NXP Semiconductors
Application Note

Document Number: AN12023
Rev. 0, 08/2017

Customizing a Composite Device Based on SDK2.2

1. Introduction

Composite devices are supported by the SDK2.2. In the application, users usually customize the special combinations. This application note shows how to add an HID device to the *usb_device_composite_cdc_msc* demo. If some other combination in a composite device is necessary, some of the rules and considerations remain the same. This example uses the FRDM-K64F board and IAR Embedded Workbench® 8.0 IDE.

Contents

2. Background knowledge

For the background knowledge about the USB protocol, endpoint, and interface, see Figure 1.

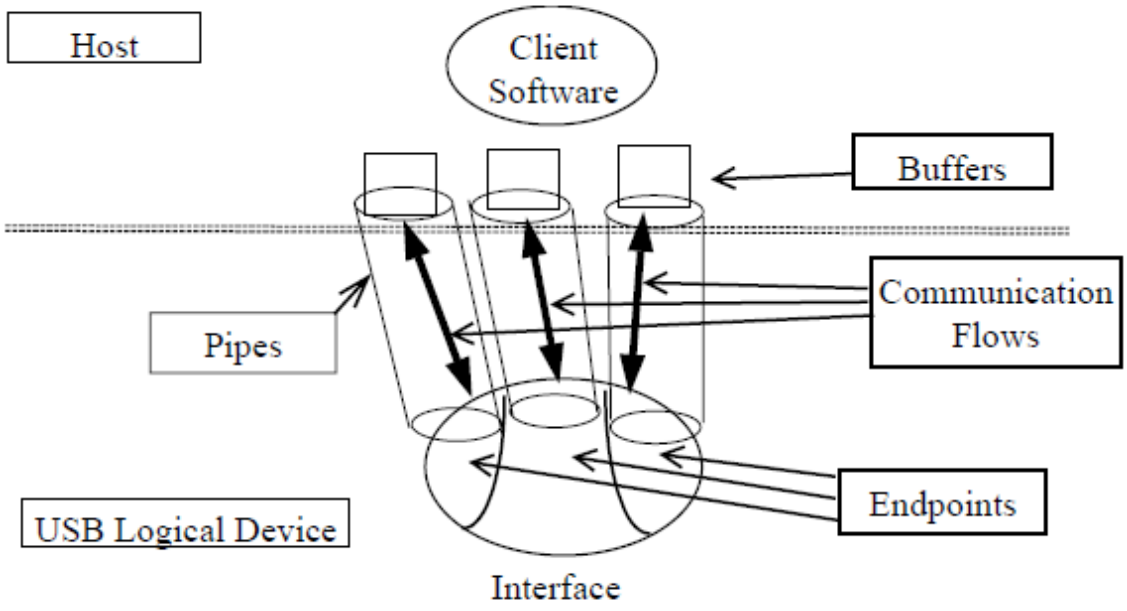

Figure 1. Endpoint and interface

The application has one (or more) device classes, the device class has one (or more) interfaces, and the interface has one (or more) endpoints. When making a composite device, design and assign the interface/endpoint for each device class correctly.

3. Steps

The key steps for adding the HID device into the *usb_device_composite_cdc_msc* demo are listed here. It is supposed that you have the SDK2.2 package already downloaded and unzipped. Open the project at *boards\frdmk64f\usb_examples\usb_device_composite_cdc_msc\bm\iar\dev_composite_cdc_msc_bm.e* to continue.

3.1. Step 1: Adding files

Copy the files listed in Table 1 from *boards\frdmk64f\usb_examples\usb_device_composite_hid_mouse_hid_keyboard\bm* to *boards\frdmk64f\usb_examples\usb_device_composite_cdc_msc\bm*.

Table 1. Files to import

Index	File name
1	hid_mouse.c
2	hid_mouse.h
3	usb_device_hid.c
4	usb_device_hid.h

Then add the files to the project, as shown in Figure 2.

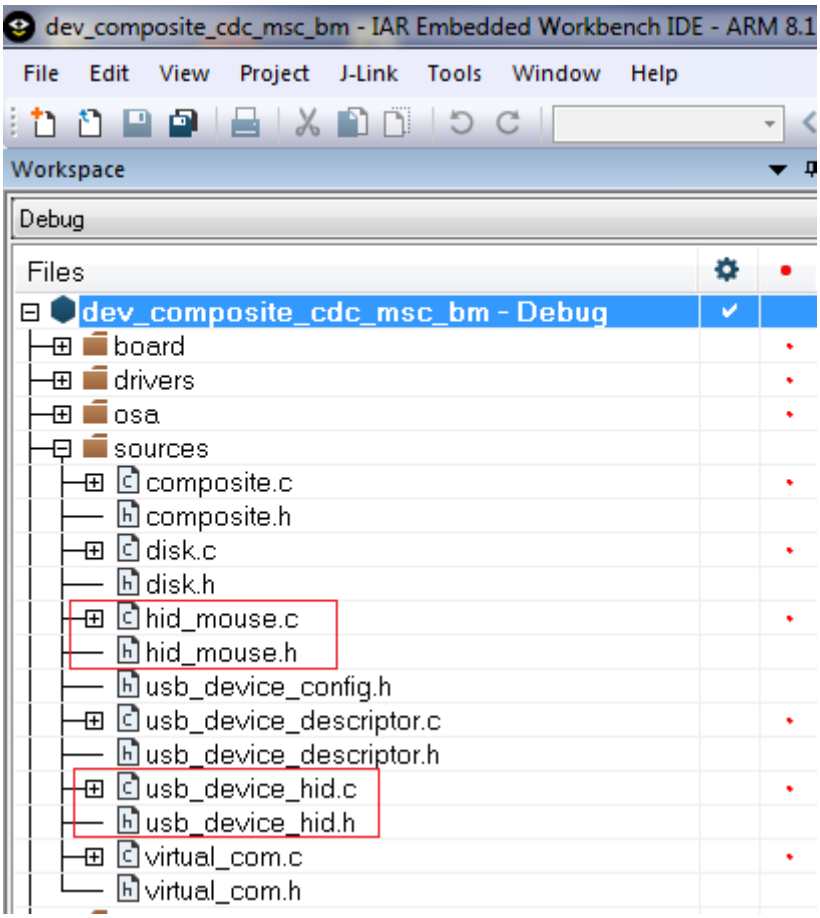

Figure 2. Adding files

3.2. Step 2: Modifying related files

3.2.1. Updating *composite.c*

The device class data structure is organized in arrays. When adding a class, add the corresponding content into the arrays and make sure to update the array length correctly.

Add the code marked in red:

```
static usb_device_composite_struct_t g_composite;
extern usb_device_class_struct_t g_UsbDeviceCdcVcomConfig;
extern usb_device_class_struct_t g_mscDiskClass;
```

```
extern usb_device_class_struct_t g_UsbDeviceHidMouseClass;

usb_device_class_config_struct_t g_compositeDevice[2+1] = {
    {
        USB_DeviceCdcVcomCallback, (class_handle_t)NULL, &g_UsbDeviceCdcVcomConfig,
    },
    {
        USB_DeviceMscCallback, (class_handle_t)NULL, &g_mscDiskClass,
    },
    {
        USB_DeviceHidMouseCallback, (class_handle_t)NULL, &g_UsbDeviceHidMouseClass,
    }
};

/* USB device class configuration information */
usb_device_class_config_list_struct_t g_compositeDeviceConfigList = {
    g_compositeDevice,
    USB_DeviceCallback,
    2+1,
};
```

To update the code in the callback, see the example code of the class you want to add:

```
usb_status_t USB_DeviceCallback(usb_device_handle handle, uint32_t event, void *param)
{
... ...
        case kUSB_DeviceEventSetConfiguration:
            if (param)
            {
                g_composite.attach = 1;
                g_composite.currentConfiguration = *temp8;
                USB_DeviceCdcVcomSetConfigure(g_composite.cdcVcom.cdcAcmHandle, *temp8);
                USB_DeviceMscDiskSetConfigure(g_composite.mscDisk.mscHandle, *temp8);
                USB_DeviceHidMouseSetConfigure(g_composite. hidMouseHandle, *temp8);
                error = kStatus_USB_Success;
            }
            break;
        case kUSB_DeviceEventGetHidReportDescriptor:
            if (param)
            {
                error =
                    USB_DeviceGetHidReportDescriptor(handle,
(usb_device_get_hid_report_descriptor_struct_t *)param);
            }
            break;
... ...
}
```

Update the code for the initialization:

```
void APPInit(void)
{
... ...
    g_composite.speed = USB_SPEED_FULL;
    g_composite.attach = 0;
    g_composite.cdcVcom.cdcAcmHandle = (class_handle_t)NULL;
    g_composite.mscDisk.mscHandle = (class_handle_t)NULL;
    g_composite. hidMouseHandle = (class_handle_t)NULL;
    g_composite.deviceHandle = NULL;
```

```
    if (kStatus_USB_Success !=
        USB_DeviceClassInit(CONTROLLER_ID, &g_compositeDeviceConfigList,
&g_composite.deviceHandle))
    {
        usb_echo("USB device composite demo init failed\r\n");
        return;
    }
    else
    {
        usb_echo("USB device composite demo\r\n");
        g_composite.cdcVcom.cdcAcmHandle = g_compositeDeviceConfigList.config[0].classHandle;
        g_composite.mscDisk.mscHandle    = g_compositeDeviceConfigList.config[1].classHandle;
        g_composite. hidMouseHandle   = g_compositeDeviceConfigList.config[2].classHandle;

        USB_DeviceCdcVcomInit(&g_composite);
        USB_DeviceMscDiskInit(&g_composite);
        USB_DeviceHidMouseInit(&g_composite);
    }
... ...
}
```

3.2.2. Updating *composite.h*

```
#include "virtual_com.h"
#include "disk.h"
#include "hid_mouse.h"

/*******************************************************************************
* Definitions
******************************************************************************/
#if defined(USB_DEVICE_CONFIG_EHCI) && (USB_DEVICE_CONFIG_EHCI > 0)
#define CONTROLLER_ID kUSB_ControllerEhci0

#define USB_DEVICE_INTERRUPT_PRIORITY (3U)
```

Add a new member here:

```
typedef struct _usb_device_composite_struct
{
    usb_device_handle deviceHandle; /* USB device handle. */
    usb_cdc_vcom_struct_t  cdcVcom;  /* CDC virtual com device structure. */
    usb_msc_struct_t       mscDisk; /* MSC disk device structure.  */
    class_handle_t hidMouseHandle;
    uint8_t speed;            /* Speed of the USB device. */
    uint8_t attach;           /* A flag to indicate whether a USB device is attached. */
    uint8_t currentConfiguration; /* Current configuration value. */
    uint8_t currentInterfaceAlternateSetting[USB_INTERFACE_COUNT];
} usb_device_composite_struct_t;
......
usb_status_t USB_DeviceMscCallback(class_handle_t handle, uint32_t event, void *param);
extern usb_status_t USB_DeviceHidMouseCallback(class_handle_t handle, uint32_t event, void
*param);
extern usb_status_t USB_DeviceHidMouseInit(usb_device_composite_struct_t *device_composite);
extern usb_status_t USB_DeviceHidMouseSetConfigure(class_handle_t handle, uint8_t configure);
```

3.2.3. Updating *usb_device_config.h*

USB_DEVICE_CONFIG_HID means the count of the HID devices, not enabling the HID. When adding two HID devices, set it to 2. You must know how many endpoints are used in the new device class.

```
......
#define USB_DEVICE_CONFIG_HID (1U)
......
#define USB_DEVICE_CONFIG_ENDPOINTS (5+1)
......
```

3.2.4. Updating *usb_device_descriptor.c*

```
......
/* Define class information for MSC disk */
usb_device_class_struct_t g_mscDiskClass = {
    g_mscDiskInterfaceList, kUSB_DeviceClassTypeMsc, USB_DEVICE_CONFIGURATION_COUNT,
};
```

Add the data structure for the new device class:

```
/* HID */
usb_device_endpoint_struct_t g_UsbDeviceHidEndpoints[USB_HID_MOUSE_ENDPOINT_COUNT] = {
    {
        /* HID mouse interrupt IN pipe */
        USB_HID_MOUSE_ENDPOINT | (USB_IN << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT),
        USB_ENDPOINT_INTERRUPT,
        FS_INTERRUPT_IN_PACKET_SIZE,
    },
};
usb_device_interface_struct_t g_UsbDeviceHidInterface[] = {{
    0U, /* The alternate setting of the interface */
    {
        USB_HID_MOUSE_ENDPOINT_COUNT, /* Endpoint count */
        g_UsbDeviceHidEndpoints,      /* Endpoints handle */
    },
    NULL,
}
};
usb_device_interfaces_struct_t g_UsbDeviceHidInterfaces[USB_HID_MOUSE_INTERFACE_COUNT] = {{
    USB_HID_MOUSE_CLASS,           /* HID mouse class code */
    USB_HID_MOUSE_SUBCLASS,        /* HID mouse subclass code */
    USB_HID_MOUSE_PROTOCOL,        /* HID mouse protocol code */
    USB_HID_MOUSE_INTERFACE_INDEX, /* The interface number of the HID mouse */
    g_UsbDeviceHidInterface,       /* Interfaces handle */
    sizeof(g_UsbDeviceHidInterface) / sizeof(usb_device_interfaces_struct_t),
}
};
usb_device_interface_list_t g_UsbDeviceHidInterfaceList[USB_DEVICE_CONFIGURATION_COUNT] = {
    {
        USB_HID_MOUSE_INTERFACE_COUNT, /* The interface count of the HID mouse */
        g_UsbDeviceHidInterfaces,      /* The interfaces handle */
    },
};

usb_device_class_struct_t g_UsbDeviceHidMouseClass = {
    g_UsbDeviceHidInterfaceList,    /* The interface list of the HID mouse */
```

```
    kUSB_DeviceClassTypeHid,        /* The HID class type */
    USB_DEVICE_CONFIGURATION_COUNT, /* The configuration count */
};
```

This is a key step. Add the descriptor for the new device class correctly. Implement more than one device class by adding the corresponding descriptor here:

```
......
uint8_t g_UsbDeviceConfigurationDescriptor[USB_DESCRIPTOR_LENGTH_CONFIGURATION_ALL] = {
    ......
    /*Bulk OUT Endpoint descriptor */
    USB_DESCRIPTOR_LENGTH_ENDPOINT, USB_DESCRIPTOR_TYPE_ENDPOINT,
USB_MSC_DISK_BULK_OUT_ENDPOINT | (USB_OUT << 7U),
    USB_ENDPOINT_BULK, USB_SHORT_GET_LOW(FS_MSC_DISK_BULK_OUT_PACKET_SIZE),
    USB_SHORT_GET_HIGH(FS_MSC_DISK_BULK_OUT_PACKET_SIZE), 0x00,/*Do not miss this comma*/  /*
The polling interval value is every 0 Frames */
    /* Interface Descriptor */
    USB_DESCRIPTOR_LENGTH_INTERFACE, /* Size of this descriptor in bytes */
    USB_DESCRIPTOR_TYPE_INTERFACE,   /* INTERFACE Descriptor Type */
    USB_HID_MOUSE_INTERFACE_INDEX,   /* Number of this interface. */
    0x00U,                           /* Value used to select this alternate setting
                                        for the interface identified in the prior field */
    USB_HID_MOUSE_ENDPOINT_COUNT,    /* Number of endpoints used by this
                                        interface (excluding endpoint zero). */
    USB_HID_MOUSE_CLASS,             /* Class code (assigned by the USB-IF). */
    USB_HID_MOUSE_SUBCLASS,          /* Subclass code (assigned by the USB-IF). */
    USB_HID_MOUSE_PROTOCOL,          /* Protocol code (assigned by the USB). */
    0x00U,                           /* Index of string descriptor describing this interface
*/

    USB_DESCRIPTOR_LENGTH_HID, /* Numeric expression that is the total size of the HID
descriptor. */
    USB_DESCRIPTOR_TYPE_HID,   /* Constant name specifying type of HID descriptor. */
    0x00U, 0x01U,              /* Numeric expression identifying the HID Class Specification
release. */
    0x00U,                     /* Numeric expression identifying country code of the
localized hardware */
    0x01U, /* Numeric expression specifying the number of class descriptors(at least one
report descriptor) */
    USB_DESCRIPTOR_TYPE_HID_REPORT, /* Constant name identifying type of class descriptor. */
    USB_SHORT_GET_LOW(USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT),
    USB_SHORT_GET_HIGH(USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT),
    /* Numeric expression that is the total size of the Report descriptor. */
    USB_DESCRIPTOR_LENGTH_ENDPOINT, /* Size of this descriptor in bytes */
    USB_DESCRIPTOR_TYPE_ENDPOINT,   /* ENDPOINT Descriptor Type */
    USB_HID_MOUSE_ENDPOINT | (USB_IN << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT),
    /* The address of the endpoint on the USB device
       described by this descriptor. */
    USB_ENDPOINT_INTERRUPT, /* This field describes the endpoint's attributes */
    USB_SHORT_GET_LOW(FS_INTERRUPT_IN_PACKET_SIZE),
    USB_SHORT_GET_HIGH(FS_INTERRUPT_IN_PACKET_SIZE), /* Maximum packet size this endpoint is
capable of
                                                        sending or receiving when this
configuration is selected. */
    FS_INTERRUPT_IN_INTERVAL,                       /* Interval for polling endpoint for
data transfers. */
};
```

Update the speed/package configuration:

```
usb_status_t USB_DeviceSetSpeed(usb_device_handle handle, uint8_t speed)
{
    ......
    while (ptr1 < ptr2)
    {
        if (ptr1->common.bDescriptorType == USB_DESCRIPTOR_TYPE_ENDPOINT)
        {
            if (USB_SPEED_HIGH == speed)
            {
                if (USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT ==
                    (ptr1->endpoint.bEndpointAddress & USB_ENDPOINT_NUMBER_MASK))
                {
                    ptr1->endpoint.bInterval = HS_CDC_VCOM_INTERRUPT_IN_INTERVAL;
                    USB_SHORT_TO_LITTLE_ENDIAN_ADDRESS(HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE,
                                                       ptr1->endpoint.wMaxPacketSize);
                }
                ... ...
                else if (USB_HID_MOUSE_ENDPOINT == (ptr1->endpoint.bEndpointAddress &
USB_ENDPOINT_NUMBER_MASK))
                {
                    ptr1->endpoint.bInterval = HS_ISO_IN_ENDP_INTERVAL;
                    USB_SHORT_TO_LITTLE_ENDIAN_ADDRESS(HS_ISO_IN_ENDP_PACKET_SIZE, ptr1-
>endpoint.wMaxPacketSize);
                }
                ... ...            }
            else
            {
                if (USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT ==
                    (ptr1->endpoint.bEndpointAddress & USB_ENDPOINT_NUMBER_MASK))
                {
                    ptr1->endpoint.bInterval = FS_CDC_VCOM_INTERRUPT_IN_INTERVAL;
                    USB_SHORT_TO_LITTLE_ENDIAN_ADDRESS(FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE,
                                                       ptr1->endpoint.wMaxPacketSize);
                }
                ... ...
                else if (USB_HID_MOUSE_ENDPOINT == (ptr1->endpoint.bEndpointAddress &
USB_ENDPOINT_NUMBER_MASK))
                {
                    ptr1->endpoint.bInterval = FS_ISO_IN_ENDP_INTERVAL;
                    USB_SHORT_TO_LITTLE_ENDIAN_ADDRESS(FS_ISO_IN_ENDP_PACKET_SIZE, ptr1-
>endpoint.wMaxPacketSize);
                }
                ... ...
            }
        }
        ptr1 = (usb_descriptor_union_t *)((uint8_t *)ptr1 + ptr1->common.bLength);
    }
    ... ...
    for (int i = 0U; i < USB_HID_MOUSE_ENDPOINT_COUNT; i++)
    {
        if (USB_SPEED_HIGH == speed)
        {
            g_UsbDeviceHidEndpoints[i].maxPacketSize = HS_INTERRUPT_IN_PACKET_SIZE;
        }
        else
        {
            g_UsbDeviceHidEndpoints[i].maxPacketSize = FS_INTERRUPT_IN_PACKET_SIZE;
```

```
        }
    }
    return kStatus_USB_Success;
}
```

Add the data and function for the new HID class:

```
// HID-----------------------------------------------------------------------------
uint8_t g_UsbDeviceHidMouseReportDescriptor[USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT] = {
    0x05U, 0x01U, /* Usage Page (Generic Desktop)*/
    0x09U, 0x02U, /* Usage (Mouse) */
    0xA1U, 0x01U, /* Collection (Application) */
    0x09U, 0x01U, /* Usage (Pointer) */

    0xA1U, 0x00U, /* Collection (Physical) */
    0x05U, 0x09U, /* Usage Page (Buttons) */
    0x19U, 0x01U, /* Usage Minimum (01U) */
    0x29U, 0x03U, /* Usage Maximum (03U) */

    0x15U, 0x00U, /* logical Minimum (0U) */
    0x25U, 0x01U, /* logical Maximum (1U) */
    0x95U, 0x03U, /* Report Count (3U) */
    0x75U, 0x01U, /* Report Size (1U) */

    0x81U, 0x02U, /* Input(Data, Variable, Absolute) 3U button bit fields */
    0x95U, 0x01U, /* Report count (1U) */
    0x75U, 0x05U, /* Report Size (5U) */
    0x81U, 0x01U, /* Input (Constant), 5U constant field */

    0x05U, 0x01U, /* Usage Page (Generic Desktop) */
    0x09U, 0x30U, /* Usage (X) */
    0x09U, 0x31U, /* Usage (Y) */
    0x09U, 0x38U, /* Usage (Z) */

    0x15U, 0x81U, /* Logical Minimum (-127) */
    0x25U, 0x7FU, /* Logical Maximum (127) */
    0x75U, 0x08U, /* Report Size (8U) */
    0x95U, 0x03U, /* Report Count (3U) */

    0x81U, 0x06U, /* Input(Data, Variable, Relative), three position bytes (X & Y & Z)*/
    0xC0U,        /* end collection, Close Pointer collection*/
    0xC0U         /* end collection, Close Mouse collection */
};
/* Get hid report descriptor request */
usb_status_t USB_DeviceGetHidReportDescriptor(usb_device_handle handle,
                                              usb_device_get_hid_report_descriptor_struct_t
*hidReportDescriptor)
{
    if (USB_HID_MOUSE_INTERFACE_INDEX == hidReportDescriptor->interfaceNumber)
    {
        hidReportDescriptor->buffer = g_UsbDeviceHidMouseReportDescriptor;
        hidReportDescriptor->length = USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT;
    }
    else
    {
        return kStatus_USB_InvalidRequest;
    }
    return kStatus_USB_Success;
}
```

3.2.5. Updating *usb_device_descriptor.h*

Find out how many interfaces/endpoints are used in the composite device. The endpoint number and interface index must be configured correctly. This is very important because any mistake here leads to an enumeration failure, and then it is very difficult to debug the root cause.

In the current application, determine the correct value for USB_HID_MOUSE_ENDPOINT_IN and USB_HID_MOUSE_INTERFACE_INDEX.

In the *dev_composite_cdc_msc* demo, three interfaces are used. Two of them are used for the CDC class, and one is used for the MSD. The corresponding definition is shown here:

```
#define USB_CDC_VCOM_INTERFACE_COUNT (2)      // cdc use two interfaces
#define USB_CDC_VCOM_CIC_INTERFACE_INDEX (0) // the interface index is 0 and 1
#define USB_CDC_VCOM_DIC_INTERFACE_INDEX (1)
#define USB_MSC_DISK_INTERFACE_COUNT (1)      // MSC DISK use one interface
#define USB_MSC_DISK_INTERFACE_INDEX (2)      // the interface index is 2
```

The interface indexes used are 0, 1, and 2. To add a new interface, the interface index must be 3.

```
#define USB_HID_MOUSE_INTERFACE_INDEX (3)     // interface index
```

MSC_DISK uses two endpoints and the endpoint numbers are 1 and 2:

```
#define USB_MSC_DISK_ENDPOINT_COUNT     (2)
#define USB_MSC_DISK_BULK_IN_ENDPOINT   (1)
#define USB_MSC_DISK_BULK_OUT_ENDPOINT (2)
```

The following code shows that the CDC uses also two endpoints and the endpoint numbers are 3 and 4:

```
#define USB_CDC_VCOM_CIC_ENDPOINT_COUNT         (1)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT (4)
#define USB_CDC_VCOM_DIC_ENDPOINT_COUNT         (2)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT       (3)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT      (3)
```

To add a new endpoint, the endpoint number must be 5:

```
#define USB_HID_MOUSE_ENDPOINT_IN      (5)    // the endpoint number used by HID
```

This code shows that the total interface count is 3:

```
#define USB_INTERFACE_COUNT (3)
```

For the HID class, add one interface. It must be updated to be 3+1:

```
#define USB_INTERFACE_COUNT (3+1) // add one interface for the new device class.
```

As for the guidance provided above, add this code:

```
#define USB_CDC_TELEPHONE_CONTROL_FUNC_DESC (0x18)
#define USB_CDC_OBEX_SERVICE_ID_FUNC_DESC (0x19)
/*HID*/
#define USB_HID_MOUSE_CLASS             (0x03)
#define USB_HID_MOUSE_SUBCLASS          (0x01)
#define USB_HID_MOUSE_PROTOCOL          (0x02)
#define USB_HID_MOUSE_INTERFACE_COUNT (1)
#define USB_HID_MOUSE_ENDPOINT_IN      (5)    // the endpoint number used by HID
#define USB_HID_MOUSE_INTERFACE_INDEX (3)    // interface index
#define USB_HID_MOUSE_ENDPOINT_COUNT   (1)
#define FS_INTERRUPT_IN_PACKET_SIZE    (8)
#define HS_INTERRUPT_IN_PACKET_SIZE    (8)
#define FS_INTERRUPT_IN_INTERVAL       (0x08)
#define HS_ISO_IN_ENDP_PACKET_SIZE     (8)
```

```
#define FS_ISO_IN_ENDP_PACKET_SIZE    (8)
#define HS_ISO_IN_ENDP_INTERVAL       (0x04)
#define FS_ISO_IN_ENDP_INTERVAL       (0x01)
#define USB_DESCRIPTOR_LENGTH_HID     (9U)
#define USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT (52)
#define USB_HID_MOUSE_REPORT_LENGTH   (0x04U)
... ...
```

When updating the configuration descriptor, update also the length:

```
#define USB_DESCRIPTOR_LENGTH_CONFIGURATION_ALL (98+25) // modify the descriptor length
correctly.
... ...
#define USB_INTERFACE_COUNT (3+1) // add one interface for the new device class
... ...
usb_status_t USB_DeviceGetStringDescriptor(usb_device_handle handle,
                                           usb_device_get_string_descriptor_struct_t
*stringDescriptor);
usb_status_t USB_DeviceGetHidReportDescriptor(usb_device_handle handle,
                                              usb_device_get_hid_report_descriptor_struct_t
*hidReportDescriptor);
#endif /* _USB_DEVICE_DESCRIPTOR_H_ */
```

3.3. Other steps

The other minor steps are not discussed in this document. See the code attached to this application note for more details.

3.4. Items to modify/add

The list of the files/functions/data structures to modify/add is shown in Table 2.

Table 2. Items to modify/add

File name	Functions/data structures
composite.c	extern usb_device_class_struct_t g_UsbDeviceHidMouseClass;
	g_compositeDevice
	USB_DeviceCallback
	APPInit
composite.h	#include "hid_mouse.h"
	usb_device_composite_struct_t

	extern usb_status_t USB_DeviceHidMouseCallback(class_handle_t handle, uint32_t event, void *param);
	extern usb_status_t USB_DeviceHidMouseInit(usb_device_composite_struct_t *device_composite);
	extern usb_status_t USB_DeviceHidMouseSetConfigure(class_handle_t handle, uint8_t configure);

usb_device_config.h	#define USB_DEVICE_CONFIG_HID (1U)
	#define USB_DEVICE_CONFIG_ENDPOINTS (5+1)
usb_device_descriptor.c	g_UsbDeviceHidEndpoints
	g_UsbDeviceHidInterface
	g_UsbDeviceHidInterfaces
	g_UsbDeviceHidInterfaceList
	g_UsbDeviceHidMouseClass
	g_UsbDeviceConfigurationDescriptor
	USB_DeviceSetSpeed

Table 2. Items to modify/add

File name	Functions/data structures
	g_UsbDeviceHidMouseReportDescriptor
	USB_DeviceGetHidReportDescriptor
usb_device_descriptor.h	#define USB_HID_MOUSE_CLASS (0x03) #define USB_HID_MOUSE_SUBCLASS (0x01) #define USB_HID_MOUSE_PROTOCOL (0x02) #define USB_HID_MOUSE_INTERFACE_COUNT (1) #define USB_HID_MOUSE_ENDPOINT_IN (5) // the endpoint number used by HID #define USB_HID_MOUSE_INTERFACE_INDEX (3) #define USB_HID_MOUSE_ENDPOINT_COUNT (1) #define FS_INTERRUPT_IN_PACKET_SIZE (8) #define HS_INTERRUPT_IN_PACKET_SIZE (8) #define FS_INTERRUPT_IN_INTERVAL (0x08) #define HS_ISO_IN_ENDP_PACKET_SIZE (8) #define FS_ISO_IN_ENDP_PACKET_SIZE (8) #define HS_ISO_IN_ENDP_INTERVAL (0x04) #define FS_ISO_IN_ENDP_INTERVAL (0x01) #define USB_DESCRIPTOR_LENGTH_HID (9U) #define USB_DESCRIPTOR_LENGTH_HID_MOUSE_REPORT (52) #define USB_HID_MOUSE_REPORT_LENGTH (0x04U)
	#define USB_DESCRIPTOR_LENGTH_CONFIGURATION_ALL (98+25)
	#define USB_INTERFACE_COUNT (3+1)
	USB_DeviceGetHidReportDescriptor
usb_device_hid.h	Add to project
usb_device_hid.c	Add to project
hid_mouse.h	Add to project
hid_mouse.c	Add to project

4. Running the result

After performing the steps described in Section 3, “Steps” and downloading the code to the target board and running it, three devices appear in the device list, as shown in Figure 3.

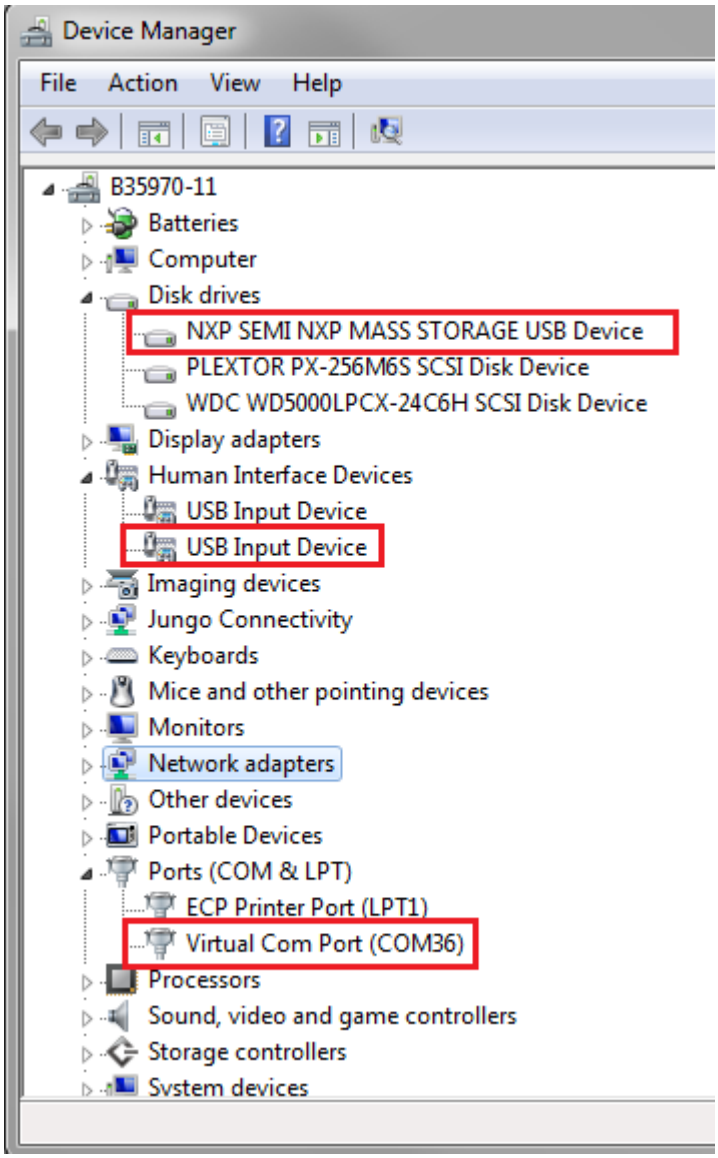

Figure 3. CDC, MSD, and HID mouse are implemented

5. Attached code

Figure 4 shows the provided packages.

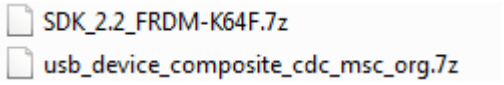

Figure 4. Attached code

The *SDK_2.2_FRDM-K64F.7z* package contains the modified project into which the MSD, CDC, and HID are implemented.

The *usb_device_composite_cdc_msc_org.7z* package contains the project before the modification. Using code-comparison tools, you can see all the changes to implement.

6. Conclusion

This application note shows a quick way of adding a HID device to the *usb_device_composite_cdc_msc* example. It shows how to implement it in the SDK2.2. If your application requirements are different, new combinations based on the common rules provided in this application note can be created (CDC+HID, MSC+HID, and so on).

7. Revision history

Table 3 summarizes the changes done to this document since the initial release.

Table 3. Revision history

Revision number	Date	Substantive changes
0	07/2017	Initial release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12023
Rev. 0
08/2017