_EnumerateDevicesNext( WB_DEVENUMERATION, devinfo, Wberr)
          IF  wb_outcome =0 THEN    ' got one!
            wb_Path=wb_dev.@USB_PATH     'make a copy of the path get Device handle
            model = wb_dev.devinfo.productname
            Gdevinfo = wb_dev.devinfo
            IF INSTR (model,"1B") THEN
'               T2 = "Wirebridge Model 1B found"    '     Commented OUT
'               EXIT LOOP
            ELSEIF INSTR (model,"1A") THEN
                T2 = "Wirebridge Model 1A found"
                EXIT LOOP
            ELSEIF INSTR (model,"2") THEN
'               T2 = "Wirebridge Model 2 found"    'Commented out
'               EXIT LOOP
            ELSE    'nout attached
                myerror="Device not detected"
                EXIT SUB
            END IF
          ELSE
```

The big change is the Initialise subroutine WB_ILISetup(hdlg AS LONG) used to setup the TFT screen and its ILI9431 chipset. It's worth downloading the ILI9431 datasheet for information.

This program uses an INI file to store various settings, these are in the utility menu under "Pin settings." The Baud rate selection is only available after the Wirebridge is initialised as it is read from the Wirebridge.



The function to find and use the available bauds is.

Word_Result = WB_GetBauds(gdevinfo,index)
Call with an Index of 0, Index is a WORD.
This returns Index as the number of entries for the list of baud rates, call with again with Index from 1 to Index to retrieve all the values. The index has a maximum value of 256 currently. This program just takes the first 30 to present in a combo box. Index 1 is the fastest available and the speeds decrease as the index increases.

```
 index = 0
   Word_Result =  WB_GetBauds(gdevinfo,index )'gdevinfo from the WB initialise
   IF Word_result > 30  THEN ' grab the first 30
 FOR I=1 TO 30
   index=I
   Word_Result =  WB_GetBauds(gdevinfo,index )
   COMBOBOX ADD hdlg, %IDC_COMBOBAUD,TRIM$(USING$("###,###,###" ,word_result) )
ELSE ' something odd, probably not initialised so don't bother
NEXT
END IF
    getsetting( "WB_Setup","SPI_Baud",defs)' retrieve previous setting
    COMBOBOX FIND hDlg, %IDC_COMBOBAUD, 1, (TRIM$(defs)) TO I 'locate its index

  IF I=0 THEN I=12  'not found (0) then use index 12 to start
    COMBOBOX SELECT hdlg, %IDC_COMBOBAUD,I    ' display the I value for the baud
```

The WB_ILISetup uses the following functions.
wb_outcome = WB_SetPinDirection (gWB_Devhandle,%DC,%PinDir.Dir_out,WBERR)
wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high , WBERR)
wb_outcome =WB_SPI_Init(gWB_Devhandle,%Mode,BaudX,WBERR)
wb_outcome = WB_SPI_Transfer(gWB_Devhandle, zCount,Sbp,rbp ,WBERR)

WB_SetPinDirection Sets the Pin to be an output or input
WB_WritePinDigital Sets the Output Pin High / Low or floating
WB_SPI_Init Sets up the SPI interface – Mode and baud rate.
Consult any documentation for the SPI peripheral for setting the baud rate and Mode
For the TFT used here the maximum baud that can be used is 6,000,000 and Mode 0 is required.
WB_SPI_Transfer does the actual work of sending and receiving bytes. The operation depends on the setting of the DC pin which selects if Data or a Command is in the buffer.

The Subroutine to Initialise the TFT calls Init9431 to load the initial setup for theTFT into an array of Command/Data pairs. Other things could be added / changed here.

```
SUB WB_ILISetup(hdlg AS LONG)
  LOCAL I,J, SPI_MODE  AS LONG
  LOCAL wb_outcome, zcount AS BYTE
  LOCAL WbErr AS WB_ERROR PTR
  LOCAL WB_ERR AS WB_ERROR
  LOCAL myerror,defs,t1,T2 AS STRING
  LOCAL rexbuffer AS STRING
  LOCAL sbp,rbp AS STRING PTR
  DIM Dset(1,1) AS STRING ' this will be redimensioned. Used directlu as send buffer
  LOCAL BaudX AS DWORD

  getsetting( "WB_Setup","SPI_Baud",defs) 'retrieve the set value
    FOR I=1 TO LEN(defs)    'remove formatting!
      T1=MID$(defs,I,1)    ' to leave a number
        IF T1=" " OR T1="," THEN   T1=""
        t2=t2+T1
    NEXT
      BaudX=VAL(t2)
      IF BaudX < 10  THEN   'no baud saved, first run?
          MSGBOX "Please Set the Baud rate in the"+$CRLF+ _
                 "    Utility Menu - Setup"+$CRLF+ _
                 "  Then run the setup again",%MB_ICONWARNING, "Demo"
          EXIT SUB   'Bail out now and set the rate
       END IF
    'as setup called and we are here the WB is found and a baud rate is available
    IF gwb_devhandle = 0 THEN EXIT SUB  'Check the handle is valid bail on no handle

    WbErr = VARPTR (WB_ERR)   'Setup the error reporting
    wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%WB_Model1A.Pin1 ,%PinDir.Dir_out,WBERR)
    myerror= WB_ERRORTYPE(@wberr.WB_ERRORTYPE)'  Convert error to text
    IF VAL(myerror)<>0 THEN
         MSGBOX Myerror,,"DEMO"
         EXIT SUB 'bail on error
    END IF
    'if it did not error we can use BYVAL %NULL instead of wberr makes it more readable
```

```
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%WB_Model1A.Pin2, %PinDir.Dir_out,BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%BLK,   %PinDir.Dir_out, BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%MOSI,  %PinDir.Dir_out, BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%MISO,  %PinDir.Dir_in, BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%CLK,   %PinDir.Dir_out, BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%DC,    %PinDir.Dir_out, BYVAL %NULL )
     wb_outcome = WB_SetPinDirection ( gWB_Devhandle,%RES,   %PinDir.Dir_out, BYVAL %NULL )
     'thats all the pins setup   now set the pin outputs
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high ,BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%WB_Model1A.Pin1,%DP_Value.high ,BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%WB_Model1A.Pin2,%DP_Value.low ,BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%BLK,%DP_Value.high , BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%MOSI,%DP_Value.low ,BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%MISO,%DP_Value.low,BYVAL %NULL)
     wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%CLK,%DP_Value.low ,BYVAL %NULL)
    'RESET ---- FLIP THE RESET (%RES) LINE  down then up
    wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%res,%DP_Value.low , BYVAL %NULL)  'reset
    wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%res,%DP_Value.high , BYVAL %NULL)
    SLEEP 50  ' let the init run for 50 ms
    'should not really be needed as the time between USB sends ought to be enough
'       MODE0 = 0 MODE1 = 1 MODE2 = 2  MODE3 = 3
        SPI_MODE=0          'mode    0 for this tft
      wb_outcome =WB_SPI_Init(gWB_Devhandle,SPI_MODE,BaudX, BYVAL %NULL)

      RexBuffer=STRING$(50," ") 'Recieve buffer
      rbp=VARPTR(rexbuffer)
      CALL Init9431( Dset()) ' Get the command list
      I=VAL(dset(0,0)) '  this is the command count

 FOR J=1 TO I
    sbp=STRPTR(dset(J,0)) ' Get address of the command buffer
    Zcount=LEN(dset(J,0))  'length of command buffer - one for this device as it happens
 'command first
    wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.low, BYVAL %NULL)  'command
    wb_outcome = WB_SPI_Transfer(gWB_Devhandle, zCount,Sbp,rbp ,BYVAL %NULL)
    wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high, BYVAL %NULL)

    'now data
      sbp=STRPTR(dset(J,1))
      Zcount=LEN(dset(J,1))
     IF zcount>0 THEN ' The command has both a command byte and data following
         wb_outcome = WB_SPI_Transfer(gWB_Devhandle, zCount,Sbp,rbp ,BYVAL %NULL)
     END IF
     ' do not send on zero bytes!!!
 NEXT
    'finally makesure the DC (data command switch) is high
    wb_outcome = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high, BYVAL %NULL)
END SUB
```

Now the interface is ready, load an image via the file menu and send it. There is an option to toggle the back light as well as some settings in the utility menu > Pin Settings. The routines to load an image are handled by the DeVil Dll (enclosed). This can open and resize a number of image types as well as rotate etc. The final image is added to a Graphic window then the SPI send is invoked.

The bitmap is retrieved, converted to the 2 byte pixel type used and placed in a buffer for sending. Here there is a limit on the size of a single send using the USB interface so the data is sent in 60 byte chunks. A variable is used (BS) so the size can be varied for testing. The default is 60 bytes

```
SUB send2TFT(hdlg AS LONG)
    LOCAL bmpbuff,tmp,lin ,dibhead AS STRING
    LOCAL M AS STRING PTR
    LOCAL Pixptr AS RGBA PTR
    LOCAL Pix AS RGBA
    LOCAL byte_result,zcount AS BYTE
    LOCAL sx,sy,K,I,j,X,IJ ,bs,landscape,xbp AS LONG  '   ,SX=320 SY=240
    LOCAL SB, RB AS STRING
    LOCAL RBp, sbp AS STRING PTR
    BS=60    'batch size of bytes  60 default  can be played with
    landscape = 0  ' Set to Portrait initially
      GRAPHIC GET BITS TO BMPBuff
      DIBHEAD = LEFT$(bmpbuff,8)
      tmp = RIGHT$(bmpbuff,-8)
      SX= CVL (LEFT$(DibHead,4))  'Screen X in pixels
      SY= CVL (MID$(DibHead,5,4)) ' Screen Y (height) pixels
      BMPBuff=tmp 'Rawdata in BGR format 4bytes /pixel
```

```
  IF SX>Sy THEN 'its landscape so use the Lbuffer. This is a landscape version as a global
                ` string created in the file input stage
   landscape=1
   BMPbuff   = Lbuffer
   DIBHEAD = LEFT$(bmpbuff,8)
   tmp = RIGHT$(bmpbuff,-8)
   SX= CVL (LEFT$(DibHead,4))  'Screen X in pixels
   SY= CVL (MID$(DibHead,5,4)) ' Screen Y (height) pixels
   BMPBuff=tmp 'Rawdata in BGR format 4bytes /pixel
  END IF

     'its formated as 320x 240y screen is actually 240/320 and fed as 240
     ' convert from BGR to RGB 16 first uses BBGRA or BRGBA
M=STRPTR(bmpbuff)  'NOTE this is BGR!!!!!!
lin=""
 FOR I=1 TO LEN(bmpBuff) STEP 4
     Pixptr=M
     pix=@pixptr
     M=M+4
     IF landscape=0 THEN
     Lin = lin + MKWRD$(BBGRA(PIX))
     ELSE
     Lin = lin + MKWRD$(BRGBA(PIX))
     END IF
 NEXT      'returns with buffer (lin) made of 2byte pixels for the TFT
 IF landscape=1 THEN  ` set the correct progress bar
    J=%IDC_PROGRESSBAR1
    CONTROL NORMALIZE HDlg, %IDC_PROGRESSBAR1
    lin=RIGHT$(lin,-1)+CHR$(0)
 ELSE
    J=%IDC_PROGRESSBAR2
    CONTROL NORMALIZE HDlg, %IDC_PROGRESSBAR2
    lin=RIGHT$(lin,-1)+CHR$(0)
 END IF
 x=LEN(lin)' this is the pixel buffer
   RB="    "        'setup the recieve buffer
   rbp=STRPTR(rb)   'and pointer
   sb=CHR$(&H2c)+CHR$(0)  'now the command to send data
   sbp=STRPTR(sb)          ' and its pointer - its actually one byte
   zcount = 1
   'DC is set for Command
   Byte_result = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.low , BYVAL %NULL)
   Byte_result = WB_SPI_Transfer(gWB_Devhandle, zcount,Sbp,rbp ,BYVAL %NULL) ' writeCmd(&H2C)
   Byte_result = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high, BYVAL %NULL)
   PROGRESSBAR SET RANGE hDlg, J, 0, 100
   PROGRESSBAR SET STEP hDlg, J, 1
       sbp=STRPTR(lin)   'Pointer to Start of buffer
       Xbp=sbp
FOR I=0 TO x STEP bs    'Step along the buffer in BS (60) byte chunks
       Byte_result = WB_SPI_Transfer(gWB_Devhandle, bs,Xbp,rbp ,BYVAL %NULL)  'Send
       PROGRESSBAR SET POS hDlg, J, INT(100*(I/X))
       Xbp=Xbp+60        'add current position (I)
       DIALOG DOEVENTS 0     'required to keep progress bar running
NEXT
   sb=CHR$(&H00) 'send the finished command!
   sbp=STRPTR(sb)
   Byte_result = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.low , BYVAL %NULL)
   Byte_result = WB_SPI_Transfer(gWB_Devhandle, 1,Sbp,rbp ,BYVAL %NULL) ' writeCmd(&H2C)
   Byte_result = WB_WritePinDigital(gWB_Devhandle ,%DC,%DP_Value.high , BYVAL %NULL)

END SUB
```

Hopefully this makes some kind of sense. SPI is not that esoteric but a knowledge of what the peripheral being driven requires is needed. Unfortunately unlike RPI or Arduno and similar there are no readymade Libraries for these things in PowerBasic.