Freescale MQX RTOS Example Guide

CDC_virtual_com example

This document explains the CDC_virtual_com example, what to expect from the example and a brief introduction to the API.

The example

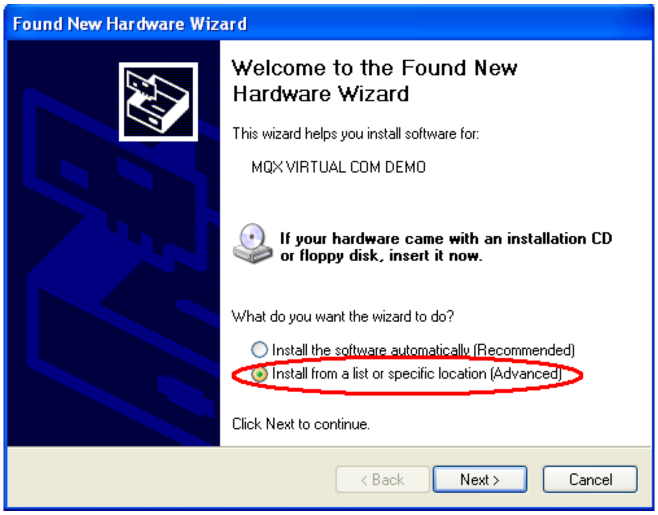
The CDC_virtual_com example enumerates as a USB CDC Class Device. The demo uses the CDC class virtues to enumerate and appear as a COM port on the USB host. This means that hosts can communicate with the USB device with the API commonly used to communicate with devices attached to a physical COM port using UART.

The implemented CDC class allows for very simple and fast development as it uses very simple API to send and receive data. On the device side (the MQX code) it consists of just a couple of buffers, status variables and functions. On the host site, as mentioned above, normal COM API functions are used. In case of a PC host, a simple terminal program can be used to run the demo, which just echoes characters received from the virtual USB UART. The example is working in a half-duplex scheme.

Running the example

Connect already running board to USB port of the host. Please do not confuse the MCU USB port cable and the BDM USB cable, MCU USB port has to be used in this case. On most of the Freescale demo boards, the device USB port will be a mini type USB connector while the debug port will use a standard sized connector.

Once the USB cable is connected, the computer will enumerate the USB device. This will usually cause an audible alarm and a pop-up to appear on the screen. The first time this is done, the computer will request an information file for driver installation. It may be necessary to manually search for this file manually.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

MQX VIRTUAL COM DEMO

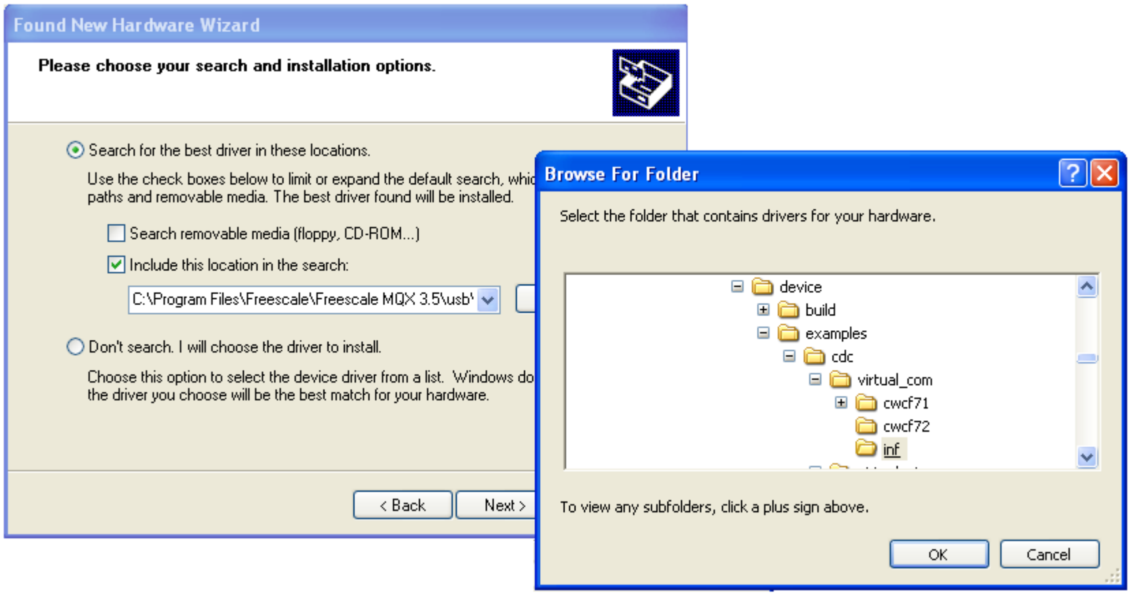
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)
● Install from a list or specific location (Advanced)

Click Next to continue.

Note: CDC devices are not a standard preinstalled class in current Windows operating systems, but there is a driver file presenting USB CDC devices as COM ports, Usbser.sys, which is referred to by the provided inf file included with the demo, the inf file can be found at: C:\Program Files\Freescale\Freescale MQX 3.x\usb\device\examples\cdc\virtual_com\inf
(notice the x in 3.x for the MQX sub-version you are using)



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, whic paths and removable media. The best driver found will be installed.

□ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

C:\Program Files\Freescale\Freescale MQX 3.5\usb\

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows do the driver you choose will be the best match for your hardware.

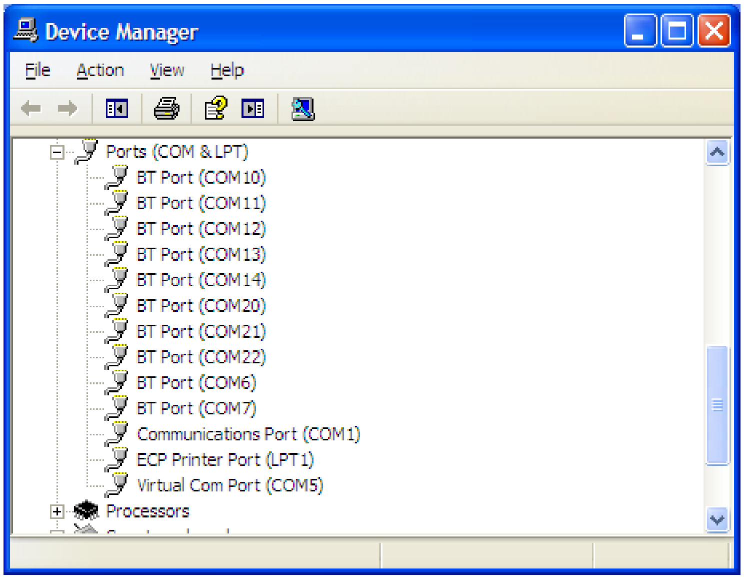
Browse For Folder

Select the folder that contains drivers for your hardware.

```
⊟ device
  ⊞ build
  ⊟ examples
    ⊟ cdc
      ⊟ virtual_com
        ⊞ cwcf71
          cwcf72
          inf
```

To view any subfolders, click a plus sign above.

Once the inf file is provided, the computer relates all information regarding the application like vendor ID (VID) product ID (PID) or text strings to this particular device, load the driver and a new COM port appears in the Windows device manager.

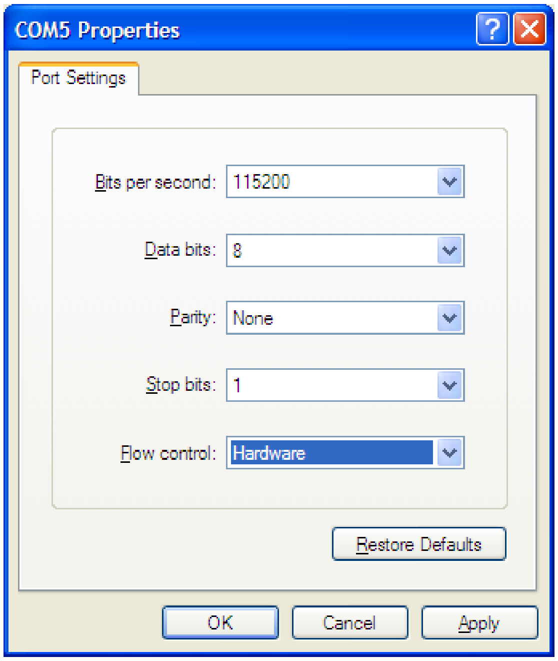


Device Manager

```
⊟ Ports (COM & LPT)
    BT Port (COM10)
    BT Port (COM11)
    BT Port (COM12)
    BT Port (COM13)
    BT Port (COM14)
    BT Port (COM20)
    BT Port (COM21)
    BT Port (COM22)
    BT Port (COM6)
    BT Port (COM7)
    Communications Port (COM1)
    ECP Printer Port (LPT1)
    Virtual Com Port (COM5)
⊞ Processors
```

Notice that the COM port number is not always 5, it varies from computer to computer.

With the driver installed, the demo can be run. As explained earlier, the demo is quite simple, all that is needed is to open HyperTerminal or any other terminal program, and select a configuration for the COM connection. Notice that CDC is capable of emulating any configuration as it is not really a physical UART, so in this example it does not really matter how the connection is configured, the example will work correctly with any settings. However for applications where the connection will be routed to a physical device or where the configuration is needed for backwards compatibility it might be necessary to use specific configuration. CDC uses control transfers to communicate this information to the device, so it is possible for the device to act accordingly.



Bits per second: 115200

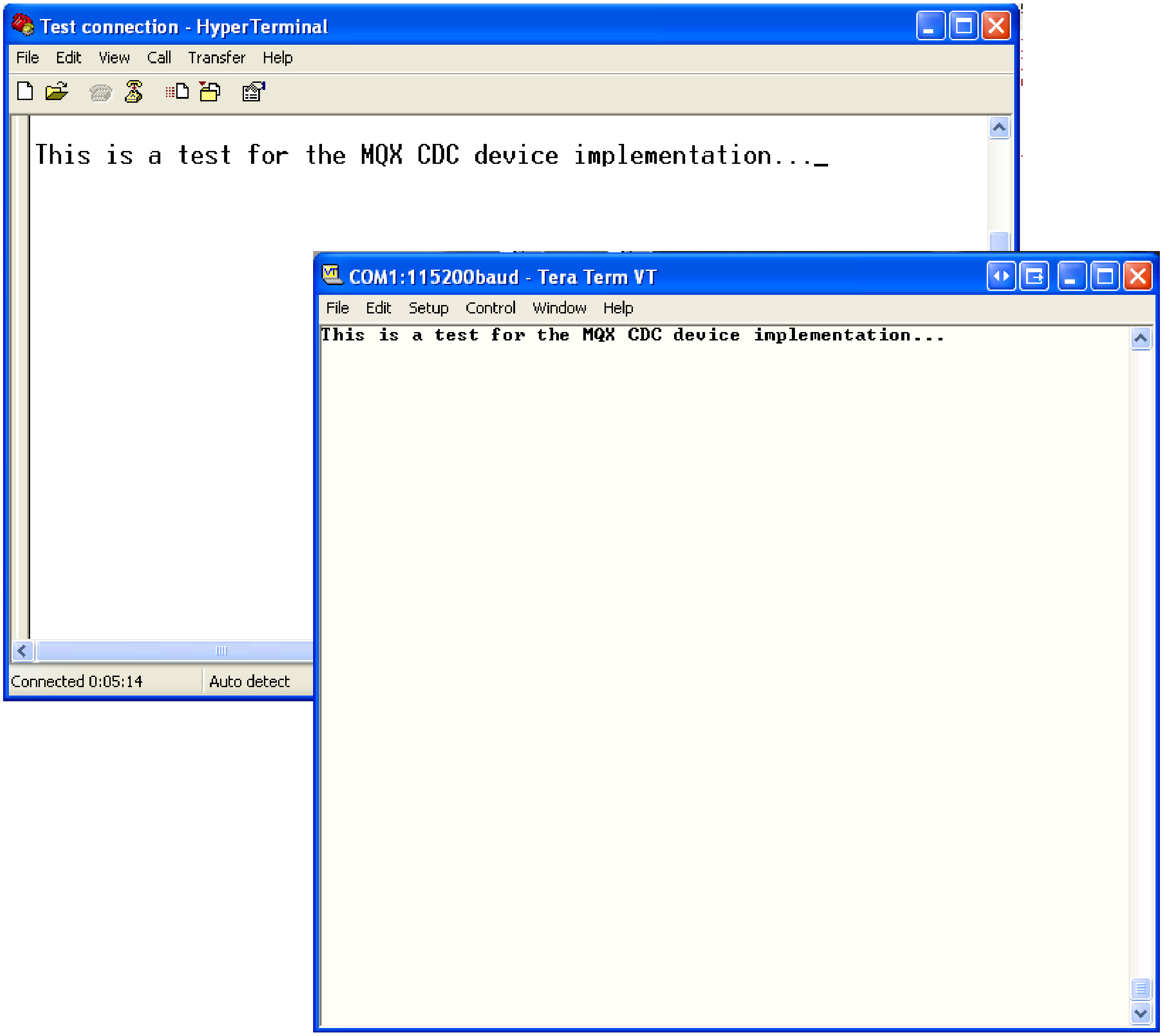
Data bits: 8

Parity: None

Stop bits: 1

Flow control: Hardware

Once the connection is open, make sure automatic echo is not configured (it is not configured by default in Windows HyperTerminal) and simply type some characters. You will see what you type on the screen because the application is echoing the received data back.

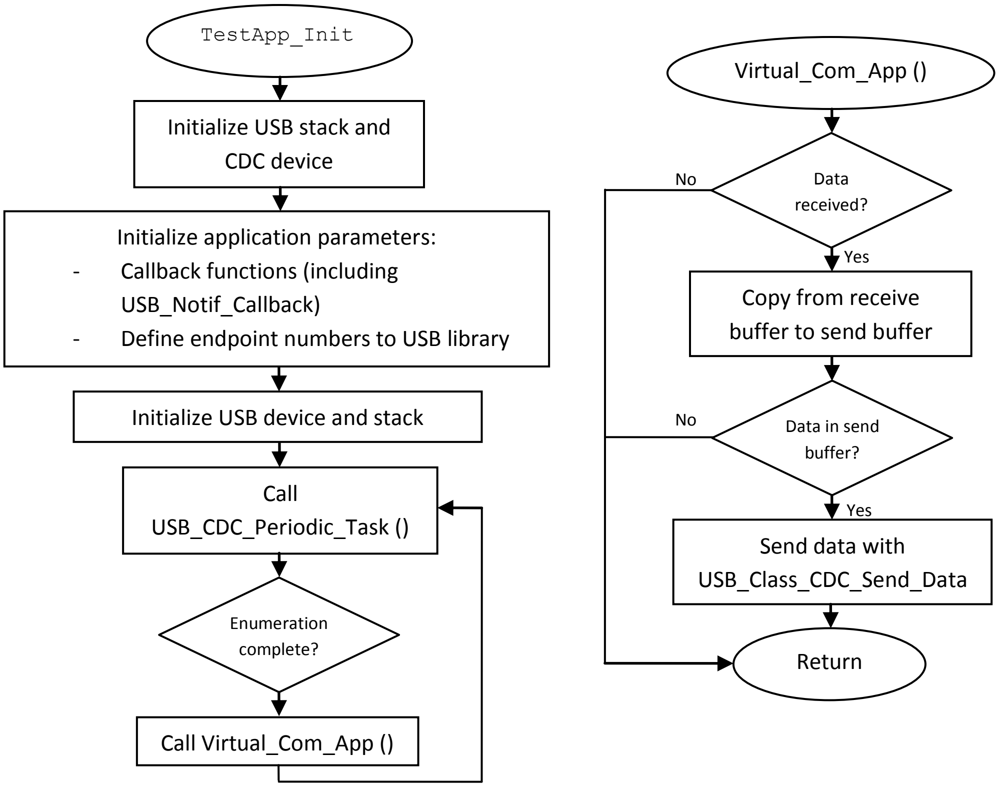

Test connection - HyperTerminal
File Edit View Call Transfer Help
Connected 0:05:14
Auto detect
COM1:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
This is a test for the MQX CDC device implementation..._
```

```
This is a test for the MQX CDC device implementation...
```

Explanation of the example

The application demo creates only one main task. The flow of the task is described in the next figure.


TestApp_Init
Initialize USB stack and CDC device
Initialize application parameters:
- Callback functions (including USB_Notif_Callback)
- Define endpoint numbers to USB library
Initialize USB device and stack
Call USB_CDC_Periodic_Task ()
Enumeration complete?
Call Virtual_Com_App ()
Virtual_Com_App ()
Data received?
No
Yes
Copy from receive buffer to send buffer
Data in send buffer?
No
Yes
Send data with USB_Class_CDC_Send_Data
Return

The main task only calls the TestApp_Init function. The TestApp_Init function initializes all the application parameters including data sizes, number and size of endpoints and USB application callbacks. At last, it initializes the USB interface.

After the initialization it enters an infinite loop that calls two functions. The USB_CDC_Periodic_Task is called in each pass through the loop and it is a necessary call for the CDC class to operate correctly. It basically refreshes status values for the USB stack.

The Virtual_Com_App is also called inside the infinite loop, but only after checking that the USB device is properly enumerated, as this function is designed to actually do the user operations, so it should only run when the communication is established. As shown on the flow chart above, the Virtual_Com_App function does just simple echoing of data. It checks g_recv_size variable, which is updated in the USB_Notif_Callback (explained below). If this variable indicates that data has been received it copies the data from receive buffer to the send buffer, resets the receive data counter variable and sets the send data counter variable to the size of the echo package. A second condition tests if the send counter (g_send_size) indicates data to be sent and if it does, the send buffer is transmitted using USB_Class_CDC_Send_Data function, which is part of the CDC library.

The USB_Notif_Callback, which is part of the callbacks initialized by TestApp_Init is a callback function that the CDC class code uses to signal to the user that a CDC event has happened. The four conditions it handles are DTE activation and deactivation (for compatibility with hardware flow control as in traditional UARTs), data received, and send complete. The DTE conditions set and clear the start_transactions variable which is tested in receive and send condition checks to make sure data can be sent or received. The receive condition, checked through the USB_APP_DATA_RECEIVED macro, copies the received data from the USB buffer to the user g_curr_recv_buf buffer. The send complete event (checked with the USB_APP_SEND_COMPLETE macro) is just a stub to add user code if applicable.