Freescale Semiconductor, Inc.
User's Guide

Document Number: USBCOMDUG
Rev. 2, 09/2015

Composite Device User's Guide

1 Overview

This document describes steps to implement the composite device based on the USB stack.

The USB Stack provides two composite device demos, *hid+audio* and *msc+cdc*. However, users can create composite devices to fit their needs. This document is a detailed step-by-step guide to create a customizable composite device**.**

Contents

2 Introduction

The composite device combines multiple independent functionalities by unifying independent functionality code into one example. For example, the single functionality code for CDC is provided in the CDC example and the single functionality code for MSC is provided in the MSC example. Creating the CDC+MSC composite device example requires combining the CDC example code and MSC example code into a single example.

Composite device descriptors are combined from the single-function device descriptors. There are two single-function devices and each device has only one interface descriptor in its configuration descriptor. If the composite device is combined using two single function devices, the interface descriptor of each device should be merged into the composite device configuration descriptor.

Implementing a composite device involves combining the descriptors and the functionality of the single function devices.

3 Setup

Before developing the composite device, the user needs to:

1. Decide how many classes to include in this composite device.
2. Decide which types of classes to include in this composite device, for example, HID + AUDIO, HID + HID, etc.
3. Prepare the device descriptor depending on use case. Particularly, the IAD should be used for AUDIO/VIDEO class, (see www.usb.org/developers/docs/whitepapers/iadclasscode_r10.pdf)
4. Ensure that the functionality of the single function device code is valid.

3.1 Design Steps

1. Prepare the descriptor-related data structure to ensure that correct information about the customized composite device is relayed to the USB device stack. See Section 4.
2. Prepare the descriptors array and ensure the descriptors are consistent with the descriptor-related data structure. See Section 5.
3. Implement the specific descriptor-related callback function which the USB Device stack calls to get the device descriptor. See Section 5.

4 USB Composite Device Structures

All these structures are defined in the USB stack code. The structures describe the class and are consistent with the descriptor. They are also used in examples.

4.1 usb_composite_info_struct_t

This structure is required for the composite device and relays interface numbers and endpoint numbers of each interface to the class driver.

This is an example for a composite device MSD + CDC:

```
static usb_composite_info_struct_t usb_composite_info =
{
    . count  =  2,
    . class    = usb_dec_class,
};
```

The variable “count” holds the number of classes included in the composite device. Because composite device MSD+CDC includes two classes, the value of variable "count" is 2.

The type of “class” is usb_class_struct_t. See the next section for more information.

4.2 usb_class_struct_t

This structure is required for the composite device and provides information about each class.

This is an example for the composite device MSD + CDC:

```
static usb_class_struct_t usb_dec_class[USB_MSD_CDC_CLASS_MAX] =
{
    {
        . type = USB_CLASS_CDC,
        . interfaces = USB_DESC_CONFIGURATION(2, usb_cdc_if),
    },
    {
        . type = USB_CLASS_MSC,
        . interfaces  = USB_DESC_CONFIGURATION(1, usb_msd_if),
    },
};
```

Type represents the type of each class included in the composite device. For example, the type of MSD class is USB_CLASS_MSC.

Interfaces include detailed interface information about the class, including interface count, ep count, ep type and ep direction. See next section for more information. Number “2” means that the CDC class has two interfaces. The interface list is “usb_cdc_if”. Number “1” means that the MSD class has one interface. The interface list is “usb_msd_if”. See next section for more information.

4.3 usb_interfaces_struct_t

This structure is required for the composite device and provides information about each class.

Prototype:

```
typedef struct _usb_interfaces_struct
{
    uint8_t               count;
    usb_if_struct_t*      interface;
```

```
} usb_interfaces_struct_t;
```

Description:

count: interface numbers for each class.

interface: interface information list.

4.4 usb_if_struct_t

This structure is required for the composite device and provides information about each interface.

Prototype:

```
typedef struct _usb_if_struct
{
    uint8_t               index;
    usb_endpoints_t       endpoints;
} usb_if_struct_t;
```

Description:

index: interface index in the interface descriptor.

endpoints: ep information struct.

This is an example for the composite device MSD + CDC:

CDC:

```
static usb_if_struct_t usb_cdc_if[2] = {
    USB_DESC_INTERFACE(0, 1, cic_ep),
    USB_DESC_INTERFACE(1, 2, dic_ep),
};
In USB_DESC_INTERFACE(0, 1, cic_ep):
```

Number “0” holds the index of the CDC class control interface. In other words, in the interface descriptor, the interface number is 0.

Number “1” means that the ep number of this interface is 1.

“cic_ep” is ep detail information structure. See Section 4.6 for more information.

MSD:

```
static usb_if_struct_t usb_msd_if[1] = {
    USB_DESC_INTERFACE(2, 2, msd_ep),
};
In USB_DESC_INTERFACE(2, 2, msd_ep):
```

The first number “2” holds the index of the MSD class control interface. In other words, in the interface descriptor, the interface number is 2.

The second number “2” means that the ep number of this interface is 2.

“cic_ep” is ep detail information structure. See Section 3.6 for more information.

4.5 usb_endpoints_t

This structure is required for the composite device and provides ep information for each interface.

4.6 usb_ep_struct_t

This structure is required for the composite device and provides ep information.

This is an example for the composite device MSD + CDC:

This is CDC class control interface endpoint information.

```
usb_ep_struct_t cic_ep[CIC_ENDP_COUNT] =  {
        #if CIC_NOTIF_ELEM_SUPPORT
        {
            .ep_num  = 5,
            .type = USB_INTERRUPT_PIPE,
            .direction = USB_SEND,
            .size = 16
        }
        #endif
};
```

5 USB descriptor functions

5.1 USB descriptor

The descriptors for each class can be obtained from the class-related examples and class spec. For composite device, the user should combine multiple class descriptors.

Note

1. Interface number in the configuration descriptor must be the correct interface number value.
2. The endpoint number value in each endpoint descriptor must be consistent with the structures in Section 1.

5.2 USB_Desc_Get_Descriptor

This function provides all standard descriptors, such as descriptor for device and class. All composite devices must implement USB_STRING_DESCRIPTOR and USB_STANDARD_DESCRIPTOR. If the class has a class-specific descriptor, it needs to be added to this function. For example, for HID class, the USB_REPORT_DESCRIPTOR needs to be added.

USB_Desc_Get_Descriptor is provided in the example when a composite device is HID + HID. Omit part can be a reference in the USB Stack.

```
uint8_t USB_Desc_Get_Descriptor(…)
{
    switch(type)
    {
        case USB_REPORT_DESCRIPTOR:
            {
                if(index == HID_GENERIC_INTERFACE_INDEX)
                {
                    *descriptor = (uint8_t *)g_generic_report_descriptor;
                    *size = GENERIC_REPORT_DESC_SIZE;
                }
                else if(index == HID_MOUSE_INTERFACE_INDEX)
                {
                    *descriptor = (uint8_t *)g_mouse_report_descriptor;
                    *size = MOUSE_REPORT_DESC_SIZE;
                }
                else
                {
                    *descriptor = NULL;
                    *size = 0;
                }
            }
            break;
    …
    }/* End Switch */
    return USB_OK;
}
```

5.3 USB_Desc_Get_Entity

This function provides information about a class driver. Any composite equipment must implement the USB_COMPOSITE_INFO and the USB_CLASS_INTERFACE_INDEX_INFO.

For Audio class, add the USB_AUDIO_UNITS.

For MSC, add the USB_MSC_LBA_INFO.

For PHDC, add the USB_PHDC_QOS_INFO.

This is an example for the composite device HID + HID.

```
uint8_t USB_Desc_Get_Entity(uint32_t handle, entity_type type, uint32_t * object)
{
    switch (type)
    {
        case USB_CLASS_INFO:
            break;
        case USB_CLASS_INTERFACE_INDEX_INFO:
            *object = 0xff;
                                    if (handle ==
(uint32_t)g_composite_device.hid_generic.app_handle)
            {
                *object = (uint32_t)HID_GENERIC_INTERFACE_INDEX;
                break;
            }
```

```
            else if (handle == (uint32_t)g_composite_device.hid_mouse.app_handle)
            {
                *object = (uint32_t)HID_MOUSE_INTERFACE_INDEX;
                break;
            }
                        break;
        case USB_COMPOSITE_INFO:
            g_usb_if_hid_generic[0].index = HID_GENERIC_INTERFACE_INDEX;
            g_usb_if_hid_generic[0].endpoints = g_usb_desc_ep_hid_generic;
            g_usb_if_hid_mouse[0].index = HID_MOUSE_INTERFACE_INDEX;
            g_usb_if_hid_mouse[0].endpoints = g_usb_desc_ep_hid_mouse;
            *object = (unsigned long)&g_usb_composite_info;
            break;
        default :
            break;
    }/* End Switch */
    return USB_OK;
}.
```

Note:

USB_CLASS_INTERFACE_INDEX_INFO get the index of class arrange. It is the position of usb_class_struct_t array for a class.

5.4 USB_Set_Configation

This is a most common configuration and the USB_Set_Configation can be empty. For a composite device, if you complete this function, you should determine which class the handle needs.

This is an example for the two CDC devices:

```
uint8_t USB_Set_Configation( cdc_handle_t handle, uint8_t config )
{
    for(i = 0; i < usb_composite_info.count; i++)
    {
        switch(usb_composite_info.class[i].type)
        {
            case USB_CLASS_CDC:
            if (handle == g_composite_device.cdc_vcom1.cdc_handle)
                usb_dec_class[i].interfaces = usb_cdc_configuration[config - 1];
            else if (handle == g_composite_device.cdc_vcom2.cdc_handle)
                usb_dec_class[i].interfaces = usb_cdc2_configuration[config - 1];
            break;
            default:
                break;
        }
    }
    return USB_OK;
}
```

6 USB Stack Configurations

1. Class Configuration:

This section provides more information for whenever two or more of the same classes are used in the composite device.

To reduce the footprint, the released USB stack does not support multiple instances of the same class in the default configuration. If two or more same of the classes are used in the composite device, the user needs to configure the class.

- For HID class, MAX_HID_DEVICE must be configured in the usb_hid.h
- For CDC class, MAX_CDC_DEVICE must be configured in the usb_cdc.h
- For MSD class, MAX_MSC_DEVICE must be configured in the usb_msc.h
- For audio class, MAX_AUDIO_DEVICE must be configured in the usb_audio.h
- For PHDC class, MAX_PHDC_DEVICE must be configured in the usb_phdc.h
- For Composite class driver, CONFIG_MAX must be configured in the usb_composite.c

The value of the configuration depends on use cases and user's needs.

For example, for the composite device HID+HID, the MAX_HID_DEVICE must be set to 2, and the CONFIG_MAX must be set to 2.

Note:

USBCFG_DEV_MAX_ENDPOINTS must not be less than "max used endpoint number + 1". "max used endpoint number" means the max endpoint number the example uses.

7 Application template

The main difference between the composite devices and other devices is application. Designing a composite device application is to design a composite device demo.

7.1 Application structure template

For a general device, a demo contains only one class. However, for the composite device, a demo contains more than one class. Likewise, a structure is required to manage the application involving more than one class.

```
typedef struct composite_device_struct
{
    composite_handle_t          composite_device;
    Function 1 structure;
    Function 2 structure;
    …
    Function n structure;
    composite_config_struct_t   composite_device_config_callback;
    class_config_struct_t       composite_device_config_list[COMPOSITE_CFG_MAX];
}composite_device_struct_t;

composite_device: the handle point to composite device. Returned by USB_Composite_Init.
composite_device_config_callback: function callback list and count.
```

```
composite_device_config_list: function callback list.

Function n structure: structure to application of a class. Just like HID mouse,
hid_mouse_struct_t hid_mouse;
HID mouse is an example to show function n structure.
```

This is an example for a composite device HID mouse + keyboard:

Prototype:

```
typedef struct composite_device_struct
{
    composite_handle_t          composite_device;
    hid_keyboard_struct_t       hid_keyboard;
    hid_mouse_struct_t          hid_mouse;
    composite_config_struct_t   composite_device_config_callback;
    class_config_struct_t       composite_device_config_list[COMPOSITE_CFG_MAX];
}composite_device_struct_t;
```

7.2 Application initialization process

1. Before initializing the USB stack by calling the USB_Composite_Init, the composite_device_config_list and composite_device_config_callback are assigned values respectively. For example, for HID mouse, the steps are as follows:
    - g_composite_device need to be declared as global variables, as the type composite_device_struct_t.

    b. Declare: class_config_struct_t* hid_mouse_config_callback_handle;

    - Then hid_mouse_config_callback_handle = &g_composite_device.composite_device_config_list[HID_MOUSE_INTERFACE_INDEX];

    c.

    - hid_mouse_config_callback_handle->composite_application_callback.callback = hid_mouse_app_callback;
    - hid_mouse_app_callback amended as Hid_USB_App_Device_Callback
    - hid_mouse_config_callback_handle->composite_application_callback.arg = &g_composite_device.hid_mouse;
    - hid_mouse_config_callback_handle->class_specific_callback.callback = hid_mouse_app_param_callback;
    - hid_mouse_app_param_callback
    - amended as

- Hid_USB_App_Class_Callback
- hid_mouse_config_callback_handle->class_specific_callback.arg = &g_composite_device.hid_mouse;
- hid_mouse_config_callback_handle->desc_callback_ptr = &g_desc_callback;
- hid_mouse_config_callback_handle->type = USB_CLASS_HID;
- OS_Mem_zero(&g_composite_device.hid_mouse, sizeof(hid_mouse_struct_t));

2.

- g_composite_device.composite_device_config_callback.count = 2;
- g_composite_device.composite_device_config_callback.class_app_callback = g_composite_device.composite_device_config_list;

3. Call USB_Composite_Init

- USB_Composite_Init(CONTROLLER_ID, &g_composite_device.composite_device_config_callback, &g_composite_device.composite_device);

4. Get a handle for each class.

   For example, HID mouse:

- g_composite_device.hid_mouse.app_handle = (hid_handle_t)g_composite_device.composite_device_config_list[HID_MOUSE_INTERFACE_INDEX].class_handle;

5. Initialize each class application.

   Such as, HID mouse:

- hid_mouse_init(&g_composite_device.hid_mouse);

8 CDC+CDC Composite device example

For this section, we use CDC+CDC composite device as an example.

8.1 USB Composite Device Structure examples

```
/* two cdc classes */
static usb_composite_info_struct_t usb_composite_info =
{
   2,
   usb_dec_class,
};

/* two cdc classes definition */
static usb_class_struct_t usb_dec_class[USB_MSD_CDC_CLASS_MAX] =
{
   {
      USB_CLASS_CDC,
      USB_DESC_CONFIGURATION(USB_CDC_IF_MAX, usb_cdc_if),
   },
   {
      USB_CLASS_CDC,
      USB_DESC_CONFIGURATION(USB_CDC_IF_MAX, usb_cdc2_if),
   },
};

/* cdc1 definition: cdc has two interfaces */
static usb_if_struct_t usb_cdc_if[USB_CDC_IF_MAX] = {
   USB_DESC_INTERFACE(0, CIC_ENDP_COUNT, cic_ep),
   USB_DESC_INTERFACE(1, DIC_ENDP_COUNT, dic_ep),
};
/* cdc2 definition: cdc has two interfaces */
```

```
static usb_if_struct_t usb_cdc2_if[USB_CDC_IF_MAX] = {
   USB_DESC_INTERFACE(2, CIC_ENDP_COUNT, cic_ep2),
   USB_DESC_INTERFACE(3, DIC_ENDP_COUNT, dic_ep2),
};

/* cdc1 endpoints definition: interface1 has one endpoint, interface2 has two endpoints*/
usb_ep_struct_t cic_ep[CIC_ENDP_COUNT] =  {
      #if CIC_NOTIF_ELEM_SUPPORT
      {
         CIC_NOTIF_ENDPOINT,
         USB_INTERRUPT_PIPE,
         USB_SEND,
         CIC_NOTIF_ENDP_PACKET_SIZE
      }
      #endif
};

usb_ep_struct_t dic_ep[DIC_ENDP_COUNT] =  {
      #if DATA_CLASS_SUPPORT
         {
            DIC_BULK_IN_ENDPOINT,
            USB_BULK_PIPE,
            USB_SEND,
            DIC_BULK_IN_ENDP_PACKET_SIZE
         },
         {
            DIC_BULK_OUT_ENDPOINT,
            USB_BULK_PIPE,
            USB_RECV,
            DIC_BULK_OUT_ENDP_PACKET_SIZE
```

```
        }
    #endif
};

/* cdc2 endpoints definition: interface1 has one endpoint, interface2 has two endpoints */
usb_ep_struct_t cic_ep2[CIC_ENDP_COUNT] =  {
    #if CIC_NOTIF_ELEM_SUPPORT
    {
        CIC2_NOTIF_ENDPOINT,
        USB_INTERRUPT_PIPE,
        USB_SEND,
        CIC_NOTIF_ENDP_PACKET_SIZE
    }
    #endif
};

usb_ep_struct_t dic_ep2[DIC_ENDP_COUNT] =  {
    #if DATA_CLASS_SUPPORT
        {
            DIC2_BULK_IN_ENDPOINT,
            USB_BULK_PIPE,
            USB_SEND,
            DIC_BULK_IN_ENDP_PACKET_SIZE
        },
        {
            DIC2_BULK_OUT_ENDPOINT,
            USB_BULK_PIPE,
            USB_RECV,
            DIC_BULK_OUT_ENDP_PACKET_SIZE
        }
```

```
    #endif
};
```

8.2 USB Composite Device descriptor examples

Modify the product ID in device descriptor. The other does not need to change.

Change interface number as shown in configuration descriptor.

Copy two CDC configuration descriptors from the CDC example or msc+cdc example and change the endpoint number to be consistent with Section 8.1.

8.2.1 USB_Desc_Get_Descriptor

```
/* string descriptor get from usb_all_languages_t g_languages */
/* other descriptor get from g_std_descriptors array */
uint8_t USB_Desc_Get_Descriptor(…)
{
    if (type == USB_STRING_DESCRIPTOR)
    {
        if(index == 0)
        {
            *descriptor = (uint8_t *)g_languages.languages_supported_string;
            *size = g_languages.languages_supported_size;
        }
        else
        {
            uint8_t lang_id=0;
            uint8_t lang_index=USB_MAX_LANGUAGES_SUPPORTED;

            for(;lang_id< USB_MAX_LANGUAGES_SUPPORTED;lang_id++)
            {
                if(index == g_languages.usb_language[lang_id].language_id)
                {
                    if(str_num < USB_MAX_STRING_DESCRIPTORS)
                    {
                        lang_index=str_num;
                    }
                    break;
                }
            }

            *descriptor =
              (uint8_t *)g_languages.usb_language[lang_id].lang_desc[str_num];
            *size =
                  g_languages.usb_language[lang_id].lang_desc_size[lang_index];
        }
    }
    else if (type < USB_MAX_STD_DESCRIPTORS+1)
    {
        *descriptor = (uint8_t *)g_std_descriptors [type];
        if(*descriptor == NULL)
        {
            return USBERR_INVALID_REQ_TYPE;
        }
```

```
        *size = g_std_desc_size[type];
    }
    else
    {
        return USBERR_INVALID_REQ_TYPE;
    }
    return USB_OK;
}
```

8.2.2 USB_Desc_Get_Entity

```
/* USB_CLASS_INTERFACE_INDEX_INFO: Return different index based on different handle */
/* USB_COMPOSITE_INFO:  return usb_composite_info  defined in section 1*/
uint8_t USB_Desc_Get_Entity(uint32_t handle, entity_type type, uint32_t * object)
{
    switch(type)
    {
        case USB_CLASS_INFO:
            break;
        case USB_CLASS_INTERFACE_INDEX_INFO:
            *object = 0xff;

            if (handle == (uint32_t)g_composite_device.cdc_vcom)
            {
                *object = (uint32_t)CDC_VCOM_INTERFACE_INDEX;
                break;
            }
            else if (handle == (uint32_t)g_composite_device.msc_disk.app_handle)
            {
                *object = (uint32_t)MSC_DISK_INTERFACE_INDEX;
                break;
            }
                        break;
        case USB_COMPOSITE_INFO:
            *object = (uint32_t)&usb_composite_info;
            break;
        default :
            break;
    }/* End Switch */
    return USB_OK;
}
```

8.2.3 USB_Set_Configation

```
/* set different class configure based on different handle. */
uint8_t USB_Set_Configation( cdc_handle_t handle, uint8_t config )
{
    for(i = 0; i < usb_composite_info.count; i++)
    {
        switch(usb_composite_info.class[i].type)
        {
            case USB_CLASS_CDC:
            if (handle == g_composite_device.cdc_vcom1.cdc_handle)
                usb_dec_class[i].interfaces = usb_cdc_configuration[config - 1];
            else if (handle == g_composite_device.cdc_vcom2.cdc_handle)
                usb_dec_class[i].interfaces = usb_cdc2_configuration[config - 1];
```

```
            break;
            default:
                break;
        }
    }
    return USB_OK;
}
```

8.3 USB Composite Device application example

8.3.1 Class Configuration

MAX_CDC_DEVICE is set to 2 in usb_cdc.h

USBCFG_DEV_MAX_ENDPOINTS is set to 9 in usb_device_config.h

8.3.2 CDC+CDC Application structure

```
/* cdc_struct_t represents cdc class */
typedef struct composite_device_struct
{
    composite_handle_t          composite_device;
    cdc_struct_t                cdc_vcom1;
    cdc_struct_t                cdc_vcom2;
    composite_config_struct_t   composite_device_config_callback;
    class_config_struct_t       composite_device_config_list[COMPOSITE_CFG_MAX];
}composite_device_struct_t;

/* cdc_struct_t is as follow. */
/* It contain variables for one cdc class. */
typedef struct _cdc_variable_struct
{
    cdc_handle_t cdc_handle;
    uint8_t g_curr_recv_buf[DATA_BUFF_SIZE];
    uint8_t g_curr_send_buf[DATA_BUFF_SIZE];
    uint8_t g_recv_size;
    uint8_t g_send_size;
    bool start_app;
    bool start_transactions;
    uint8_t out_endpoint;
    uint8_t in_endpoint;
}cdc_struct_t;
```

8.3.3 CDC+CDC Application

1. composite_device_struct_t g_composite_device;

2.

/* call cdc_vcom_preinit by parameter cdc_struct_t */

cdc_vcom_preinit(&g_composite_device.cdc_vcom1);

cdc_vcom_preinit(&g_composite_device.cdc_vcom2);

3. /* Init cdc1: use callback functions defined by existing examples; use g_composite_device.cdc_vcom1 as parameter for callback */

cdc_vcom_config_callback_handle = &g_composite_device.composite_device_config_list[CDC_VCOM_INTERFACE_INDEX];

cdc_vcom_config_callback_handle->composite_application_callback.callback = VCom_USB_App_Callback;

VCom_USB_App_Callback amended as VCom_USB_App_Device_Callback

cdc_vcom_config_callback_handle->composite_application_callback.arg = &g_composite_device.cdc_vcom1;

cdc_vcom_config_callback_handle->class_specific_callback.callback = (usb_class_specific_handler_func)VCom_USB_Notif_Callback;

VCom_USB_Notif_Callback amended as VCom_USB_App_Class_Callback

cdc_vcom_config_callback_handle->class_specific_callback.arg = &g_composite_device.cdc_vcom1;

cdc_vcom_config_callback_handle->desc_callback_ptr = &desc_callback;

cdc_vcom_config_callback_handle->type = USB_CLASS_CDC;

4. /* Init cdc2: use callback functions defined by existing examples; use g_composite_device.cdc_vcom2 as parameter for callback */

msc_disk_config_callback_handle = &g_composite_device.composite_device_config_list[MSC_DISK_INTERFACE_INDEX];

msc_disk_config_callback_handle->composite_application_callback.callback = VCom_USB_App_Callback; //SG

VCom_USB_App_Callback amended as VCom_USB_App_Device_Callback

msc_disk_config_callback_handle->composite_application_callback.arg = &g_composite_device.cdc_vcom2; //SG

msc_disk_config_callback_handle->class_specific_callback.callback = (usb_class_specific_handler_func)VCom_USB_Notif_Callback; //SG

VCom_USB_Notif_Callback amended as VCom_USB_App_Class_Callback

msc_disk_config_callback_handle->class_specific_callback.arg = &g_composite_device.cdc_vcom2; //SG

msc_disk_config_callback_handle->desc_callback_ptr = &desc_callback;

msc_disk_config_callback_handle->type = USB_CLASS_CDC;

5.

g_composite_device.composite_device_config_callback.count = 2;

g_composite_device.composite_device_config_callback.class_app_callback = g_composite_device.composite_device_config_list;

6. /* composite init function call */

USB_Composite_Init(CONTROLLER_ID,
&g_composite_device.composite_device_config_callback,
&g_composite_device.composite_device);

7. /* save the handle for each cdc device */

g_composite_device.cdc_vcom1.cdc_handle = (cdc_handle_t)g_composite_device.composite_device_config_list[CDC_VCOM_INTERFACE_INDEX].class_handle;

g_composite_device.cdc_vcom2.cdc_handle = (cdc_handle_t)g_composite_device.composite_device_config_list[1].class_handle;

8. /* init some fields of g_composite_device.cdc_vcom1 and g_composite_device.cdc_vcom1*/

g_composite_device.cdc_vcom1.out_endpoint = DIC_BULK_OUT_ENDPOINT;

g_composite_device.cdc_vcom2.out_endpoint = DIC2_BULK_OUT_ENDPOINT;

g_composite_device.cdc_vcom1.in_endpoint = DIC_BULK_IN_ENDPOINT;

g_composite_device.cdc_vcom2.in_endpoint = DIC2_BULK_IN_ENDPOINT;

9. /* cdc_vcom_init call. Step8 can be moved to this function code. */

cdc_vcom_init(&g_composite_device.cdc_vcom1);

cdc_vcom_init(&g_composite_device.cdc_vcom2);

10. /* app tasks: call cdc_vcom_task by cdc_struct_t parameter*/

void APP_task()

```
{
   while(TRUE)
   {
      cdc_vcom_task((void*)&g_composite_device.cdc_vcom1);
      cdc_vcom_task((void*)&g_composite_device.cdc_vcom2);
   }
}
```

9 Revision history

This table summarizes revisions to this document since the release of the previous version

Table 1 Revision History		
Revision number	**Date**	**Substantive changes**
0	12/2014	Initial release
1	04/2015	Substantive changes
2		Section 5.3, Section 6, Section 8.2.2, Section 8.3.1

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: USBCOMDUG
Rev. 2
09/2014