Future Technology Devices International Ltd.

Application Note

AN_171

Vinculum-II USB Host Using the CDC Driver

Document Reference No.: FT_000413

Version 1.0

Issue Date: 2011-02-17

This application note provides an example of how to use the FTDI Vinculum-II (VNC2) USB Host CDC driver. Sample source code is included.

Future Technology Devices International Limited (FTDI)

Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
E-Mail (Support): support1@ftdichip.com Web: http://www.ftdichip.com

Table of Contents

1 Introduction

VNC2 USB Host capability has been extended with the development of a USB Communications Device Class[1] (CDC) driver which will be supported in the Vinculum II Development Tools from v1.4.0 onwards. This application note provides a guide to using the CDC driver. It contains a description of an application that connects to a CDC device present on USB Host 1 and establishes a connection between the data interface on the CDC device and the UART on the VNC2.[2]

It has only been tested on a Samsung SGH-E900 mobile phone, but the general principles apply to any CDC device.

The sample source code contained in this application note is provided as an example and is neither guaranteed nor supported by FTDI.

[1] USB Communications Device Class is a standard class defined by USB-IF.

[2] While the CDC-UART interface is the subject of this application note, a CDC device can connect with any other suitable peripheral interface on VNC2.

2 USB Host CDC Driver Basics

In the driver hierarchy, the CDC driver is layered on top of the USB Host driver, as shown in Figure 1.

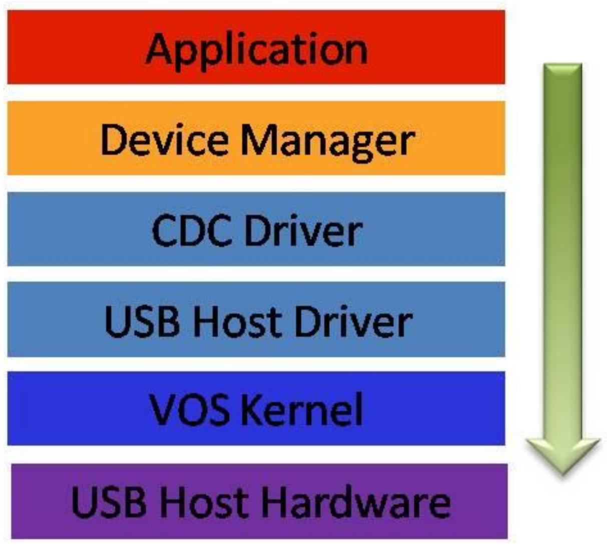

Figure 1: Driver Hierarchy

A layered driver has the standard device driver format that includes functions for *init()*, *open()*, *close()*, *read()*, *write()* and *ioctl()*. For further details, see [1].

2.1 Initialisation

Before it can be used, the CDC driver must be initialized. An application should call the function *usbHostCDC_init()* before the kernel scheduler is started with *vos_start_scheduler()*.

```
unsigned char usbHostCDC_init(unsigned char vos_dev_num);
```

2.1.1 usbHostCDC_init

Syntax

```
unsigned char usbHostCDC_init(unsigned char devNum)
```

Description

Initialise the CDC driver, and register the driver with the Device Manager.

Parameters

The device number to use when registering the driver with the Device Manager is passed in the *devNum* parameter.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The driver can be attached to a CDC device once the USB Host enumeration has completed. As a result of this call, the driver is registered with the VOS Device Manager, and all subsequent accesses to the CDC driver are through the standard the Device Manager interface.

2.2 Operation

A layered driver works in the same manner as a standard driver, and device accesses are performed using the standard Device Manager API functions. Before it can be accessed, a device must be opened with *vos_dev_open()*, and a handle obtained. The handle is used in all subsequent device access operations: reading and writing the device are performed with *vos_dev_read()* and *vos_dev_write()* respectively; device control is performed with *vos_dev_ioctl()*; and the device is closed with *vos_dev_close()*.

2.3 Control

The CDC driver uses the common IOCTL structure *common_ioctl_cb_t*, so it accepts the same IOCTL request codes as the UART driver; these requests are listed in UART IOCTL Calls, see [1]. In addition, the CDC driver has its own specific set of control functions, and it supports the following IOCTL request codes shown in Table 1:

IOCTL	Function
VOS_IOCTL_USBHOSTCDC_ATTACH	Attach the driver to a USB interface device.
VOS_IOCTL_USBHOSTCDC_DETACH	Detach the driver from the USB interface device.
VOS_IOCTL_USBHOSTCDC_START_POLL	Start polling for data from the device.
VOS_IOCTL_USBHOSTCDC_STOP_POLL	Stop polling for data from the device.
VOS_IOCTL_USBHOSTCDC_GET_LINE_CODING	Get line coding.
VOS_IOCTL_USBHOSTCDC_SET_LINE_CODING	Set line coding.
VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE	Set line control state.

Table 1: CDC IOCTL Request Codes

These IOCTL requests are described in the following sections.

2.3.1 VOS_IOCTL_USBHOSTCDC_ATTACH

Description

A layered driver implements an *attach()* function that is used to establish a connection to its underlying driver. In the CDC driver, this function is implemented as an *ioctl* request, *VOS_IOCTL_USBHOSTCDC_ATTACH*.

This function attaches the CDC driver to an interface in the USB Host controller. The host controller handle is that obtained from *vos_dev_open()* when the USBHost driver was opened. This function does not check the VID and PID or the class, subclass and protocol values of the device, since these are configurable.

Parameters

The device interface handles for the control and data interfaces and USB Host controller handle must be written in a *usbHostCDC_ioctl_cb_attach_t* structure that is passed in the *set.data* member of the *common_ioctl_cb_t* structure. The *usbHostCDC_ioctl_cb_attach_t* structure is defined as follows:

```
typedef struct _usbHostCDC_ioctl_cb_attach_t
{
        VOS_HANDLE hc_handle;
        usbhost_device_handle *ifCtrl;
        usbhost_device_handle *ifData;
} usbHostCDC_ioctl_cb_attach_t;
```

Returns

If the attach is successful, then this function returns *USBHOSTCDC_OK*. If the parameters passed to the function are incorrect, then this function returns *USBHOSTCDC_INVALID_PARAMETER*. If the interface does not have control and a bulk IN and bulk OUT endpoint, then this function returns *USBHOSTCDC_NOT_FOUND*.

Example

See section 4.6 for an example of the use of *VOS_IOCTL_USBHOSTCDC_ATTACH*.

2.3.2 VOS_IOCTL_USBHOSTCDC_DETACH

Description

A layered driver implements a *detach()* function that is used by an instance of its device to close down a connection previously established with *attach()*. In the CDC driver, this function is implemented as an *ioctl* request, *VOS_IOCTL_USBHOSTCDC_DETACH*.

Parameters

This function takes no parameters.

Returns

This function always returns *USBHOSTCDC_OK*.

2.3.3 VOS_IOCTL_USBHOSTCDC_START_POLL

Description

This function signals the driver to start polling the attached device.

Parameters

This function takes no parameters.

Returns

This function always returns *USBHOSTCDC_OK*.

2.3.4 VOS_IOCTL_USBHOSTCDC_STOP_POLL

Description

This function signals the driver to stop polling the attached device.

Parameters

This function takes no parameters.

Returns

This function always returns *USBHOSTCDC_OK*.

2.3.5 VOS_IOCTL_USBHOSTCDC_GET_LINE_CODING

Description

This function returns the currently configured line coding. Line coding consists of line-character formatting properties such as DTE rate, number of data bits, number of stop bits and parity type. For further details, see section 6.3.11 in [4].

Parameters

The address of a *usbhostcdc_line_coding_t* structure is written to the *get.data* field passed in the *common_ioctl_cb_t* structure. The *usbhostcdc_line_coding_t* structure is defined as follows:

```
typedef struct _usbHostCDC_line_coding_t {
        unsigned long dwDTERate;   // Data terminal rate, in bits per second
        unsigned char bCharFormat; // 0 – 1 stop bit
                                   // 1 – 1.5 stop bits
                                   // 2 – 2 stop bits
        unsigned char bParityType; // 0 – None
                                   // 1 – Odd
                                   // 2 – Even
                                   // 3 – Mark
                                   // 4 – Space
        unsigned char bDataBits;   // Data bits (5,6,7,8 or 16)
} usbHostCDC_line_coding_t;
```

Returns

If successful, this function returns *USBHOSTCDC_OK*. Otherwise, a USB Host error code is returned.

Example

See section 4.7 for an example of the use of *VOS_IOCTL_USBHOSTCDC_GET_LINE_CODING*.

2.3.6 VOS_IOCTL_USBHOSTCDC_SET_LINE_CODING

Description

This function sets asynchronous line-character formatting properties such as DTE rate, number of data bits, number of stop bits and parity type. For further details, see section 6.3.10 in [4].

Parameters

A *usbhostcdc_line_coding_t* structure is filled-in and its address is written to the *set.data* field passed in the *common_ioctl_cb_t* structure. The *usbhostcdc_line_coding_t* structure is defined in 2.3.5.

Returns

If successful, this function returns *USBHOSTCDC_OK*. Otherwise, a USB Host error code is returned.

Example

See section 4.7 for an example of the use of *VOS_IOCTL_USBHOSTCDC_SET_LINE_CODING*.

2.3.7 VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE

Description

This function generates the RS232 control signals. For further details, see section 6.3.12 in [4].

Parameters

The address of an *unsigned short* variable that contains the values of the RS232 control signals is written to the *set.data* field in the *common_ioctl_cb_t* structure. Control signal bits are defined as follows:

```
#define USBHOSTCDC_DTE_PRESENT          1
#define USBHOSTCDC_ACTIVATE_CARRIER     2
```

Returns

If successful, this function returns *USBHOSTCDC_OK*. Otherwise, a USB Host error code is returned.

Example

See section 4.7 for an example of the use of *VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE*.

3 Device Requirements

The CDC driver supports a particular subset of the USB Communications Device Class. In particular, the device should have 2 interfaces, one for control and one for data.

3.1 Control Interface

The first interface should be the control interface: Class 2, Subclass 2, Protocol 1.

The control interface should contain 1 endpoint that is of Interrupt IN type.

3.2 Data Interface

The second interface should be the data interface: Class 10, Subclass 0, Protocol 0.

The data interface should contain 2 endpoints of type BULK, one for IN and one for OUT data.

4 Developing an Application

This section describes the development of an application that uses the CDC driver. The application connects to a CDC device present on USB Host 1 and establishes a connection between the data interface on the CDC device and the UART on the VNC2.

The full project including source files is available in the Samples directory in the Vinculum II Development Tools release, version 1.4.0 and later.

The application has only been tested on a Samsung SGH-E900 mobile phone, but the general principles outlined in this section apply to application development for any CDC device. Some mobile phones will require specific setup in order to appear as a CDC device.

4.1 Open the USB Host Port

This application expects a CDC device to be plugged into USB Host port 1, so first it opens the port.

```
#define VOS_DEV_USB_HOST1    1      // USB Host port 1 device number
#define VOS_DEV_USB_HOST2    2      // USB Host port 2 device number (not used)

VOS_HANDLE hUsb1;

hUsb1 = vos_dev_open(VOS_DEV_USB_HOST1);
if (hUsb1 == 0xffff)
      ; // error
```

4.2 Open the UART

This application establishes a connection between the phone and the UART, so it opens the UART.

```
#define VOS_DEV_UART         3      // UART device number

VOS_HANDLE hUart;

hUart = vos_dev_open(VOS_DEV_UART);
if (hUart == 0xffff)
      ; // error
```

4.3 Wait for Enumeration

The application waits for a CDC device to be plugged into USB Host port 1. The USB Host request, *VOS_IOCTL_USBHOST_GET_CONNECT_STATE*, is called to interrogate the state of the port.

```
usbhost_ioctl_cb_t hc_iocb;
unsigned char i;

do
{
       // wait for enumeration to complete
       vos_delay_msecs(500);

       // user ioctl to see if bus available
       hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
       hc_iocb.get = &i;
       vos_dev_ioctl(hUsb1, &hc_iocb);
}
while (i != PORT_STATE_ENUMERATED);
```

4.4 Find the Phone

The application establishes if the device attached to USB Host port 1 is a CDC device and determines if it is a member of the CDC subset supported by the CDC driver (as defined in section 3). The following code shows the sequence of USB Host requests that is necessary to accomplish this. For clarity, error checking has been removed.

```
usbhost_ioctl_cb_class_t hc_iocb_class;
usbhost_device_handle *ifDev1;
usbhost_device_handle *ifDev2;

// find mobile phone
hc_iocb_class.dev_class = USB_CLASS_CDC_CONTROL;
hc_iocb_class.dev_subclass = USB_SUBCLASS_CDC_CONTROL_ABSTRACT;
hc_iocb_class.dev_protocol = 1;

// user ioctl to find first CDC control device
hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
hc_iocb.handle.dif = NULL;
hc_iocb.set = &hc_iocb_class;
hc_iocb.get = &ifDev1;
vos_dev_ioctl(hUsb1, &hc_iocb);

// user ioctl to find first next device
// by definition it must be the interface following the CDC control device
hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;
hc_iocb.handle.dif = ifDev1;
hc_iocb.get = &ifDev2;
vos_dev_ioctl(hUsb1, &hc_iocb);

// Check that the found device is indeed a data device
hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO;
hc_iocb.handle.dif = ifDev2;
hc_iocb.get = &hc_iocb_class;
vos_dev_ioctl(hUsb1, &hc_iocb);

// verify the USB class information for the data class
if ((hc_iocb_class.dev_class == USB_CLASS_CDC_DATA) &&
       (hc_iocb_class.dev_subclass == 0) &&
       (hc_iocb_class.dev_protocol == 0))
{
       // found the phone!
}
```

4.5 Open the CDC Driver

The application has found a suitable phone connected to USB Host port 1. Now open the CDC driver.

```
#define VOS_DEV_CDC   4      // CDC device number

VOS_HANDLE hCDC;

hCDC = vos_dev_open(VOS_DEV_CDC);
if (hCDC == 0xffff)
      ; // error
```

4.6 Attach the CDC Driver to the USB Host Driver

The application calls the *VOS_IOCTL_USBHOSTCDC_ATTACH* function to establish a connection between the CDC driver and the USB Host driver. It fills in a *usbHostCDC_ioctl_cb_attach_t* structure with the handle of USB Host port 1, and the device interface handles for the control and data interfaces.

```
common_ioctl_cb_t CDC_iocb;
usbHostCDC_ioctl_cb_attach_t CDC_att;

// Fill in CDC attach structure
CDC_att.hc_handle = hUsb1;
CDC_att.ifCtrl = ifDev1;
CDC_att.ifData = ifDev2;

CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_ATTACH;
CDC_iocb.set.data = &CDC_att;

if (vos_dev_ioctl(hCDC, &CDC_iocb) != USBHOSTCDC_OK)
        ; // error
```

4.7 Configure the Phone

Having attached the CDC driver to the USB Host port, the application configures the communications settings on the phone. It does this by sending *VOS_IOCTL_USBHOSTCDC_GET_LINE_CODING*, *VOS_IOCTL_USBHOSTCDC_SET_LINE_CODING* and *VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE* requests to the CDC driver. The configuration sequence depends on the type of phone. The following code fragments show the sequence of requests necessary to configure the Samsung SGH-E900 mobile phone. Error checking has been removed for clarity.

```
usbHostCDC_line_coding_t CDC_line;
unsigned short lineStatus;

// Obtain the line coding information for the CDC device
CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_GET_LINE_CODING;
CDC_iocb.get.data = &CDC_line;
vos_dev_ioctl(hCDC, &CDC_iocb);

// Turn off DTE present and Carrier Active signal
lineStatus = 0;
CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE;
CDC_iocb.set.data = &lineStatus;
vos_dev_ioctl(hCDC, &CDC_iocb);

// Turn on the DTE present and Carrier Active signals
lineStatus = (USBHOSTCDC_DTE_PRESENT | USBHOSTCDC_ACTIVATE_CARRIER);
CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE;
CDC_iocb.set.data = &lineStatus;
vos_dev_ioctl(hCDC, &CDC_iocb);

// Set the line coding information back to the previous setting
CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_SET_LINE_CODING;
CDC_iocb.set.data = &CDC_line;
vos_dev_ioctl(hCDC, &CDC_iocb);

// Turn on the DTE present and Carrier Active signals
lineStatus = (USBHOSTCDC_DTE_PRESENT | USBHOSTCDC_ACTIVATE_CARRIER);
CDC_iocb.ioctl_code = VOS_IOCTL_USBHOSTCDC_SET_LINE_CONTROL_STATE;
CDC_iocb.set.data = &lineStatus;
vos_dev_ioctl(hCDC, &CDC_iocb);
```

4.8 Read from the Phone

In this application, data read from the phone is transferred to the UART. This functionality is implemented in a dedicated thread, and uses the Device Manager interface functions *vos_dev_read()* to read the phone, and *vos_dev_write()* to write to the UART. This is illustrated in the following code fragments.

```
void CDC2UART(void)
{
        unsigned char status;
        char buffer[16];

        // perform steps 3.1 - 3.7
        // signal other threads that initialization sequence is

        do {
                // read from the CDC device
                status = vos_dev_read(hCDC, buffer, 1, NULL);
                if (status != USBHOSTCDC_OK) {
                        // read fatal error, so break
                        break;
                }

                // then write to UART
                status = vos_dev_write(hUart, buffer, 1, NULL);
                if (status != UART_OK) {
                        // fatal error, so break
                        break;
                }
         } while (1);
}
```

CDC2UART is a thread created in *main()* by calling *vos_create_thread()*. CDC2UART is responsible for initializing the system, and when this is completed it enters its forever-loop of reading from the phone and writing to the UART. This example demonstrates reading one byte from the phone and passing it on to the UART.

4.9 Write to the Phone

In this application, data received on the UART interface is written to the phone. This functionality is implemented in a dedicated thread, and uses the Device Manager interface functions *vos_dev_read()* to read the UART, and *vos_dev_write()* to write to the phone. This is illustrated in the following code fragments.

```
void UART2CDC(void)
{
        unsigned char status;
        char buffer[16];

        // wait for other thread to complete the initialization sequence

        do {
                // read from the UART
                status = vos_dev_read(hUart, buffer, 1, NULL);
                if (status != UART_OK) {
                        // fatal error, so break
                        break;
                }

                // then write to CDC device
                status = vos_dev_write(hCDC, buffer, 1, NULL);
                if (status != USBHOSTCDC_OK) {
                        // fatal error, so break
```

```
                                break;
                        }
                } while (1);
        }
```

UART2CDC is a thread created in *main()* by calling *vos_create_thread()*. *hUart* is the handle of the UART, obtained when the UART was opened in the CDC2UART thread. This example demonstrates reading one byte from the UART and passing it on to the phone via the CDC driver.

5 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales)	sales1@ftdichip.com
E-mail (Support)	support1@ftdichip.com
E-mail (General Enquiries)	admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com
Web Shop URL	http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales)	us.sales@ftdichip.com
E-Mail (Support)	us.support@ftdichip.com
E-Mail (General Enquiries)	us.admin@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales)	cn.sales@ftdichip.com
E-mail (Support)	cn.support@ftdichip.com
E-mail (General Enquiries)	cn.admin@ftdichip.com
Web Site URL	http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

6 Appendix A – References

Document References

[1] FTDI Application Note AN_151, *Vinculum II User Guide*, FTDI, 2010. Available from http://www.ftdichip.com/Support/Documents/AppNotes.htm

[2] *Universal Serial Bus Specification Revision 2.0*, USB Implementers Forum, 2000. Available from http://www.usb.org/developers/docs/

[3] *Universal Serial Bus Class Definitions for Communications Devices Revision 1.2*, USB Implementers Forum, 2007. Available from http://www.usb.org/developers/devclass_docs#approved

[4] *Universal Serial Bus Communications Class Subclass Specification for PSTN Devices Revision 1.2*, USB Implementers Forum, 2007. Available from http://www.usb.org/developers/devclass_docs#approved

Acronyms and Abbreviations

Terms	Description
CDC	USB Communications Device Class
VNC2	Vinculum II
VOS	Vinculum Operating System
DTE	Data Terminal Equipment

7 Appendix B – Revision History

Revision	Changes	Date
1.0	Initial Release	2011-02-17