# AN956

# Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software

*Author:* *Rawin Rojvanit*
*Microchip Technology Inc.*

## INTRODUCTION

The RS-232 serial interface is no longer a common port found on a personal computer (PC). This is a problem because many embedded applications use the RS-232 interface to communicate with external systems, such as PCs. A solution is to migrate the application to the Universal Serial Bus (USB) interface. There are many different ways to convert an RS-232 interface to USB, each requiring different levels of expertise. The simplest method is to emulate RS-232 over the USB bus. An advantage of this method is the PC application will see the USB connection as an RS-232 COM connection and thus, require no changes to the existing software. Another advantage is this method utilizes a Windows® driver included with Microsoft® Windows® 98SE and later versions, making driver development unnecessary.

The objectives of this application note are to explain some background materials required for a better understanding of the serial emulation over USB method and to describe how to migrate an existing application to USB. A device using the implementation discussed in this document shall be referred to as a USB RS-232 emulated device. The author assumes that the reader has some basic knowledge of the USB standard. All references to the USB specification in this document refer to USB specification revision 2.0.

Features in version 1.0 of the RS-232 Emulation firmware:

- A relatively small code footprint of 3 Kbytes for the firmware library
- Data memory usage of approximately 50 bytes (excluding the data buffer)
- Maximum throughput speed of about 80 Kbytes
- Data flow control handled entirely by the USB protocol (RS-232 XON/XOFF and hardware flow control are omitted in this version)
- Does not require additional drivers; all necessary files, including the `.inf` files for Microsoft® Windows® XP and Windows® 2000, are included

## OVERVIEW

A Windows application sees a physical RS-232 connection as a COM port and communicates with the attached device using the CreateFile, ReadFile, and WriteFile functions. The UART module on the PICmicro® device provides an embedded device with a hardware interface to this RS-232 connection. When switching to USB, the Windows application can see the USB connection as a virtual COM port via services provided by two Windows drivers, `usbser.sys` and `ccport.sys`. In-depth details regarding these Windows drivers are outside the scope of this document. A virtual COM port provides Windows applications with the same programming interface; therefore, modification to the existing application PC software is unnecessary.

The areas that do require changes are the embedded hardware and firmware. For hardware, a microcontroller with an on-chip full speed USB peripheral is required to implement this solution. The PIC18F2455/2550/4455/4550 family of microcontrollers is used here as an example. References to the device data sheet in this document apply to the *"PIC18F2455/2550/4455/4550 Data Sheet"* (DS39632).

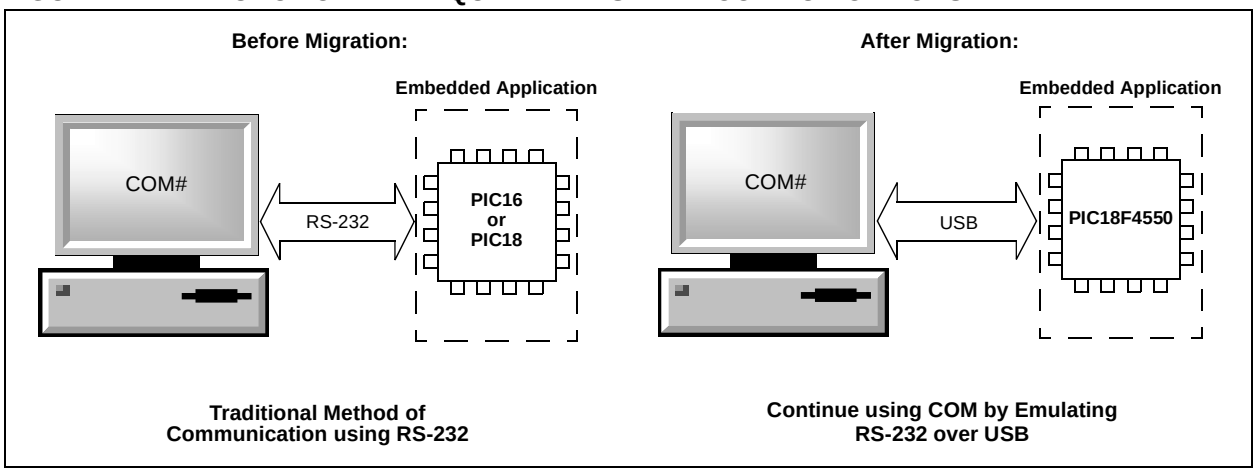
Firmware modifications to the existing application code are minimal; only small changes are needed to call the new USB UART functions provided as part of the USB firmware framework written in C. Figure 1 shows an overview of the migration path.

Migrating to USB using the RS-232 serial emulation method provides the following advantages:

- It has little or no impact on the PC software application
- It requires minimal changes to the existing application firmware
- It shortens the development cycle time
- It eliminates the need to support and maintain a Windows driver which is a very demanding task
- Finally, the method described here utilizes a clear migration path from many existing PICmicro devices to the PIC18F2455/2550/4455/4550 family of microcontrollers, making the upgrade to USB straightforward

Since the USB protocol already handles the details of low-level communication, the concept of baud rate, parity bit and flow control for the RS-232 standard becomes abstract.

**FIGURE 1: FUNCTIONALLY EQUIVALENT SERIAL COMMUNICATIONS**




## USB CDC Specification

The Communication Device Class (CDC) specification defines many communication models, including serial emulation. All references to the CDC specification in this document refer to version 1.1. The Microsoft Windows driver, `usbser.sys`, conforms to this specification. Therefore, the embedded device must also be designed to conform with this specification in order to utilize this existing Windows driver.

The CDC specification describes an abstract control model for serial emulation over USB in Section 3.6.2.1. In summary, two USB interfaces are required. The first one is the Communication Class interface, using one IN interrupt endpoint. This interface is used for notifying the USB host of the current RS-232 connection status from the USB RS-232 emulated device. The second one is the Data Class interface, using one OUT bulk endpoint and one IN bulk endpoint. This interface is used for transferring raw data bytes that would normally be transferred over the real RS-232 interface.

Designers do not have to worry about creating descriptors or writing function handlers for Class-Specific Requests. Descriptors for a USB RS-232 emulated device and all required handlers for Class-Specific Requests listed in Table 4 of the CDC specification are provided with the Microchip USB CDC firmware. A diagram of the descriptors structure for a USB RS-232 emulated device can be found in **Appendix A: "CDC Descriptor Schema for Serial Emulation"**. An example of the CDC descriptors can also be found in Section 5.3 of the CDC specification.

## Class-Specific Requests and Notifications

Microchip USB firmware implements the following Class-Specific Request handlers as listed in Table 4 of the CDC specification:

- SEND_ENCAPSULATED_COMMAND
- GET_ENCAPSULATED_COMMAND
- SET_LINE_CODING
- GET_LINE_CODING
- SET_CONTROL_LINE_STATE

More support will be provided in subsequent versions of the USB UART firmware. Currently, the firmware does not return any RESPONSE_AVAILABLE notification but it returns a NAK instead to tell the host that no response is available. Refer to Sections 3.6.2.1, 6.2 and 6.3 of the CDC specification for more details.

# MICROCHIP USB FIRMWARE: USB UART FUNCTIONS

All references to the USB UART functions in this document refer to functions provided in version 1.0 of the CDC firmware driver. This driver is provided as part of the Microchip USB firmware framework. The functions provided in this library are very similar in functionality to ones provided in the MPLAB® C18 USART library. The specific functions are listed in Table 1.

The choice of selecting which function to use depends on several factors:

- Is the string of data null-terminated?
- Is the length of the data string known?
- Is the data source located in program memory or in RAM?

**TABLE 1: SUMMARY OF THE USB UART FUNCTIONS**

| Function | Description |
|---|---|
| putrsUSBUSART | Write a null-terminated string from program memory to the USB. |
| putsUSBUSART | Write a null-terminated string from data memory to the USB. |
| mUSBUSARTTxRom | Write a string of known length from program memory to the USB. |
| mUSBUSARTTxRam | Write a string of known length from data memory to the USB. |
| mUSBUSARTIsTxTrfReady | Is the driver ready to accept a new string to write to the USB? |
| getsUSBUSART | Read a string from the USB. |
| mCDCGetRxLength | Read the length of the last string read from the USB. |

### putrsUSBUSART

putrsUSBUSART writes a string of data to the USB including the null character. Use this version to transfer literal strings of data, or data located in program memory, to the host.

**Note:** The transfer mechanism for device-to-host (put) is more flexible than host-to-device (get). It can handle a string of data larger than the maximum size of the bulk IN endpoint. A state machine is used to transfer a long string of data over multiple USB transactions. This background service is carried out by CDCTxService().

#### Syntax:

```
void putrsUSBUSART(const rom char *data)
```

#### Precondition:

mUSBUSARTIsTxTrfReady() must return '1' before this function can be called. The string of characters pointed to by data must be equal to or smaller than 255 bytes, including the null-terminated character.

#### Input:

data

Pointer to a null-terminated string of data. If a null character is not found, only the first 255 bytes of data will be transferred to the host.

#### Output:

None

#### Side Effects:

None

#### Example:

```
void example_1(void)
{
        if(mUSBUSARTIsTxTrfReady())
                putrsUSBUSART("Hello World.");
}//end example_1

rom char example_string[] = {"Microchip"};
void example_2(void)
{
        if(mUSBUSARTIsTxTrfReady())
                putrsUSBUSART(example_string);
}//end example_2
```

### putsUSBUSART

putsUSBUSART writes a string of data to the USB including the null character. Use this version to transfer data located in data memory to the host.

**Note:** The transfer mechanism for device-to-host (put) is more flexible than host-to-device (get). It can handle a string of data larger than the maximum size of the bulk IN endpoint. A state machine is used to transfer a long string of data over multiple USB transactions. This background service is carried out by CDCTxService().

#### Syntax:

```
void putsUSBUSART(char *data)
```

#### Precondition:

mUSBUSARTIsTxTrfReady() must return '1' before this function can be called. The string of characters pointed to by data must be equal to or smaller than 255 bytes, including the null-terminated character.

#### Input:

data

Pointer to a null-terminated string of data. If a null character is not found, only the first 255 bytes of data will be transferred to the host.

#### Output:

None

#### Side Effects:

None

#### Example:

```
char example_string[4];
void example_1(void)
{
        example_string[0]='U';
        example_string[1]='S';
        example_string[2]='B';
        example_string[3]=0x00;
        if(mUSBUSARTIsTxTrfReady())
                putsUSBUSART(example_string);
}//end example_1
```

**mUSBUSARTTxRom**

Use this macro to transfer data located in program memory. The length of data to be transferred must be known and passed in as a parameter.

The response of this function is undefined if it is called when `mUSBUSARTIsTxTrfReady()` returns '`0`'.

**Note:** This macro only handles the setup of the transfer. The actual transfer is handled by `CDCTxService()`.

### Syntax:

```
void mUSBUSARTTxRom(rom byte *pData, byte len)
```

### Precondition:

`mUSBUSARTIsTxTrfReady()` must return '`1`' before this function can be called. Value of `len` must be equal to or smaller than 255 bytes.

### Input:

`pDdata`

Pointer to the starting location of data bytes.

`len`

Number of bytes to be transferred.

### Output:

None

### Side Effects:

None

### Example:

```
rom char example_string[] = {0x31,0x32,0x33};
void example_1(void)
{
        if(mUSBUSARTIsTxTrfReady())
                mUSBUSARTTxRom((rom byte*)example_string,3);
}//end example_1
```

**mUSBUSARTTxRam**

Use this macro to transfer data located in data memory. The length of data to be transferred must be known and passed in as a parameter.

The response of this function is undefined if it is called when `mUSBUSARTIsTxTrfReady()` returns '0'.

**Note:** This macro only handles the setup of the transfer. The actual transfer is handled by `CDCTxService()`.

### Syntax:

```
void mUSBUSARTTxRam(byte *pData, byte len)
```

### Precondition:

`mUSBUSARTIsTxTrfReady()` must return '1' before this function can be called. Value of `len` must be equal to or smaller than 255 bytes.

### Input:

`pDdata`

Pointer to the starting location of data bytes.

`len`

Number of bytes to be transferred.

### Output:

None

### Side Effects:

None

### Example:

```
char example_string[3];
void example_1(void)
{
        example_string[0] = 'U';
        example_string[1] = 'S';
        example_string[2] = 'B';
        if(mUSBUSARTIsTxTrfReady())
                mUSBUSARTTxRam((byte*)example_string,3);
}//end example_1
```

### mUSBUSARTIsTxTrfReady

This macro is used to check if the CDC class is ready to send more data.

Typical Usage: `if(mUSBUSARTIsTxTrfReady())`

**Note:** Do not call this macro as a blocking function (i.e., `while(!mUSBUSARTIsTxTrfReady());`).

#### Syntax:

`BOOL mUSBUSARTIsTxTrfReady(void)`

#### Precondition:

None

#### Input:

None

#### Output:

BOOL

If the firmware driver is ready to receive a new string of data to write to USB, it will return '`1`', else it will return '`0`'.

#### Side Effects:

None

#### Example:

```
void example_1(void)
{
        if(mUSBUSARTIsTxTrfReady())
                putrsUSBUSART("Microchip");
}//end example_1
```

### getsUSBUSART

getsUSBUSART copies a string of bytes received through USB CDC bulk OUT endpoint to a user's specified location. It is a non-blocking function. It does not wait for data if there is no data available. Instead, it returns '0' to notify the caller that there is no data available.

**Note:** If the actual number of bytes received is larger than the number of bytes expected (len), only the expected number of bytes specified will be copied to buffer. If the actual number of bytes received is smaller than the number of bytes expected (len), only the actual number of bytes received will be copied to the buffer.

#### Syntax:

```
byte getsUSBUSART(char *buffer, byte len)
```

#### Precondition:

Value of input argument, len, should be equal to or smaller than the maximum endpoint size responsible for receiving bulk data from USB host for CDC class.

This maximum endpoint size value is defined as CDC_BULK_OUT_EP_SIZE and is found in the file usbcfg.h.

CDC_BULK_OUT_EP_SIZE could be equal to 8, 16, 32 or 64 bytes.

Input argument buffer should point to a buffer area that is bigger or equal to the size specified by len.

#### Input:

buffer

Pointer to where received bytes are to be stored.

len

The number of bytes expected.

#### Output:

byte

The number of bytes copied to the buffer.

#### Side Effects:

Publicly accessible variable cdc_rx_len is updated with the number of bytes copied to buffer. Once getsUSBUSART is called, subsequent retrieval of cdc_rx_len can be done by calling macro mCDCGetRxLength().

#### Example:

```
char input_buffer[64];
void example_1(void)
{
        byte index, count, total_sum;
        if(getsUSBUSART(input_buffer, 8)
        {
                count = mCDCGetRxLength();
                total_sum = 0;
                for(index = 0; index < count; index++)
                        total_sum += input_buffer[index];
        }//end if
}//end example_1

void example_2(void)
{
        if(getsUSBUSART(input_buffer, CDC_BULK_OUT_EP_SIZE)
        {
                // Do something…
        }//end if
}//end example_2
```

### mCDCGetRxLength

mCDCGetRxLength is used to retrieve the number of bytes copied to user's buffer by the most recent call to getsUSBUSART function.

#### Macro:

byte mCDCGetRxLength(void)

#### Precondition:

None

#### Input:

None

#### Output:

byte

mCDCGetRxLength returns the number of bytes copied to user's buffer from the last call to getsUSBUSART.

#### Side Effects:

None

#### Example:

```
char input_buffer[64];
void example_1(void)
{
        if(getsUSBUSART(input_buffer, 2)
        {
                // Do something with input_buffer[0]

                if(mCDCGetRxLength() == 2)

                // Do something with input_buffer[1]
        }//end if
}//end example_1
```

## Important Things to Know When Using the USB UART Functions

While the provided USB UART functions greatly simplify the integration of USB into an application, there are still some considerations to keep in mind when developing or modifying application code.

### CODE DESIGN

1. The Microchip USB firmware is a cooperative multitasking environment. There should not be any blocking functions in the user code. As USB tasks are polled and serviced in the main program loop, any blocking functions that are dependent on the state of the USB may cause a deadlock. Use a state machine in place of a blocking function.
2. `mUSBUSARTTxRom` and `mUSBUSARTTxRam` expect a data pointer of type `rom byte*` and `byte*`, respectively. Type casting may be necessary.
3. `while(!mUSBUSARTIsTxTrfReady());` is a blocking function. Do not use it.
4. `putrsUSBUSART`, `putsUSBUSART`, `mUSBUSARTTxRom` and `mUSBUSARTTxRam` are not blocking functions. They do not send out data to the USB host immediately, nor wait for the transmission to complete. All they do is set up the necessary Special Function Registers and state machine for the transfer operation.

   The routine that actually services the transfer of data to the host is `CDCTxService()`. It keeps track of the state machine and breaks up long strings of data into multiple USB data packets. It is called once per main program loop in the `USBTasks()` service routine. (The state machine for `CDCTxService()` is shown in **Appendix B: "CDC State Machine"**.)

   Because of this, back-to back function calls will not work; each new call will override the pending transaction. The correct way of sending consecutive strings is to use a state machine, as shown in Example 1. An alternative to using a state machine is to use a global intermediate buffer, as shown in Example 2.
5. A correct set of descriptors for the CDC class must be used. Refer to the reference design project for an example.
6. The endpoint size for the data pipes are defined by CDC_BULK_OUT_EP_SIZE and CDC_BULK_IN_EP_SIZE, located in the header file `usbcfg.h`. Since these endpoints are of type bulk, the maximum endpoint size described must either be 8, 16, 32 or 64 bytes.
7. The type `byte` is defined as an `unsigned char` in the header file `typedefs.h`.
8. Always check whether the firmware driver is ready to send more data out to the USB host by calling `mUSBUSARTIsTxTrfReady()`.

**EXAMPLE 1: CORRECT USE OF USB UART FUNCTION CALLS**

**Incorrect Method (back-to-back calls):**

```
if(mUSBUSARTIsTxTrfReady())
{
    putrsUSBUSART("Hello World");
    putrsUSBUSART("Hello Again");
}//end if
```

**Correct Method (state machine):**

```
byte state = 0;
if(state == 0)
{
    if(mUSBUSARTIsTxTrfReady())
    {
        putrsUSBUSART("Hello World");
        state++;
    }//end if
}
else if(state == 1)
{
    if(mUSBUSARTIsTxTrfReady())
    {
        putrsUSBUSART("Hello Again");
        state++;
    }//end if
}//end if
```

**EXAMPLE 2: ALTERNATIVE GLOBAL BUFFER METHOD**

```
char io_buffer[64];
. . .
// Main Program Loop
while(1)
{
    USBTasks();
    if (mUSBUSARTIsTxTrfReady())
        putsUSBUSART(io_buffer);
    // SendToBuffer attachs multiple
    // strings together
    // (The user will need to provide
    // their own SendToBuffer function)
    SendToBuffer("Hello World");
    SendToBuffer("Hello Again");
}// end while
```

### SETTING UP THE CODE PROJECT

1. Insert `#include "system\usb\usb.h"` in each file that uses the CDC functions.
2. USB_USE_CDC should be defined in the file `usbcfg.h` when using the CDC functions.
3. The source and header files, `cdc.c` and `cdc.h`, should be added to the project source files. They can be found in the directory "`system\usb\class\cdc\`".

### USB Vendor ID (VID) and Product ID (PID)

The VID and PID are important because they are used by the Windows operating system to differentiate USB devices and to determine which device driver is to be used. The VID is a 16-bit number assigned by the USB Implementers Forum (USB-IF). It must be obtained by each manufacturer that wants to market and sell USB products. The VID can be purchased directly from USB-IF. More detailed information can be found at: http://www.usb.org/developers/vendor.

Each VID comes with 65,536 different PIDs which is also a 16-bit number. In the Microchip USB firmware framework, the VID and PID are located in the file `usbdsc.c`. Both values can be modified to match different product VID and PID numbers.

### Drivers for Microsoft Windows® 2000 and Windows® XP

Microsoft Windows does not have a standard `.inf` file for the CDC driver. The drivers are, however, part of the Windows installation. The only thing necessary to do is to provide an `.inf` file when a CDC device is first connected to a Windows system.

Example `.inf` files are provided with the CDC RS-232 Emulation Reference Project and are located in the source code directory `<Install>\fw\CDC\inf`. Before using them, they must be modified to reflect the application's specific VID and PID. This is in addition to any changes to `usbdsc.c` that have already been made and must match those values. The VID and PID are located in the string "`USB\VID_xxxx&PIDyyyy`", where "`xxxx`" is the hexadecimal VID and "`yyyy`" is the hexadecimal PID. The string is generally part of one of the lines under the heading "`[DeviceList]`".

If desired, users may also modify the variable definitions under the heading "`[Strings]`". This changes the device identification text seen by the user in the device manager.

## SUMMARY

The RS-232 serial emulation provides an easy migration path for applications moving from UART to USB. On the PC side, it requires minimal software modification. On the embedded device side, the PIC18F2455/2550/4455/4550 family of microcontrollers provides a simple hardware upgrade path from the PIC16C745/765 and PIC18FXX2 families of devices. Library firmware with user-friendly APIs are also included for convenience. Tutorial exercises are available as part of the CDC RS-232 Emulation Reference Project.

## REFERENCES

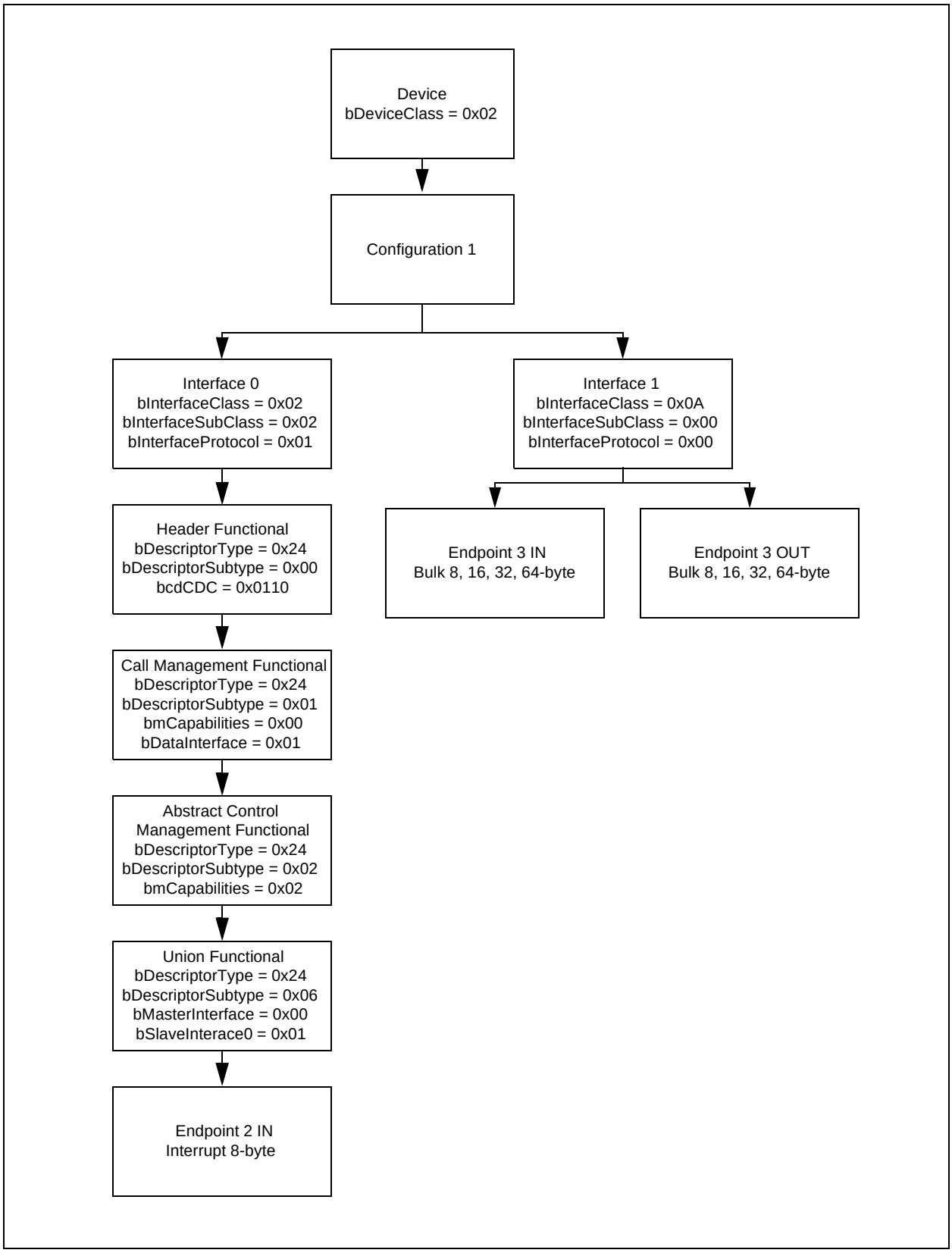
*"PIC18F2455/2550/4455/4550 Data Sheet"* (DS39632), Microchip Technology Inc., 2004.

*"USB Specification Revision 2.0"*, USB Implementers Forum Inc., 2000.

*"USB Class Definitions for Communication Devices Version 1.1"*, USB Implementers Forum Inc., 1999.

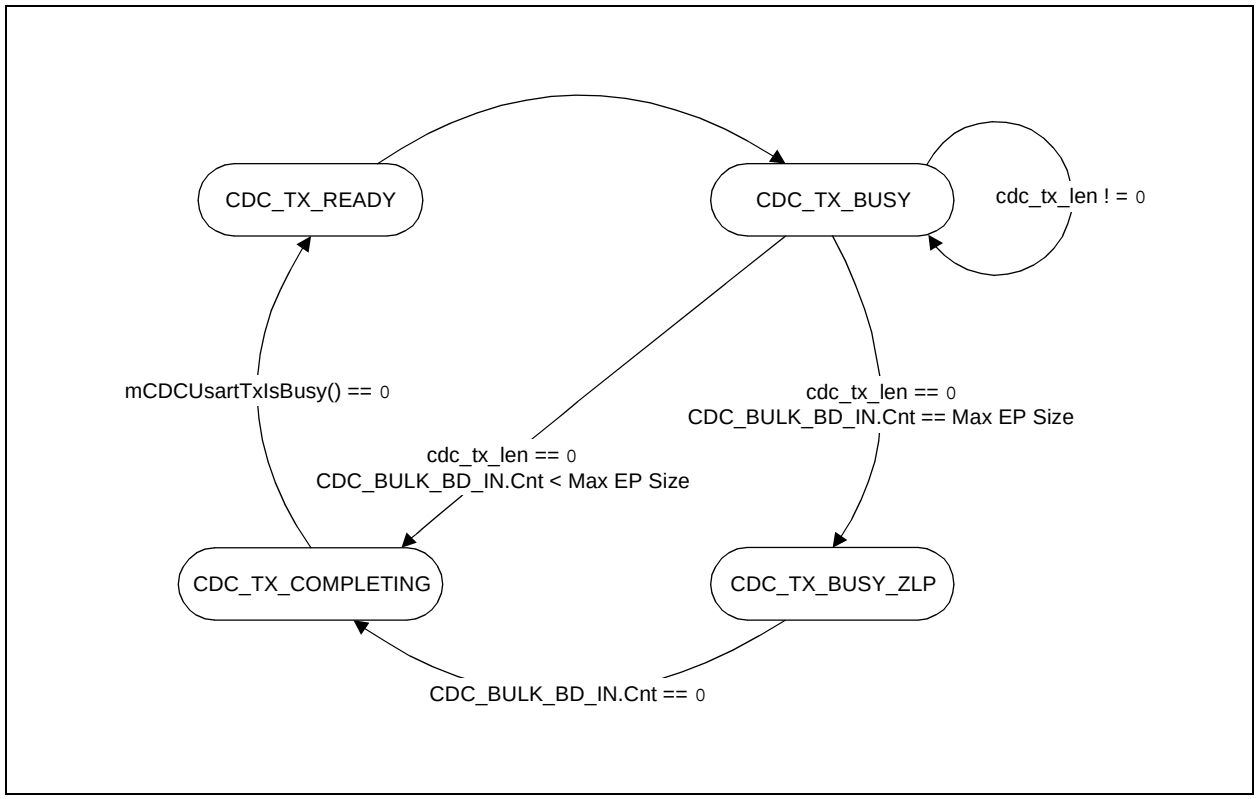
## APPENDIX A: CDC DESCRIPTOR SCHEMA FOR SERIAL EMULATION

**FIGURE A-1: EXAMPLE OF A DESCRIPTOR SET FOR SERIAL EMULATION**



## APPENDIX B: CDC STATE MACHINE

**FIGURE B-1: DIAGRAM OF THE `CDCTxService()` STATE MACHINE**




## APPENDIX C: SOURCE CODE

Because of its size, the complete source code for this application note is not included in this text. You may download the complete source code, including all necessary files and the Microsoft Windows `.inf` file, from the Microchip web site at the internet address:

**www.microchip.com**

## APPENDIX D: NOTE ON THE CDC RS-232 EMULATION TUTORIAL

The CDC RS-232 Emulation Reference Project provides a complete demonstration of the application discussed in this document, as well as a tutorial for developing applications. The tutorial exercises are written specially for the PICDEM™ FS USB Demo Board. Modifications can be made to run the source project on a different platform. The tutorial is included as part of the source code project. Refer to the `user.c` file in the project for more information.

When using the HyperTerminal program in Microsoft Windows, physically or electrically reconnecting the USB device causes the Windows operating system to assign a new driver handler for that particular device, thus causing any applications that have the path to the old handle to stop communicating. In HyperTerminal, hanging up the connection before disconnecting the device allows the program to re-enumerate the device COM port and therefore, reference the correct driver handler.

**NOTES:**

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.



**Trademarks**



**QUALITY MANAGEMENT SYSTEM**
**CERTIFIED BY DNV**
**ISO/TS 16949:2002**

*Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

# WORLDWIDE SALES AND SERVICE

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

**Atlanta**
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

**Boston**
Westford, MA
Tel: 978-692-3848
Fax: 978-692-3821

**Chicago**
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

**Dallas**
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

**Detroit**
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

**Kokomo**
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

**San Jose**
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

**Toronto**
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

## ASIA/PACIFIC

**Australia - Sydney**
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

**China - Beijing**
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

**China - Chengdu**
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

**China - Fuzhou**
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

**China - Hong Kong SAR**
Tel: 852-2401-1200
Fax: 852-2401-3431

**China - Shanghai**
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

**China - Shenyang**
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

**China - Shenzhen**
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

**China - Shunde**
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

**China - Qingdao**
Tel: 86-532-502-7355
Fax: 86-532-502-7205

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

**India - New Delhi**
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

**Japan - Kanagawa**
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

**Korea - Seoul**
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

**Singapore**
Tel: 65-6334-8870
Fax: 65-6334-8850

**Taiwan - Kaohsiung**
Tel: 886-7-536-4818
Fax: 886-7-536-4803

**Taiwan - Taipei**
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

**Taiwan - Hsinchu**
Tel: 886-3-572-9526
Fax: 886-3-572-6459

## EUROPE

**Austria - Weis**
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

**Denmark - Ballerup**
Tel: 45-4450-2828
Fax: 45-4485-2829

**France - Massy**
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

**Germany - Ismaning**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**England - Berkshire**
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/20/04