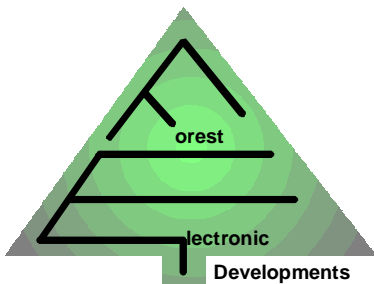


# ***Forest Electronic Developments WIZ-C Professional USB Made Easy CDC Library***

**THE FED C Compiler is intended for all serious programmers of the PIC who would like the convenience of a high level language together with the speed of assembler. With our C Compiler you no longer have to worry about ROM and RAM paging, you can call to a depth limited only by RAM not by the 8 level call stack, use 16 and 32 bit arithmetic types for full precision, and use any of our standard library routines for general purpose data handling and interfacing.**

**This manual describes the USB library for use with WIZ-C professional.**



**Forest Electronic Developments**  
12 Buldowne Walk  
Sway  
LYMINGTON  
Hampshire  
SO41 6DU

**[info@fored.co.uk](mailto:info@fored.co.uk)**

Or see the **Forest Electronic Developments** home page on the World Wide Web at the following URL:

**<http://www.foresd.co.uk>**

## ***Introduction***

Writing and using USB drivers for PIC based projects can be very hard work. Microchip have released their CDC code which has been converted for the FED C Compiler series, but even this requires understanding of the detailed operation of the USB protocols and their implementation on the PIC.

FED have created this library set to allow users to add USB serial emulation to the PIC as easily as any other WIZ-C element. Once added then an occurrence is created to call C routines as soon as bytes are received from the USB port. Users have to undertake no additional configuration other than the standard element checks.

The library set provides additional capability to simulate a PC terminal emulator operating over a USB port. Standard configuration files simulate the reset and configuration of the interface to take it into the configured state transparently to the user.

The USB Forum defines a standard for serial communications devices called the Communications Device Class (CDC). The Microchip library on which the FED development kits are based is called the CDC library and we refer to CDC throughout this manual.

## ***Installing the library***

This library requires version 11.04 or later of FED's Integrated Development Environments (FED C Compiler or WIZ-C). Install the updated IDE FIRST and then the library from the USB CD afterwards. Choose the Setup Library option from the USB CD. If you have installed the IDE in a location which is not the standard one (c:\program files\fed...) then you will need to enter the correct location during the setup process.

Following installation the IDE will now have a new USB terminal emulator device. The library will be installed in the USB sub-directory of the libs directory. The example projects will be installed in the Projects\USB directory below the main PIXIE directory. Finally there will be a new element called USBCDC on the USB tab for 18F devices.

## ***Installing the PC driver***

By default the library uses the Microchip Vendor and Product ID's. The serial communications interface does not need a driver on the Windows 2000 and XP operating systems – however an information file is required to inform the PC of the type of interface to be used.

The FED development board is supplied with an application (The LoopBack example). When it is inserted it will automatically be recognised as a serial interface. For the Microchip USB FSDemo board you will need to program a serial interface hex file to the board as it ships with the full speed demo installed.

Insert the USB CD and when the opening menu comes up close the menu application. Plug the USB cable into the PC. The “Add New Hardware Wizard” will start, click Next and then select the option “Search for a new driver for my hardware (Recommended)”. On the next screen select the CD-ROM option and then Windows will identify the appropriate driver for your version of Windows and install it.

### ***Licence issues***

The library is based on modified Microchip CDC code. The licence agreement for this code is included here in Section 8. You must agree to the terms of this licence before using this library.

In essence you are allowed to modify the code, use it in your own applications and allow 3<sup>rd</sup> parties to use it and program it to devices provided that derived code is only ever used for PIC devices and that any 3<sup>rd</sup> parties also agree to the full terms of the licence.


**Contents**

**1 The USB interface ..... 5**

**2 Connecting a PIC device for USB CDC functions ..... 5**

**3 Hardware considerations ..... 5**

**4 Using the Microchip USB boot loader with the Microchip or FED development boards..... 6**

**5 WIZ- C Element, USB CDC interface  ..... 7**

5.1 External I/O's ..... 7

5.2 Parameters..... 7

5.3 Occurrences..... 8

5.4 Public Calls & Variables..... 8

5.5 Hooked Elements ..... 8

**6 USB Terminal emulation ..... 9**

**7 Examples ..... 10**

7.1 Example 1 – Simple Loopback ..... 10

7.2 Example 2 – Simple Serial port ..... 11

7.3 Example 3 Non WIZ-C users..... 13

**8 Reference ..... 17**

8.1 Non blocking code ..... 17

8.2 Throughput..... 17

8.3 Library code status..... 17

8.4 Functions..... 19

**unsigned char USBAddTx(unsigned char x);..... usbcdc.h**  
**..... 20**

**unsigned int USBGetRx(); ..... usbcdc.h**  
**..... 20**

**unsigned int USBRxSize(); ..... usbcdc.h**  
**..... 21**

**unsigned char USBCanTx(); ..... usbcdc.h**  
**..... 21**

**9 USB Development Board ..... 22**

## 1 The USB interface

The USB interface is now well known. There are 3 speeds available for the USB 2.0 interface, low speed, full speed and High speed. High speed is the 480Mbps interface which most people usually associate with USB 2.0 devices. Low speed is 1Mbps originally for cheaper hardware such as the PIC 16C765. Full Speed operates at 12Mbps.

The 18Fxxx series of devices which are the target of this library offer low and full speed functionality, the serial emulation included here uses the full speed mode.

### 1.1 Connecting a PIC device for USB CDC functions

To connect a PIC to the USB interface only 4 connections need to be made :

PIC Pin	Connection	USB Socket - pin
	Vcc (optional)	1
	Ground	4
16	D+	3
15	D-	2

The Microchip library on which the WIZ-C version is based has 2 LED's to display status. These are optionally connected to spare port pins and to display the status of the interface. They are exceptionally useful to confirm correct operation during development and are further described below. However on production devices they may be removed to reduce power especially in suspended mode.

### 1.2 Hardware considerations

This library requires operation of the PIC at 48MHz with an external crystal of 20MHz – it is possible to use other rates if the configuration bits are set appropriately, but the standard is 20MHz. It is strongly advised that the PIC is mounted on a ground plane for stability – our test circuits worked very well on the 28 and 40 pin DHMicro development boards which are available from [www.fored.co.uk](http://www.fored.co.uk).

Section 9 describes the USB kit provided by FED on which our examples run. We recommend reading the USB section of the data sheet to review further use of the device on the USB bus.

## 2 Using the Microchip USB boot loader with the Microchip or FED development boards.

For development purposes the microchip USB boot loader is very convenient. It is installed from the CD as described above.

Both the Microchip USB demonstration board and the FED USB development board are supplied with the USB bootloader code pre-installed. In both cases Switch 1 and Switch 2 (SW1 and SW2) are used to control operation of the board between the bootloader and user code.

SW1 is the reset switch. Press SW2 then SW1 and then release SW1 followed by SW2 to run the bootloader code. The Microchip boot loader can then be used.

Press and release SW1 without touching SW2 to run the normal code.

So – to install any of the examples supplied by FED – run the bootloader code. Load the example code using the Microchip boot loader. Program the board. Now press SW1 for a couple of seconds and then release it to run the example code.

**IMPORTANT NOTE.** The FED board uses a 18F2550 device and the USB LED's are connected to pins RA3 and RA4 whereas the Microchip bootloader is configured for LED's on RD0 and RD1. The bootloader will NOT flash the LED's when it is running although the examples will do so. Nevertheless the boot loader will work fine !



### 3 WIZ- C Element, USB CDC interface

The WIZ-C USB CDC element considerably eases initialisation, however users can set up all the parameters manually if required. Note that there is an STI file included automatically in all projects which use this element – this automatically sets up the USB interface by simulating SIE exchanges to take it into the configured state. Simulated configuration is complete by 50mS.

The icon for the library in WIZ-C is on the USB tab.

Set the processor frequency to 48000000 to use this element.

#### 3.1 External I/O's

Name	Type	Description
USBLED1,USBLED2	Output	These are the optional two active high connections for the status LEDs of the USB. They should be connected by 300R resistors to the anodes of two LED's if the status LED's are required.

#### 3.2 Parameters

<b>Use Boot Loader</b>	Check this box if the project is to use a bootloader in memory space 0 to 800 hex. The obvious example here is the Microchip USB bootloader.
<b>Flash USB LEDs</b>	Even if the LED's are connected this parameter allows flashing to be turned off and on. If turned off then the USB code to flash the LED's is not compiled.
<b>Define Standard Config</b>	When selected this automatically includes the code to set the configuration fuses for the FED USB development board, or for the Microchip USB development board. Basically the fuse set is for a 20MHz crystal oscillator, external reset enabled and PLL set for 48MHz internal operation. The bootloader protection fuse is set in accordance with the use of the bootloader.
<b>Vendor ID, Product ID</b>	These two values are the Vendor ID and Product ID used to identify the device to the operating system of the USB host. The default values are 1240 and 10 respectively (decimal) which are the Microchip standard values. Production units will need a unique code – see the section on USB ID's.

### ***3.3 Occurrences***

<b>USBByteRx</b>	This occurrence will be triggered whenever there are bytes remaining in the USB receive buffer. The bytes can then be read using the USBGetRx function.
<b>SetLine</b>	This occurrence is triggered whenever the host changes the line parameters (the bit rate, etc). The new values can be read from the line_coding structure (see below).
<b>SetControl</b>	This occurrence is triggered whenever the host commands the interface to set the control lines (the bit rate, etc). The new values can be read from the control_signal_bitmap structure (see below).

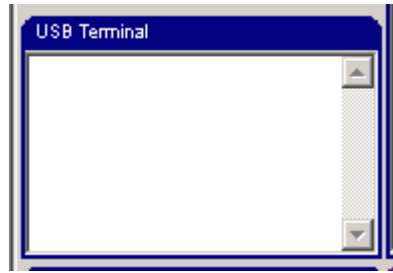
### ***3.4 Public Calls & Variables***

**All of the public calls and variables are described in the remainder of the document.**

### ***3.5 Hooked Elements***

**There are no hooked elements for this element**

## 4 USB Terminal emulation



The USB terminal emulation is a standard external device which emulates a bulk transfer to the USB bus – it correctly reads the size and buffer addresses from the PIC Buffer Descriptor Table. It is added by right clicking the debugging window and using the “Add New External Device” menu option.

The emulation works on a 1mS cycle, every 1mS it will send any characters entered since the last transmission, and will receive any data flagged as available by the end point. USB bus delays are ignored so data will appear on the terminal all at once. Therefore it will appear as “bursty” in transmitting and receiving data. This all makes it a pretty rough emulation but it is certainly good enough to fully check out USB based applications.

It has two parameters :

### 4.1.. Send CR as CR-LF

This may be set to Yes or No, when set a carriage return (Enter key) character will be translated and sent as two characters (ASCII codes 10 and 13).

### 4.1.. EndPoint

This selects the End Point to be used for the terminal – the normal value is 3 which is the value used for these library elements, however it can be set to any value from 3 to 15.

## 5 Examples

All of the examples are designed to run on the FED USB development board described in Appendix A. This uses a 18F2550 device on one of our 28 pin development boards with a USB socket attached. However they will also all run on the Microchip USB demonstration board which has a 18F4550 device – in this case they will run without modification except that the LED's will not flash, the LED connections need to be reconnected to PORT D bits 0 and 1 to make the Microchip board operate with LED's.

The examples may be found in the Program Files\FED\PIXIE\Projects\USB directory.

All examples set the processor frequency to 48000000.

### 5.1 Example 1 – Simple Loopback

This project is the one programmed into the PIC for the development board provided by FED. The only element used is the USB CDC element. An occurrence is set up on the USB RxByte, and all bytes received are simply sent straight back over the interface. If the board is plugged in and the drivers installed then a communications package may be used to connect to the board, any bytes sent to the interface will be received back again.

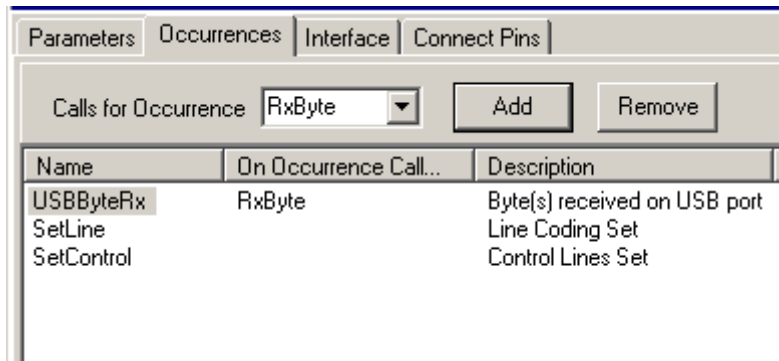
Open the Loopback project which is in the Loopback directory. Note that there is a STI file included in the project – this automatically sets up the USB interface by simulating SIE exchanges to take it into the configured state. This STI file will be included automatically whenever the USB element is included. Configuration is complete by 50mS.

Look at the Element editor parameters :

Use BootLoader	<input checked="" type="checkbox"/>
Flash USB LEDs	<input checked="" type="checkbox"/>
Define Standard Config	<input checked="" type="checkbox"/>
Vendor ID	<input type="text" value="1240"/>
Product ID	<input type="text" value="10"/>

We have chosen to use the boot loader. The LED's are set to flash and we are using the standard configuration for a 20MHz crystal and 48MHz internal operation.

Now look at the occurrences :



We are not interested in the SetLine and SetControl occurrences, however on receiving a byte on the USB interface we call the function RxByte.

Here is RxByte from the LoopBack\_User.c file:

```
char x,y;

void RxByte()
{
    x++;
    y=USBGetRx();
    USBAddTx(y);
}
```

This is a very simple function which simply reads the value from the USB interface and writes it back to the interface.

It is possible to simulate this project. Simply run it, wait for the LED's to start flashing alternately on the Debug window and type characters into the USB terminal and watch them being reflected.

## 5.2 Example 2 – Simple Serial port

The simple serial port demonstrates how easy it is to create a USB serial converter using WIZ-C. In this example we also use the serial port set up occurrence and display the current bit rate set up by the host. The port is set up at a fixed 115200 bps – extra codes would need to be added to allow bit rate and other parameters to be set.

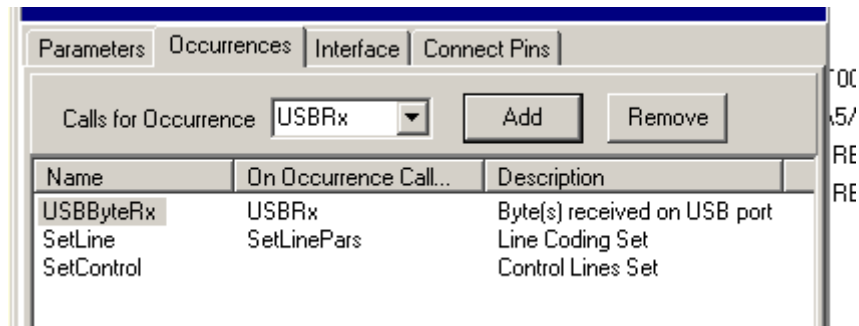
Run the simulation and type characters into the USB terminal on the debugging window and watch them echo out on the USART and vice versa. Type ~ on the USB terminal to see how the bit rate is reflected.

This example will run immediately on the Microchip demonstration board. It will also run immediately on the FED board, however this will required the serial port to be wired up as shown in Appendix A, and also the LED's will not flash – the LED's must be connected to RA3 and RA4 in the element designer and the code recompiled..

This example has two elements, the USB element and the interrupt driven serial port. Note that interrupts are disabled during USB set up and only enabled once the USB interface is configured.

A real serial port application would be better written using a polled USART function in the same way as the USB port is polled. It would also have to react to the bit rate and other port parameters sent by the host.

Any byte received on the USB interface is transmitted on the standard serial port and vice versa. If a '~' character is received on the USB interface then the currently set bit rate is transmitted back on the USB interface. If the host changes the serial parameters then an LED on bit D2 is inverted (if off it will be turned on and vice versa). Note with some hosts the parameters are sent more than once so the LED may only be seen to flash briefly.



Note in this example we have an occurrence on USB received data, but also when the line coding is set.

The USB receive function simply sends data back on the serial port, however if the received byte is a '~' then it prints the currently set bit rate back to the PC, a similar occurrence is set up for the USART :

```
void USBRx()
{
  x=USBGetRx();
  AddTx(x);

  if (x=='~')
  {
    AddUSBString("\n\rBit Rate 0x");
    DoHex(line_coding.dwDTERate._dword);
    AddUSBString("\n\r");
  }
}

void USARTRx()
{
  while(GetRxSize())
  {
    x=WaitRx();
    USBAddTx(x);
  }
}
```

Note that the USARTRx() occurrence is only triggered once, even if many characters are waiting, that is why we use the GetRxSize() function to read all available characters and empty the USART buffer.

The line coding structure is described in detail below.

The occurrence of changing the line parameters causes a LED on D2 to flash (this is LED3 on the Microchip board);

```
void SetLinePars()
{
```



## FED PIC C Library Reference

```
#define MChipBL 1 // Set to 1 or 0 to define use of the Microchip USB bootloader
```

This code defines whether the USB bootloader is in use, leave the line out, or set to 0 to turn it off and locate the code at address 0.

```
#pragma preprocdefine UseUSBLED=1 // Set to 1 or 0 to define use of LED's
```

This defines whether the LED's are in use.

```
// The next lines define the vendor and product id's

#pragma preprocdefine _VendorID=1240 // Vendor ID
#pragma preprocdefine _ProductID=10 // Product ID
```

### Vendor ID and product ID

```
// Define the LED ports and bits

#pragma preprocdefine USBLED1Port=PORTA
#pragma preprocdefine USBLED1Bit=3
#pragma preprocdefine USBLED2Port=PORTA
#pragma preprocdefine USBLED2Bit=4
```

Set to show the port and bit to which the USB LED's should be connected if required.

```
// The next line auto includes the configuration stimulus files

#pragma projectfile "$(FEDPATH)\libs\usb\cdc\sti\configure.sti"
```

This line automatically includes the configure.sti file in the project window if it is not already present. On simulation this file runs up the interface until it is in the configured state.

```
// The next lines define configurations values

#_config _CONFIG1L,0x24
#_config _CONFIG1H,0x0e
#_config _CONFIG2L,0x3f
#_config _CONFIG2H,0x1e
#_config _CONFIG2H+1,0
#_config _CONFIG3H,0x81
#_config _CONFIG4L,0x81
#_config _CONFIG4L+1,0
#_config _CONFIG5L,0x0f
#_config _CONFIG5H,0xc0
#_config _CONFIG6L,0x0f
#_config _CONFIG7L,0x0f
#_config _CONFIG7H,0x40

#if MChipBL==1
    #_config _CONFIG6H,0xa0 // Write protect boot block
    const int MCBOOTLOADER=1; // Set to 1 or 0 (or undefined) for bootloader
#else
    #_config _CONFIG6H,0xe0
#endif
#endif
```

These lines are the values for the configuration registers. Note that the boot loader protection is set according to the value of the earlier definition.

```
// The next lines are functions in USBSupport.c

void WIZUSBTasks();
void UserInit();
void InitializeSystem(void);
void USBAddTx(char x);
unsigned int USBGetRx();
unsigned char USBRxSize();
```

```
unsigned char USBCanTx();
```

These lines are the definitions of functions in USBSupport.c. They are defined in the function reference later in this manual. As there is no specific header for USBSupport.c they are included in the main file here.

```
extern unsigned char rxlen;           // length of receive buffer
```

Also in USBSupport.c – the length of the receive buffer.

```
// Finally these are the definitions for the application
unsigned char LastSw;
```

A list of variables used by the application – in this case the last switch state.

```
// This is the application start up code
void AppInit()
{
    bTRA5=0;           // Drive 3rd LED
}
```

This code is called by main and is specific to the application – in this case all we are doing is to drive bit RA5 – the 3<sup>rd</sup> LED.

```
// This is the loop for the application which checks the switch and incoming
characters

void AppLoop()
{
    char x;

    if (rxlen)           // Characters waiting in buffer
    {
        x=USBGetRx();
        USBAddTx(x);
        bRA5=x&1;           // Set LED to state of lowest bit
    }
    if (bRB4!=LastSw)   // Switch has changed state
    {
        LastSw=!LastSw;
        USBAddTx('S'+LastSw); // Show switch state
    }
}
```

This is the application loop – it is called once every time the loop in main runs. The code is non-blocking, that is that it will never stall waiting for something to happen. In our example here this function checks for received characters and processes them and checks to see if the switch has changed.

```
void main()
{
    InitializeSystem();           // Start up USB
    UserInit();                   // USB Support start up
    AppInit();                     // Local start up
    LastSw=1;
}
```

This code at the start of main is the initialisation. InitializeSystem() is the Microchip function to set up the USB registers. UserInit() is the function within the USBSupport.c file which sets up the USB LED's and AppInit() is the function described above which initialises the application and is local to this file.

## FED PIC C Library Reference

```
while(1)
{
  WIZUSBTasks();          // Run tasks
  AppLoop();
}
```

Finally this code is the main loop – it is very important. WIZUSBTasks() is in USBSupport.c and this function polls the USB interface and then checks the local transmit and receive buffers to see if data should be sent over the USB, or received from it.

## 6 Reference

### 6.1 *Non blocking code*

The USB code provided by Microchip on which these libraries are based is not interrupt based but polled. This makes the code much easier to understand and avoids timing issues.

User code must be non-blocking. This means that at no time must the user program ever enter a loop waiting for some external occurrence as in this event the USB code will not be called and the bus will almost certainly fail and disconnect itself.

The code must be written to test for events and take an action as a result.

### 6.2 *Throughput*

The maximum throughput is constrained by program performance and buffer size. A simple program should be able to sustain well in excess of 1Mbps throughput on the bus. However the maximum transfer in any USB frame is constrained by the size of the output buffer which is 64 bytes (there are two buffers of 64 bytes in practice to allow one to be filled whilst the other is being transmitted). A USB frame starts every 1mS so the defaults in our library allow a maximum of 64000 bytes per second – equivalent to a normal serial port rate of 640,000bps.

There is a similar limit on input.

Contact us if you wish to change these library defaults which have been chosen as a balance between performance and memory size.

### 6.3 *Library code status*

There are a small number of variables in use by the library which are of interest :

#### **unsigned char usb\_device\_state**

This is used by the library to show the state of the USB interface. It takes one of the following values :

```
#define DETACHED_STATE      0
#define ATTACHED_STATE      1
#define POWERED_STATE       2
#define DEFAULT_STATE       3
#define ADR_PENDING_STATE   4
#define ADDRESS_STATE       5
#define CONFIGURED_STATE    6
```

These correspond to the USB chapter 9 definitions. All we need to know is that if the value of `usb_device_state` is less than 6 then the USB interface is not configured and cannot be used. You may notice that in all of our examples we have shown this value in the variables window. It is set to the value 6 at about 50mS when the configuration stimulus file completes.

#### **unsigned char cdc\_trf\_state**

This is used by the library to show the state of the serial interface layer of the USB interface code. It takes one of the following values :

```
#define CDC_TX_READY          0
#define CDC_TX_BUSY          1
#define CDC_TX_BUSY_ZLP      2
#define CDC_TX_COMPLETING    3
```

This value is critical for functions which interface to the USB library directly – it is used by the WIZUSBTasks() function for example. If the putUSBUSART() function is called to send a buffer to the USB interface then the buffer should be considered as locked until the cdc\_trf\_state variable returns back to CDC\_TX\_READY. For example (from WIZUSBTasks()) :

```
if (txlen && (cdc_trf_state==CDC_TX_READY))
{
    putUSBUSART(tdpbase,txlen);
    txlen=0;
    if (tdpbase!=output_buffer)
        tdp=tdpbase=output_buffer;          // Back to base of buffer
    else
        tdpbase=tdp;                        // Use remainder of buffer
}
```

This code notifies the USB library that a buffer of bytes is waiting to be sent at address tdpbase. This buffer will then be locked so the code moves the buffer pointer (tdp) to a free location until the current buffer has been sent.

### LINE\_CODING line\_coding

The line\_coding structure holds the data sent by the host to set up the modem interface, this is normally set up on device configuration. An example of the use of the structure is shown above in Example 2.

The structure holds the following values :

```
typedef union _LINE_CODING
{
    struct
    {
        byte _byte[LINE_CODING_LENGTH];
    };
    struct
    {
        DWORD   dwDTERate;          // Complex data structure
        byte    bCharFormat;
        byte    bParityType;
        byte    bDataBits;
    };
} LINE_CODING;
```

This is a 7 byte structure defined by the USB Forum.

The value dwDTERate is the bit rate of the interface and is defined using a Microchip type. To read the value use :

```
unsigned long bitrate;

Bitrate=line_coding.dwDTERate._dword;
```

The value bCharFormat is the number of stop bits as follows :

```
0   1 Stop bit
1   1.5 Stop bits
```

2 2 Stop bits

The value bParityType defines parity as follows :

```

0  None
1  Odd
2  Evcn
3  Mark
4  Space

```

The value bDataBits holds the number of data bits which will be 5,6,7,8 or 16.

### **CONTROL\_SIGNAL\_BITMAP control\_signal\_bitmap**

The line\_coding structure holds the data sent by the host to set up the modem control lines, This is normally set up during device operation.

```

typedef union _CONTROL_SIGNAL_BITMAP
{
    byte _byte;
    struct
    {
        unsigned DTE_PRESENT:1;    // [0] Not Present [1] Present
        unsigned CARRIER_CONTROL:1; // [0] Deactivate [1] Activate
    };
} CONTROL_SIGNAL_BITMAP;

```

## **6.4 Functions**

The following functions are available to WIZ-C programs automatically once the USB element is included. Most require the USBSupport.C file to be copied into the project for non-WIZ programs.

---

***unsigned char USBAddTx(unsigned char x);******usbcdc.h*****Description**

This function adds a single character to the output buffer. When the USB bus is next free for transmission the entire output buffer will be flagged for transfer to the host and the buffer locked. This is entirely transparent to the application program as the buffer will flip to another location whilst the previous buffer is locked for transmission. Use the function USBCanTx() to check if the buffer is full – however this will only be required if the interface is transferring bytes at close to full capacity.

**Parameters**

**x**            The character to be transmitted.

**Return**

The function returns 1 if the character was successfully added and 0 if the output buffer is full and the character has not been added.

**Example**

```
void DoCRLF()
{
    USBAddTX('\n');
    USBAddTX('\r');
}
```

---

***unsigned int USBGetRx();******usbcdc.h*****Description**

This function reads a single character from the input buffer. If the buffer fills up owing to reads being performed less quickly than the USB interface can transfer data then the USB interface will refuse to accept further data until the buffer is freed.

**Parameters**

**None.**

**Return**

The function returns the value of the character read, or –1 if there are no characters in the input buffer.

**Example**

```
char c;
char GotIt=0;

if (USBRxSize()) {c=USBGetRx(); GotIt=1;}
```

---

***unsigned int USBRxSize();******usbcdc.h******Description***

This function returns the size of the input buffer – that is the number of characters waiting to be read from the USB interface.

***Parameters***

None.

***Return***

The function returns the number of characters waiting, or 0 if there are no characters waiting to be read.

***Example***

```
char c;
char GotIt=0;

if (USBRxSize()) {c=USBGetRx(); GotIt=1;}
```

---

***unsigned char USBCanTx();******usbcdc.h******Description***

This function returns 1 or 0 dependant on whether the transmit buffer is full. If 1 is returned then a character can be added to using USBAddTx(), if 0 is returned then the buffer is full.

***Parameters***

None.

***Return***

The function returns true (1) if the transmit buffer has space, or 0 otherwise.

***Example***

```
char HasTransmitted;

void NonBlockTx(char x)
{
    HasTransmitted=0;
    if (USBCanTx()) {HasTransmitted=1; USBAddTx(x);}
}
```

## 7 USB Development Board

The USB development board is based on our DHMicro 28 pin development board with the following components fitted :

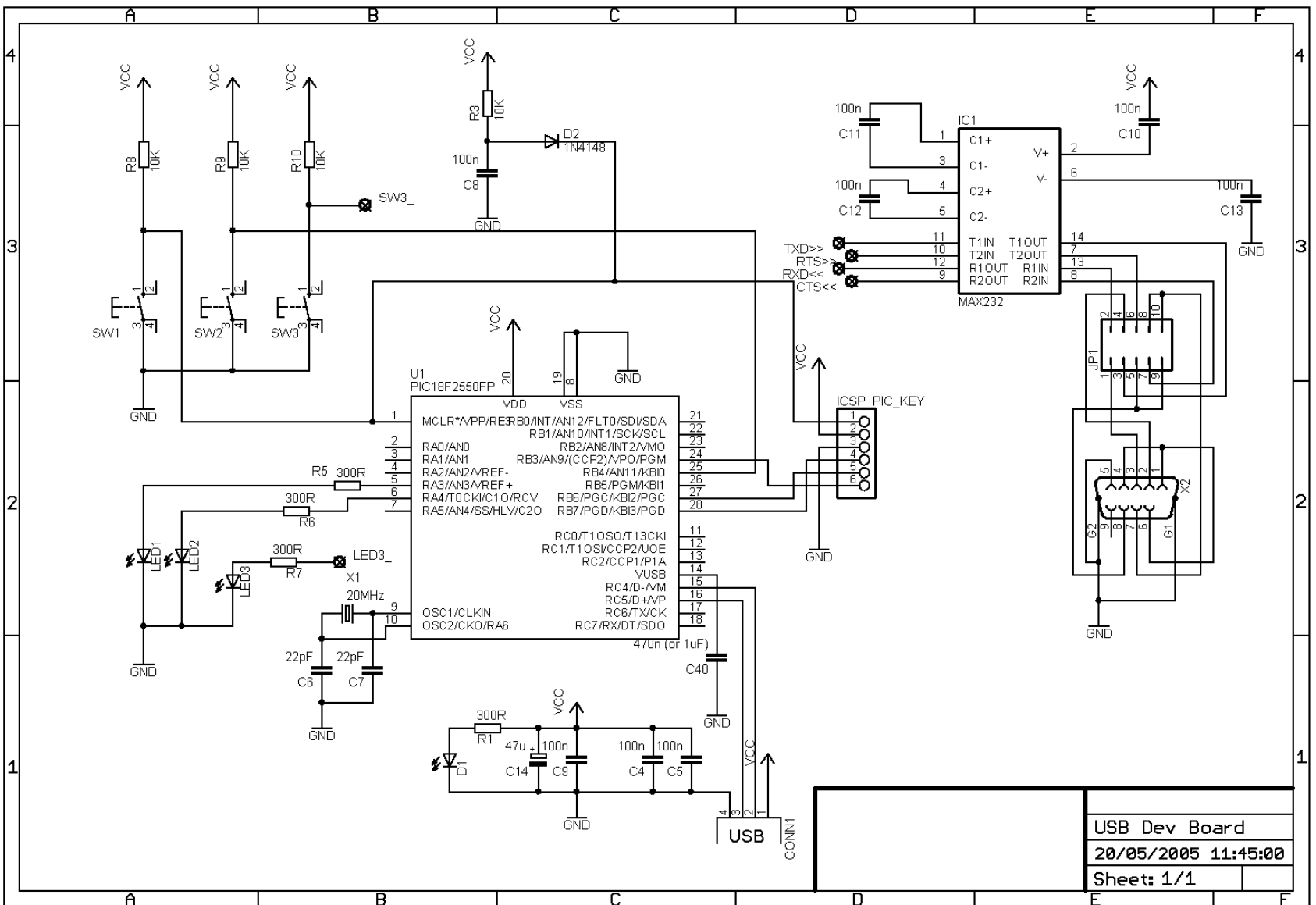
Component	Notes
LED 1	Connected to RA3 – used as a USB LED
LED 2	Connected to RA4 – used as a USB LED
LED 3	Left available for the user
SW1	Connected to Reset
SW2	Connected to pin RB4 – used with SW1 to select between boot loader and normal operation, may also be used as a general purpose press switch
SW3	Left available for the user
IC2	Serial interface (RS-232) – left available for the user
JU1	ICSP connector – Fitted to allow the user of PIC Key to program the device
USB Conn	A USB connector is fitted and connected together with the USB decoupling capacitor

Check the development board manual which is supplied on the CD for details of other components which are tracked but not fitted. Note that the buzzer and external power supply components are not fitted but may be added if required.

As supplied the board is programmed with the Microchip boot loader and the loopback example.

The circuit diagram for the board as supplied is shown below (Note that components tracked but not fitted are not shown).

Use a 20MHz or 4MHz crystal



## 8 Microchip Licence agreement

You must agree to the following conditions before using any of the code provided with the USB development kit.

### IMPORTANT:

MICROCHIP TECHNOLOGY INC. ("COMPANY") IS WILLING TO LICENSE USB FRAMEWORK SOFTWARE AND ACCOMPANYING DOCUMENTATION OFFERED TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "**I ACCEPT**" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "**I DO NOT ACCEPT**," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

### NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT FOR MICROCHIP USB FRAMEWORK SOFTWARE IMPORTANT - READ CAREFULLY.

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, either as an individual or a single entity, ("Licensee") and Microchip Technology Incorporated ("Company") for Company's USB Framework software which may include firmware source code ("Software") and accompanying proprietary documentation ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. The Software and Documentation are protected by U.S. copyright laws and international copyright treaties, and other intellectual property laws and treaties.

In consideration of the mutual covenants of the parties and for the consideration received herein, the parties hereby agree as follows:

**1. Ownership of Proprietary Rights.** Except as expressly licensed herein, Company retains all right, title and interest in and to the Software ("Proprietary Rights"), including, but not limited to: (i) patent, copyright, trade secret and similar right in the Software and underlying technology; (ii) all copies and derivative works thereof (by whomever produced) and (iii) the Documentation. Licensee shall have no right to sell, assign or otherwise transfer all or any portion of the Software or the Proprietary Rights for the Software except as expressly set forth in the Agreement. Except to the extent provided in this Agreement, all uses of the Proprietary Rights of the Software hereunder, including hardware, firmware and software implementations of the technology, will inure to the benefit of the Company, and any all equities or rights in and to the Proprietary Rights of the Software resulting from Licensee's acts or endeavors hereunder will automatically transfer to the Company.

### 2. License Grant.

(a) Subject to the terms of this Agreement, Company grants, strictly to Licensee, a personal, worldwide, fully paid-up, non-exclusive, non-transferable limited license (without the right to sublicense) to reproduce, market, license, distribute, use and create derivative works of the Software, solely for incorporation into a product manufactured by Licensee which only implements the Software on a proprietary products manufactured by Company and purchased by Licensee from Company or Company's authorized distributor (a "Device"). The product produced by the Licensee which integrates a Device programmed with the Software shall hereafter be called a "Licensee Product." The Company also grants, strictly to Licensee, a personal, worldwide, fully paid-up, non-exclusive, non-transferable limited license (without the right to sublicense) to use the Documentation in support of Licensee's authorized use of the Software.

(b) Licensee may enter into agreements with a person or entity not a party to this Agreement to program the Software into the Device (hereafter "Subcontractors"); **provided, however,** that such Subcontractor assents to and agrees to be bound by terms and conditions similar to this Agreement directly with Company.

(c) Authorized representatives of the Company shall have the right to reasonably inspect

Licensee's premises and to audit Licensee's records and inventory of Licensee Products, whether located on Licensee's premises or elsewhere at any time, announced or unannounced, and in its sole and absolute discretion, in order to ensure Licensee's adherence to the terms of this Agreement.

### 3. Licensee Obligations.

(a) Licensee will not (i) engage in unauthorized sublicensing, disclosure or distribution of Software or Documentation, (ii) use all or any portion of the Software or the Proprietary Rights of the Software except in conjunction with a Device; or (iii) reverse engineer (by disassembly, decompilation or otherwise) the Software. For purposes of clarity, Licensee's use of the Software binary driver file on a Windows® operating system to interface with a Device does not violate this Section 3(a).

(b) Licensee will not, during the term of this Agreement or any time thereafter, attack, dispute or contest, directly or indirectly, the Company's exclusive right, title and interest in or to the Proprietary Rights of the Software or the validity of the Company's ownership thereof, nor will Licensee assist or encourage others to do so.

### 4. Indemnification.

(a) Licensee Indemnity. Licensee will defend, indemnify and hold Company harmless from and against any and all claims, losses, liabilities, damages, costs, and expenses (including attorney's fees) directly or indirectly arising from or related to (i) any use or disclosure of the Software not permitted under this Agreement, (ii) any use of the Software in combination with other products, equipment, software or data not supplied by Company, including but not limited to use in the Licensee Product, (iii) any modification of the Software made by any person other than Company, or (iv) any products liability claims with respect to the Licensee Product.

(b) **THE FOREGOING STATES THE SOLE AND EXCLUSIVE REMEDY AND LIABILITY OF THE PARTIES FOR INTELLECTUAL PROPERTY INFRINGEMENT.**

### 5. Term and Termination.

(a) **Term.** This Agreement and the license granted herein shall be effective from the date of acceptance of the Agreement, or first installation, copy or use of the Software, whichever event occurs first. The term of this Agreement shall run until terminated in accordance with this Section 5.

(b) **Termination for Cause.** This Agreement may be terminated if either party materially fails to perform or comply with any provisions of this Agreement. In the event of Licensee's breach, termination shall be effective immediately without notice from Company.

(c) **Termination for Convenience.** This Agreement may be terminated by either party for convenience upon 60 days written notice.

(d) **Effects of Termination.** Upon termination, Licensee shall immediately discontinue all use of the Software, including but limited to inclusion in any Licensee Product, and shall destroy any tangible media Licensee has on which the Software or Documentation exist, and remove the Software and Documentation from any and all systems.

**6. Injunctive Relief.** Licensee agrees that the provisions in this Agreement regarding unauthorized use of the Software and nondisclosure are necessary to protect the legitimate business interests of the Company. Licensee also agrees that monetary damages alone cannot adequately compensate the Company if there is a violation of such provisions by Licensee and that injunctive relief against Licensee is essential for the protection of the Company. Licensee agrees, therefore, that if the Company alleges that Licensee has breached or violated such provisions then, in addition to any other remedies it may have, the Company will have the right to petition a court of competent jurisdiction, with the requirement for the posting of a bond, for injunctive relief against Licensee in addition to all other remedies at law or in equity.

**7. Confidentiality.** Licensee agrees that all source code, source documentation and

underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are the Company's proprietary information ("**Proprietary Information**"). Except as expressly and unambiguously allowed herein, the Licensee will hold in confidence and not use or disclose any Proprietary Information and shall similarly bind its employees and Subcontractors in writing. Proprietary Information shall not include information that (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by it independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If a party is required to disclose Proprietary Information by law, court order, or government agency, such disclosure shall not be deemed a breach of this Agreement. Licensee's obligation under this Section 7 will survive for a period of twenty (20) years following the expiration or termination of this Agreement.

**8. Warranties and Disclaimers.** THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. LICENSEE ASSUMES THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION, AND COMPANY ASSUMES NO RESPONSIBILITY FOR THE ACCURACY OR APPLICATION OR OF ERRORS OR OMISSIONS IN THE SOFTWARE. THE LIMITED REMEDIES SET FORTH HEREIN SHALL APPLY NOTWITHSTANDING FAILURE OF THEIR ESSENTIAL PURPOSE.

**10. Limited Liability.** IN NO EVENT SHALL COMPANY BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Company for damages hereunder will in no event exceed \$100 and Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

**11. General.**

(a) **Electronic Notices and Requests.** When Licensee visits Company's website or send e-mails to the Company, Licensee is communicating with Company electronically. Licensee hereby consents to receive communications from Company electronically. Company will communicate with Licensee by e-mail or by posting notices on www.Microchip.com. Licensee agrees that all agreements, notices, disclosures and other communications that Company provide to Licensee electronically satisfy any legal requirement that such communications be in writing. In giving notice to Company, Licensee will address notices to Legal.Department@Microchip.com. and include Licensee's name, company name, physical address and phone number, and include "USB Framework Software" in the subject line.

(b) **Governing Law.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS, THEREOF AND WITHOUT REGARD TO THE UNITED NATIONS CONVENTION ON CONTRACTS FOR THE INTERNATIONAL SALE OF GOODS. The sole jurisdiction and venue for actions related to the subject matter hereof shall be the state and federal courts located in Arizona.

(c) **Attorneys' Fees.** If either the Company or Licensee employs attorneys to enforce any

**FED PIC C Library Reference**

rights arising out of or relating to this Agreement, the prevailing party shall be entitled to recover its reasonable attorneys' fees, costs and other expenses.

(d) **Entire Agreement.** This Agreement shall constitute the entire agreement between the parties with respect to the subject matter hereof. It shall not be modified except by a written agreement signed by an authorized representative of the Company.

(e) **Severability.** If any provision of this Agreement shall be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect and enforceable.

(f) **Waiver.** No waiver of any breach of any provision of this Agreement shall constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver shall be effective unless made in writing and signed by an authorized representative of the waiving party.

(g) **Export Regulation.** Each party hereby agrees to comply with all export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority.

(h) **Survival.** The indemnities herein, and any right of action for breach of this Agreement prior to termination shall survive any termination of this Agreement.

(i) **Assignment.** Neither this Agreement nor any rights, licenses or obligations hereunder, may be assigned by Licensee without the prior written approval of the Company.