<http://www.cnblogs.com/shangdawei/p/4712305.html>

Communication Device Class，简称CDC  
USB Compound Device，USB复合设备  
USB Composite Device，USB组合设备  
  
摘要  
USB复合设备 Compound Device内嵌Hub和多个Function，每个Function都相当于一个独立的USB外设，有自己的PID/VID。  
USB组合设备Composite Device内只有一个Function，只有一套PID/VID，通过将不同的interface定义为不同的类来实现多个功能的组合。  
  
正文  
  
Compound Device内嵌Hub和多个Function，每个Function都相当于一个独立的USB外设，有自己的PID/VID。  
Composite Device内只有一个Function，只有一套PID/VID，通过将不同的interface定义为不同的类来实现多个功能的组合。  
  
很多人认为一个USB接口上实现多个设备，就是指复合设备，其实，这是不确切的，  
虽然USB Compound Device和USB Composite Device都会被百度翻译为USB复合设备。  
  
在一个USB接口上实现多个设备有2中方法，  
一种是Compound Device，就是复合设备；  
另一种是Composite Device，就是组合设备。  
  
在USB2.0的标准协议中，定义如下：  
When multiple functions are combined with a hub in a single package, they are referred to as a compound device.  
A device that has multiple interfaces controlled independently of each other is referred to as a composite device.  
  
所以，复合设备其实就是几个设备通过一个USB Hub形成的单一设备；  
组合设备也就是具有多个接口的设备，每个接口代表一个独立的设备。  
显然，如果是想同样的功能的话，组合设备的方法要简单很多(可以去看一下USB2.0协议中，USB2.0 Hub的复杂度)。

**参见**[**USB Descriptors**](http://www.cnblogs.com/shangdawei/archive/2013/04/19/3030091.html)

USB复合设备一般用Interface Association Descriptor（IAD）实现，就是在要合并的接口前加上IAD描述符。

IAD描述符：

typedef struct \_USBInterfaceAssociationDescriptor {

BYTE bLength: 0x08 //描述符大小

BYTE bDescriptorType: 0x0B //IAD描述符类型

BYTE bFirstInterface: 0x00 //起始接口

BYTE bInterfaceCount: 0x02 //接口数

BYTE bFunctionClass: 0x0E //类型代码

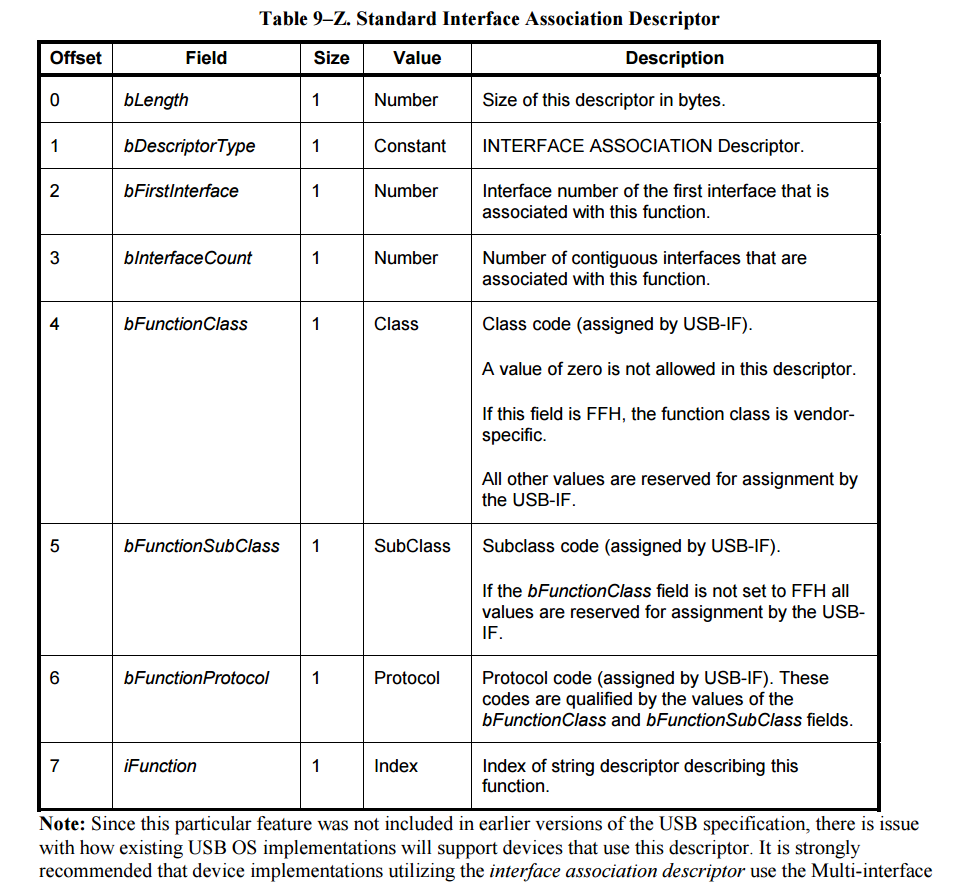
BYTE bFunctionSubClass: 0x03 //子类型代码

BYTE bFunctionProtocol: 0x00 //协议代码

BYTE iFunction: 0x04 //描述字符串索引

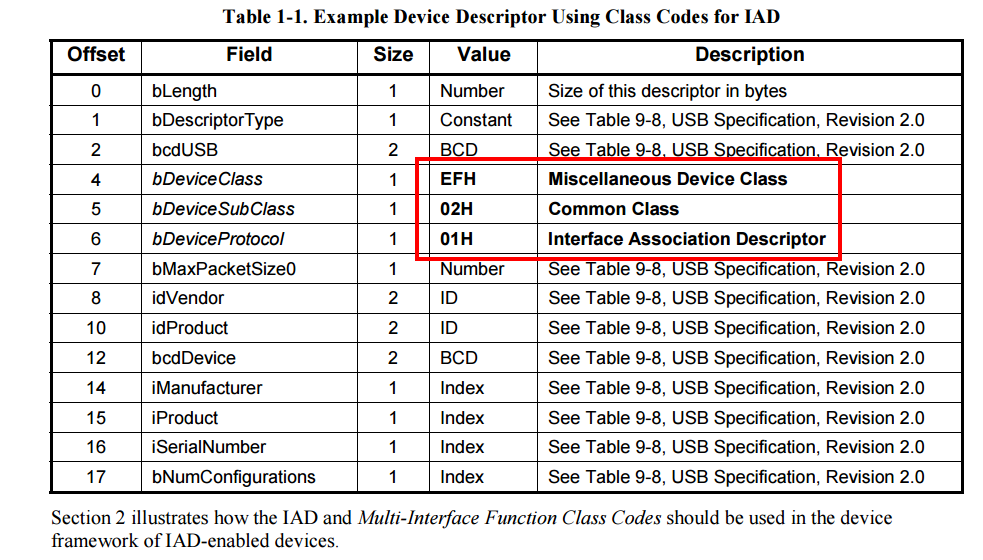
}

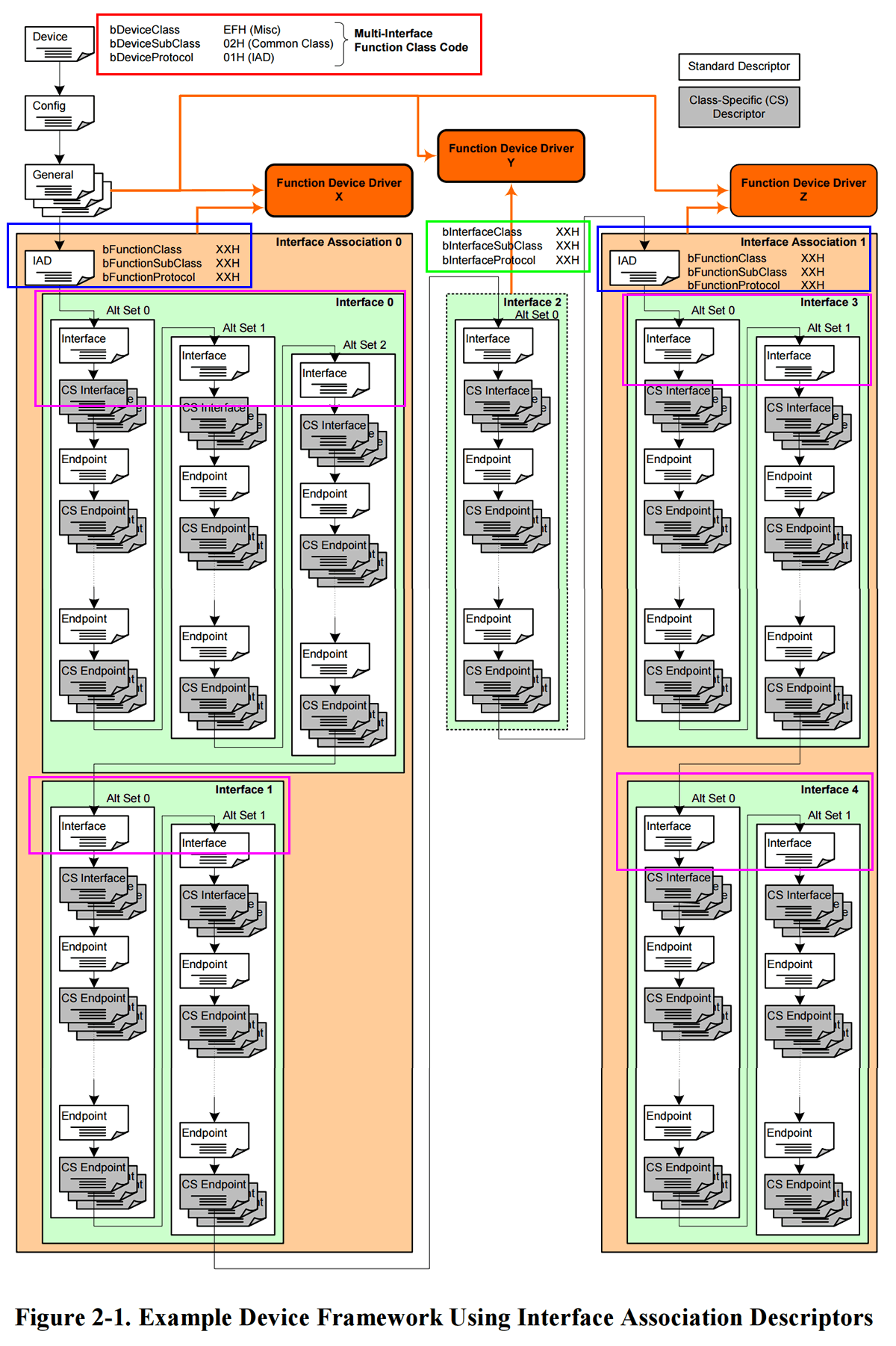
The bFirstInterface field of the IAD indicates the number of the first interface in the function.  
The bInterfaceCount field of the IAD indicates how many interfaces are in the interface collection.   
Interfaces in an IAD interface collection must be contiguous (there can be no gaps in the list of interface numbers),   
and so a count with a first interface number is sufficient to specify all of the interfaces in the collection.



<http://www.usb.org/developers/docs/whitepapers/iadclasscode_r10.pdf>

**USB Interface Association Descriptor Device Class code ans Use Model**





**USB Interface Association Descriptor Example**

The following illustrates a descriptor layout for a composite USB device.   
The example device has two functions:

Function 1: Video Class  
This function is defined by an interface association descriptor (IAD)   
and contains two interfaces: interface zero (0) and interface one (1).  
The system generates hardware and compatible identifiers (IDs) for the function,   
as described in Enumeration of Interface Collections on USB Devices with IADs.   
After matching the appropriate INF file, the system loads the Video Class driver stack.

Function 2: Human Input Device  
This function contains only one interface: interface two (2).  
The system generates hardware and compatible IDs for the function,   
as described in Enumeration of Interfaces on USB Composite Devices.   
After matching the appropriate INF file, the system loads the Human Input Device (HID) class driver.

--------------------------------------------------------------------------------

Device Descriptor:

--------------------------------------------------------------------------------

BYTE bLength 0x12

BYTE bDescriptorType 0x01

WORD bcdUSB 0x0200

BYTE bDeviceClass 0xEF

BYTE bDeviceSubClass 0x02

BYTE bDeviceProtocol 0x01

BYTE bMaxPacketSize0 0x40

WORD idVendor 0x045E

WORD idProduct 0xFFFF

WORD bcdDevice 0x0100

BYTE iManufacturer 0x01

WORD iProduct 0x02

WORD iSerialNumber 0x03

BYTE bNumConfigurations 0x01

--------------------------------------------------------------------------------

Configuration Descriptor:

--------------------------------------------------------------------------------

BYTE bLength 0x09

BYTE bDescriptorType 0x02

WORD wTotalLength 0x....

BYTE bNumInterfaces 0x03 < Interface 0, 1, 2 >

BYTE bConfigurationValue 0x01

BYTE iConfiguration 0x01

BYTE bmAttributes 0x80 (BUS Powered)

BYTE bMaxPower 0xFA (500 mA)

--------------------------------------------------------------------------------

Interface Association Descriptor:

--------------------------------------------------------------------------------

BYTE bLength 0x08

BYTE bDescriptorType 0x0B

BYTE bFirstInterface 0x00 < Interface 0 >

BYTE bInterfaceCount 0x02 < Interface 1 >

BYTE bFunctionClass 0x0E

BYTE bFunctionSubClass 0x03

BYTE bFunctionProtocol 0x00

BYTE iFunction 0x04

--------------------------------------------------------------------------------

Interface Descriptor (Video Control):

--------------------------------------------------------------------------------

BYTE bLength 0x09

BYTE bDescriptorType 0x04

BYTE bInterfaceNumber 0x00 < Interface 0 >

BYTE bAlternateSetting 0x00

BYTE bNumEndpoints 0x01

BYTE bInterfaceClass 0x0E

BYTE bInterfaceSubClass 0x01

BYTE bInterfaceProtocol 0x00

BYTE iInterface 0x05

Class Specific Descriptor(s):

Endpoint Descriptor(s):

--------------------------------------------------------------------------------

Interface Descriptor (Video Streaming):

--------------------------------------------------------------------------------

BYTE bLength 0x09

BYTE bDescriptorType 0x04

BYTE bInterfaceNumber 0x01 < Interface 1 >

BYTE bAlternateSetting 0x00

BYTE bNumEndpoints 0x01

BYTE bInterfaceClass 0x0E

BYTE bInterfaceSubClass 0x02

BYTE bInterfaceProtocol 0x00

BYTE iInterface 0x06

Class Specific Descriptor(s):

Endpoint Descriptor(s):

--------------------------------------------------------------------------------

Interface Descriptor (Human Input Devices):

--------------------------------------------------------------------------------

BYTE bLength 0x09

BYTE bDescriptorType 0x04

BYTE bInterfaceNumber 0x02 < Interface 2 >

BYTE bAlternateSetting 0x00

BYTE bNumEndpoints 0x01

BYTE bInterfaceClass 0x03

BYTE bInterfaceSubClass 0x01

BYTE bInterfaceProtocol 0x01

BYTE iInterface 0x07

Class Specific Descriptor(s):

Endpoint Descriptor(s):

**USB Interface Association Descriptor (IAD)**IAD是Interface Association Descriptor，功能是把多个接口定义为一个类设备。  
Windows下，IAD和Composite设备在设备管理器中没有什么区别，甚至使用的驱动也都是Composite驱动  
USB interface association descriptor allows the device to group interfaces that belong to a function.  
The Universal Serial Bus Specification, revision 2.0, does not support grouping more than one interface   
of a composite device within a single function.  
  
However, the USB Device Working Group (DWG) created USB device classes that allow for functions with multiple interfaces,   
and the USB Implementor's Forum issued an Engineering Change Notification (ECN) that defines a mechanism for grouping interfaces.  
  
The ECN specifies a USB descriptor, called the Interface Association Descriptor (IAD), that allows hardware manufacturers to define groupings of interfaces.  
The device classes that are most likely to use IADs include:  
  
USB Video Class Specification (Class Code - 0x0E)  
USB Audio Class Specification (Class Code - 0x01)  
USB Bluetooth Class Specification (Class Code - 0xE0)  
  
Windows 7, Windows Server 2008, Windows Vista, Microsoft Windows Server 2003 Service Pack 1 (SP1), and Microsoft Windows XP Service Pack 2 (SP2) support IADs.  
  
Manufacturers of composite devices typically assign a value of zero ( 0x00 ) in the device descriptor, as specified by the Universal Serial Bus Specification.

bDeviceClass　　　　　　0x00 - defined at Interface level

bDeviceSubClass　　　　0x00 - Unknown

bDeviceProtocol　　　　0x00 – Unknown

This allows the manufacturer to associate each individual interface with a different device class and protocol.

The USB-IF core team has devised a special class and protocol code set that notifies the operating system that one or more IADs are present in device firmware.   
**A device's device descriptor must have the values that appear in the following table**  
or else the operating system will not detect the device's IADs or group the device's interfaces properly.  
These code values also alert versions of Windows that do not support IADs to install a special-purpose bus driver that correctly enumerates the device.   
Without these codes in the device descriptor, the system might fail to enumerate the device, or the device might not work properly.

bDeviceClass　　　　　　0xEF

bDeviceSubClass　　　　0x02

bDeviceProtocol　　　　0x01

IAD是Interface Association Descriptor，功能是把多个接口定义为一个类设备。这个描述符是最新更新的，基本在书上是看不到的。  
  
多说无益，自己看<http://www.microsoft.com/whdc/archive/IAD.mspx>。用起来非常简单的。  
  
Windows下，IAD和Composite设备在设备管理器中没有什么区别，甚至使用的驱动也都是Composite驱动。  
Linux下，确实能发现这个设备，不过我用IAD实现的CDC并没有自动装载到ttyACM0，这个就以后研究了。

【问题】  
最近试了2种在一个USB接口上实现多个设备的方法，也成功的用这2种方式在偶的JTAGICE mkII上增加了一个CDC串口。  
偶的JTAGICE mkII还扩展了usb\_to\_xxx和vsllink的支持，usb\_to\_xxx和vsllink的USB底层是libusb。  
驱动安装完后，CDC串口和JTAGICE mkII(上位机用AVRStudio)都可以正常工作。  
但如果使用libusb的话，可能有一个问题，当调用usb\_set\_configuration时，libusb会阻塞，直到拔除USB设备后，才会返回。  
虽然AVRStudio可以正常操作，但avrdude却会出问题，同样在基于libusb的usb\_to\_xxx和vsllink上，也出同样的问题。  
  
【解决方法】  
发现在Windows的设备管理器中，禁用或者卸载这个CDC串口的话，就一切正常了。  
看来一山不容二虎，使用libusb的时候，就禁用CDC串口。使用CDC串口的时候，再使能。  
这个是由于Windows下的CDC设备，无法使用libusb操作，除非不使用usbser驱动，而是使用libusb驱动。

**JTAGICE mkII和USB-CDC弄一起的USB描述：**

const u8 USB\_ConfigDescriptor[USB\_CONFIG\_DESC\_SIZE] =

{

/\* Configuation Descriptor \*/

0x09, /\* bLength: Configuation Descriptor size \*/

USB\_CONFIGURATION\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Configuration \*/

USB\_CONFIG\_DESC\_SIZE, 0x00, /\* wTotalLength:no of returned bytes \*/

0x03, /\* bNumInterfaces: 3 interfaces \*/

0x01, /\* bConfigurationValue: Configuration value \*/

0x00, /\* iConfiguration: Index of string descriptor describing the configuration \*/

0x80, /\* bmAttributes: self powered \*/

0x32, /\* MaxPower 100 mA \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* **JTAGICE mkII** \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Interface Association Descriptor \*/

0x08, /\* bLength: Interface Descriptor size \*/

USB\_INTERFACE\_ASSOCIATION\_TYPE, /\* bDescriptorType: IAD \*/

0x00, /\* bFirstInterface \*/

0x01, /\* bInterfaceCount \*/

0xFF, /\* bFunctionClass \*/

0x00, /\* bFunctionSubClass \*/

0x00, /\* bFunctionProtocol \*/

0x02, /\* iFunction \*/

/\* **Interface Descriptor** \*/

0x09, /\* bLength: Interface Descriptor size \*/

USB\_INTERFACE\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Interface \*/

0x00, /\* bInterfaceNumber: Number of Interface \*/

0x00, /\* bAlternateSetting: Alternate setting \*/

0x02, /\* bNumEndpoints: Two endpoints used \*/

0xFF, /\* bInterfaceClass \*/

0x00, /\* bInterfaceSubClass \*/

0x00, /\* bInterfaceProtocol \*/

0x00, /\* iInterface: \*/

/\* **Endpoint IN2 Descriptor** \*/

0x07, /\* bLength: Endpoint Descriptor size \*/

USB\_ENDPOINT\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Endpoint \*/

0x82, /\* bEndpointAddress: (IN2) \*/

0x02, /\* bmAttributes: Bulk \*/

SNAIL\_DATA\_SIZE, 0x00, /\* wMaxPacketSize: 0x0040 \*/

0x0A, /\* bInterval: 10 \*/

/\* **Endpoint OUT2 Descriptor** \*/

0x07, /\* bLength: Endpoint Descriptor size \*/

USB\_ENDPOINT\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Endpoint \*/

0x02, /\* bEndpointAddress: (OUT2) \*/

0x02, /\* bmAttributes: Bulk \*/

SNAIL\_DATA\_SIZE, 0x00, /\* wMaxPacketSize \*/

0x0A, /\* bInterval \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* **USB-CDC** \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Interface Association Descriptor \*/

0x08, /\* bLength: Interface Descriptor size \*/

USB\_INTERFACE\_ASSOCIATION\_TYPE, /\* bDescriptorType: IAD \*/

0x01, /\* bFirstInterface \*/

0x02, /\* bInterfaceCount \*/

0x02, /\* bFunctionClass \*/

0x02, /\* bFunctionSubClass \*/

0x01, /\* bFunctionProtocol \*/

0x04, /\* iFunction \*/

/\* **Interface Descriptor** \*/

0x09, /\* bLength: Interface Descriptor size \*/

USB\_INTERFACE\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Interface \*/

0x01, /\* bInterfaceNumber: Number of Interface \*/

0x00, /\* bAlternateSetting: Alternate setting \*/

0x01, /\* bNumEndpoints: One endpoints used \*/

0x02, /\* bInterfaceClass: Communication Interface Class \*/

0x02, /\* bInterfaceSubClass: Abstract Control Model \*/

0x01, /\* bInterfaceProtocol: Common AT commands \*/

0x00, /\* iInterface \*/

/\* **Header Functional Descriptor** \*/

0x05, /\* bLength: Endpoint Descriptor size \*/

0x24, /\* bDescriptorType: CS\_INTERFACE \*/

0x00, /\* bDescriptorSubtype: Header Func Desc \*/

0x10, 0x01, /\* bcdCDC: spec release number \*/

/\* **Call Managment Functional Descriptor** \*/

0x05, /\* bFunctionLength \*/

0x24, /\* bDescriptorType: CS\_INTERFACE \*/

0x01, /\* bDescriptorSubtype: Call Management Func Desc \*/

0x00, /\* bmCapabilities: D0+D1 \*/

0x02, /\* bDataInterface: 1 \*/

/\* **ACM Functional Descriptor** \*/

0x04, /\* bFunctionLength \*/

0x24, /\* bDescriptorType: CS\_INTERFACE \*/

0x02, /\* bDescriptorSubtype: Abstract Control Management desc \*/

0x02, /\* bmCapabilities \*/

/\* **Union Functional Descriptor** \*/

0x05, /\* bFunctionLength \*/

0x24, /\* bDescriptorType: CS\_INTERFACE \*/

0x06, /\* bDescriptorSubtype: Union func desc \*/

0x01, /\* bMasterInterface: Communication class interface \*/

0x02, /\* bSlaveInterface0: Data Class Interface \*/

/\* **Endpoint IN1 Descriptor** \*/

0x07, /\* bLength: Endpoint Descriptor size \*/

USB\_ENDPOINT\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Endpoint \*/

0x81, /\* bEndpointAddress: (IN1) \*/

0x03, /\* bmAttributes: Interrupt \*/

0x10, 0x00, /\* wMaxPacketSize: \*/

0xFF, /\* bInterval: \*/

/\* **Data class interface descriptor** \*/

0x09, /\* bLength: Endpoint Descriptor size \*/

USB\_INTERFACE\_DESCRIPTOR\_TYPE, /\* bDescriptorType: \*/

0x02, /\* bInterfaceNumber: Number of Interface \*/

0x00, /\* bAlternateSetting: Alternate setting \*/

0x02, /\* bNumEndpoints: Two endpoints used \*/

0x0A, /\* bInterfaceClass: CDC \*/

0x00, /\* bInterfaceSubClass: \*/

0x00, /\* bInterfaceProtocol: \*/

0x00, /\* iInterface \*/

/\* **Endpoint IN3 Descriptor** \*/

0x07, /\* bLength: Endpoint Descriptor size \*/

USB\_ENDPOINT\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Endpoint \*/

0x83, /\* bEndpointAddress: (IN3) \*/

0x02, /\* bmAttributes: Bulk \*/

VCOM\_DATA\_SIZE, 0x00, /\* wMaxPacketSize: \*/

0x00, /\* bInterval \*/

/\* **Endpoint OUT3 Descriptor** \*/

0x07, /\* bLength: Endpoint Descriptor size \*/

USB\_ENDPOINT\_DESCRIPTOR\_TYPE, /\* bDescriptorType: Endpoint \*/

0x03, /\* bEndpointAddress: (OUT3) \*/

0x02, /\* bmAttributes: Bulk \*/

VCOM\_DATA\_SIZE, 0x00, /\* wMaxPacketSize \*/

0x00 /\* bInterval: ignore for Bulk transfer \*/

};

**Mass Storage和CDC复合设备的描述符：**

const uint8\_t MSDCDC\_ConfigDescriptor[] =

{

0x09, /\* bLength: Configuation Descriptor size \*/

0x02, /\* bDescriptorType: Configuration \*/

MSDCDC\_SIZ\_CONFIG\_DESC,

0x00,

0x03, /\* bNumInterfaces:3 interface \*/

0x01, /\* bConfigurationValue: Configuration value \*/

0x00, /\* iConfiguration: Index of string descriptor describing the configuration \*/

0xC0, /\* bmAttributes: bus powered \*/

0x32, /\* MaxPower 100 mA \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Descriptor of Mass Storage interface \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

0x09, /\* bLength: Interface Descriptor size \*/

0x04, /\* bDescriptorType: Interface descriptor type \*/

0x00, /\* bInterfaceNumber: Number of Interface \*/

0x00, /\* bAlternateSetting: Alternate setting \*/

0x02, /\* bNumEndpoints\*/

0x08, /\* bInterfaceClass: MASS STORAGE Class \*/

0x06, /\* bInterfaceSubClass : SCSI transparent\*/

0x50, /\* nInterfaceProtocol \*/

0x00, /\* iInterface: \*/

// Endpoint 5 Descriptor

0x07, /\*Endpoint descriptor length = 7\*/

0x05, /\*Endpoint descriptor type \*/

0x85, /\*Endpoint address (IN, address 1) \*/

0x02, /\*Bulk endpoint type \*/

0x40, /\*Maximum packet size (64 bytes) \*/

0x00,

0x00, /\*Polling interval in milliseconds \*/

// Endpoint 4 Descriptor

0x07, /\*Endpoint descriptor length = 7 \*/

0x05, /\*Endpoint descriptor type \*/

0x04, /\*Endpoint address (OUT, address 2) \*/

0x02, /\*Bulk endpoint type \*/

0x40, /\*Maximum packet size (64 bytes) \*/

0x00,

0x00, /\*Polling interval in milliseconds\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IAD \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

0x08, // bLength: Interface Descriptor size

0x0B, // bDescriptorType: IAD

0x01, // bFirstInterface

0x02, // bInterfaceCount

0x02, // bFunctionClass: CDC

0x02, // bFunctionSubClass

0x01, // bFunctionProtocol

0x04, //!! string descriptor for this interface

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Descriptor of cdc interface \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

0x09, // bLength: Interface Descriptor size

0x04, // bDescriptorType: Interface

0x01, // bInterfaceNumber: Number of Interface

0x00, // bAlternateSetting: Alternate setting

0x01, // bNumEndpoints: 3 endpoints used

0x02, // bInterfaceClass: Communication Interface Class

0x02, // bInterfaceSubClass: Abstract Control Model

0x01, // bInterfaceProtocol: Common AT commands

0x00, // iInterface:

//Header Functional Descriptor

0x04, // bLength: Endpoint Descriptor size

0x24, // bDescriptorType: CS\_INTERFACE

0x00, // bDescriptorSubtype: Header Func Desc

0x10, // bcdCDC: spec release number

//0x01,

//Call Managment Functional Descriptor

0x05, // bFunctionLength

0x24, // bDescriptorType: CS\_INTERFACE

0x01, // bDescriptorSubtype: Call Management Func Desc

0x00, // bmCapabilities: D0+D1

0x02, // bDataInterface: 1+1

//ACM Functional Descriptor

0x04, // bFunctionLength

0x24, // bDescriptorType: CS\_INTERFACE

0x02, // bDescriptorSubtype: Abstract Control Management desc

0x02, // bmCapabilities

//Union Functional Descriptor

0x05, // bFunctionLength

0x24, // bDescriptorType: CS\_INTERFACE

0x06, // bDescriptorSubtype: Union func desc

0x01, //!! bMasterInterface: Communication class interface

0x02, //!! bSlaveInterface0: Data Class Interface

//Endpoint 2 Descriptor

0x07, // bLength: Endpoint Descriptor size

0x05, // bDescriptorType: Endpoint

0x82, // bEndpointAddress: (IN2)

0x03, // bmAttributes: Interrupt

0x08, // wMaxPacketSize:

0x00,

0xFF, // bInterval:

//Data class interface descriptor

0x09, // bLength: Endpoint Descriptor size

0x04, // bDescriptorType:

0x02, // bInterfaceNumber: Number of Interface

0x00, // bAlternateSetting: Alternate setting

0x02, // bNumEndpoints: Two endpoints used

0x0A, // bInterfaceClass: CDC

0x00, // bInterfaceSubClass:

0x00, // bInterfaceProtocol:

0x00, // iInterface:

//Endpoint 3 Descriptor

0x07, // bLength: Endpoint Descriptor size

0x05, // bDescriptorType: Endpoint

0x03, // bEndpointAddress: (OUT3)

0x02, // bmAttributes: Bulk

0x40, // wMaxPacketSize:

0x00,

0x00, // bInterval: ignore for Bulk transfer

//Endpoint 1 Descriptor

0x07, // bLength: Endpoint Descriptor size

0x05, // bDescriptorType: Endpoint

0x81, // bEndpointAddress: (IN3)

0x02, // bmAttributes: Bulk

0x40, // wMaxPacketSize:

0x00,

0x00 // bInterval

};

**USB CDC + HID + IAD interface**

The descriptor belove, generated from MCP2200 device board. MCP2200 is programmed PIC18F14k50.   
And it look like composite device(IAD, HID +CDC). I have test it by pickit2 programmer.  
I know Windows xp sp3 or higher versions needed for IAD devices.   
Anyway, I need smilar descriptor and working PDS project . This descriptor may useful or not. I'm not sure.

1. USB Composite Device

|  |  |
| --- | --- |
| **Connection Status** | Device connected |
| **Current Configuration** | 1 |
| **Speed** | Full |
| **Device Address** | 1 |
| **Number Of Open Pipes** | 5 |

1. **Device Descriptor** MCP2200 USB Serial Port Emulator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 12h |  |
| 1 | bDescriptorType | 1 | 01h | Device |
| 2 | bcdUSB | 2 | 0200h | USB Spec 2.0 |
| 4 | bDeviceClass | 1 | EFh | Miscellaneous |
| 5 | bDeviceSubClass | 1 | 02h | Common Class |
| 6 | bDeviceProtocol | 1 | 01h | Interface Association Descriptor |
| 7 | bMaxPacketSize0 | 1 | 08h | 8 bytes |
| 8 | idVendor | 2 | 04D8h | Microchip Technology, Inc. |
| 10 | idProduct | 2 | 00DFh |  |
| 12 | bcdDevice | 2 | 0101h | 1.01 |
| 14 | iManufacturer | 1 | 01h | "Microchip Technology Inc." |
| 15 | iProduct | 1 | 02h | "MCP2200 USB Serial Port Emulator" |
| 16 | iSerialNumber | 1 | 03h | "0000133170" |
| 17 | bNumConfigurations | 1 | 01h |  |

1. **Configuration Descriptor 1** Bus Powered, 100 mA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 09h |  |
| 1 | bDescriptorType | 1 | 02h | Configuration |
| 2 | wTotalLength | 2 | 006Bh |  |
| 4 | bNumInterfaces | 1 | 03h |  |
| 5 | bConfigurationValue | 1 | 01h |  |
| 6 | iConfiguration | 1 | 00h |  |
| 7 | bmAttributes | 1 | 80h | Bus Powered |
|  | 4..0: Reserved |  | ...00000 |  |
|  | 5: Remote Wakeup |  | ..0..... | No |
|  | 6: Self Powered |  | .0...... | No, Bus Powered |
|  | 7: Reserved (set to one) (bus-powered for 1.0) |  | 1....... |  |
| 8 | bMaxPower | 1 | 32h | 100 mA |

1. **Interface Association Descriptor** Abstract Control Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 08h |  |
| 1 | bDescriptorType | 1 | 0Bh | Interface Association |
| 2 | bFirstInterface | 1 | 00h |  |
| 3 | bInterfaceCount | 1 | 02h |  |
| 4 | bFunctionClass | 1 | 02h | CDC Control |
| 5 | bFunctionSubClass | 1 | 02h | Abstract Control Model |
| 6 | bFunctionProtocol | 1 | 01h | AT Commands: V.250 etc |
| 7 | iFunction | 1 | 00h |  |

1. **Interface Descriptor 0/0** CDC Control, 1 Endpoint

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 09h |  |
| 1 | bDescriptorType | 1 | 04h | Interface |
| 2 | bInterfaceNumber | 1 | 00h |  |
| 3 | bAlternateSetting | 1 | 00h |  |
| 4 | bNumEndpoints | 1 | 01h |  |
| 5 | bInterfaceClass | 1 | 02h | CDC Control |
| 6 | bInterfaceSubClass | 1 | 02h | Abstract Control Model |
| 7 | bInterfaceProtocol | 1 | 01h | AT Commands: V.250 etc |
| 8 | iInterface | 1 | 00h |  |

1. **Header Functional Descriptor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bFunctionLength | 1 | 05h |  |
| 1 | bDescriptorType | 1 | 24h | CS Interface |
| 2 | bDescriptorSubtype | 1 | 00h | Header |
| 3 | bcdCDC | 2 | 0110h | 1.10 |

1. **Abstract Control Management Functional Descriptor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bFunctionLength | 1 | 04h |  |
| 1 | bDescriptorType | 1 | 24h | CS Interface |
| 2 | bDescriptorSubtype | 1 | 02h | Abstract Control Management |
| 3 | bmCapabilities | 1 | 06h |  |
|  | 7..4: Reserved |  | 0000.... |  |
|  | 3: Connection |  | ....0... |  |
|  | 2: Send Break |  | .....1.. | Send Break request supported |
|  | 1: Line Coding |  | ......1. | Line Coding requests and Serial State notification supported |
|  | 0: Comm Features |  | .......0 |  |

1. **Union Functional Descriptor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bFunctionLength | 1 | 05h |  |
| 1 | bDescriptorType | 1 | 24h | CS Interface |
| 2 | bDescriptorSubtype | 1 | 06h | Union |
| 3 | bControlInterface | 1 | 00h |  |
| 4 | bSubordinateInterface0 | 1 | 01h | CDC Data |

1. **Call Management Functional Descriptor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bFunctionLength | 1 | 05h |  |
| 1 | bDescriptorType | 1 | 24h | CS Interface |
| 2 | bDescriptorSubtype | 1 | 01h | Call Management |
| 3 | bmCapabilities | 1 | 00h |  |
|  | 7..2: Reserved |  | 000000.. |  |
|  | 1: Data Ifc Usage |  | ......0. | Call management only over Comm Ifc |
|  | 0: Call Management |  | .......0 | Does not handle call management itself |
| 4 | bDataInterface | 1 | 01h |  |

1. **Endpoint Descriptor 82** 2 In, Interrupt, 2 ms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 07h |  |
| 1 | bDescriptorType | 1 | 05h | Endpoint |
| 2 | bEndpointAddress | 1 | 82h | 2 In |
| 3 | bmAttributes | 1 | 03h | Interrupt |
|  | 1..0: Transfer Type |  | ......11 | Interrupt |
|  | 7..2: Reserved |  | 000000.. |  |
| 4 | wMaxPacketSize | 2 | 0008h | 8 bytes |
| 6 | bInterval | 1 | 02h | 2 ms |

1. **Interface Descriptor 1/0** CDC Data, 2 Endpoints

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 09h |  |
| 1 | bDescriptorType | 1 | 04h | Interface |
| 2 | bInterfaceNumber | 1 | 01h |  |
| 3 | bAlternateSetting | 1 | 00h |  |
| 4 | bNumEndpoints | 1 | 02h |  |
| 5 | bInterfaceClass | 1 | 0Ah | CDC Data |
| 6 | bInterfaceSubClass | 1 | 00h |  |
| 7 | bInterfaceProtocol | 1 | 00h |  |
| 8 | iInterface | 1 | 00h |  |

1. **Endpoint Descriptor 03** 3 Out, Bulk, 32 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 07h |  |
| 1 | bDescriptorType | 1 | 05h | Endpoint |
| 2 | bEndpointAddress | 1 | 03h | 3 Out |
| 3 | bmAttributes | 1 | 02h | Bulk |
|  | 1..0: Transfer Type |  | ......10 | Bulk |
|  | 7..2: Reserved |  | 000000.. |  |
| 4 | wMaxPacketSize | 2 | 0020h | 32 bytes |
| 6 | bInterval | 1 | 00h |  |

1. **Endpoint Descriptor 83** 3 In, Bulk, 64 bytes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 07h |  |
| 1 | bDescriptorType | 1 | 05h | Endpoint |
| 2 | bEndpointAddress | 1 | 83h | 3 In |
| 3 | bmAttributes | 1 | 02h | Bulk |
|  | 1..0: Transfer Type |  | ......10 | Bulk |
|  | 7..2: Reserved |  | 000000.. |  |
| 4 | wMaxPacketSize | 2 | 0040h | 64 bytes |
| 6 | bInterval | 1 | 00h |  |

1. **Interface Descriptor 2/0** HID, 2 Endpoints

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 09h |  |
| 1 | bDescriptorType | 1 | 04h | Interface |
| 2 | bInterfaceNumber | 1 | 02h |  |
| 3 | bAlternateSetting | 1 | 00h |  |
| 4 | bNumEndpoints | 1 | 02h |  |
| 5 | bInterfaceClass | 1 | 03h | HID |
| 6 | bInterfaceSubClass | 1 | 00h |  |
| 7 | bInterfaceProtocol | 1 | 00h |  |
| 8 | iInterface | 1 | 00h |  |

1. **HID Descriptor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 09h |  |
| 1 | bDescriptorType | 1 | 21h | HID |
| 2 | bcdHID | 2 | 0111h | 1.11 |
| 4 | bCountryCode | 1 | 00h |  |
| 5 | bNumDescriptors | 1 | 01h |  |
| 6 | bDescriptorType | 1 | 22h | Report |
| 7 | wDescriptorLength | 2 | 001Dh | 29 bytes |

1. **Endpoint Descriptor 81** 1 In, Interrupt, 1 ms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 07h |  |
| 1 | bDescriptorType | 1 | 05h | Endpoint |
| 2 | bEndpointAddress | 1 | 81h | 1 In |
| 3 | bmAttributes | 1 | 03h | Interrupt |
|  | 1..0: Transfer Type |  | ......11 | Interrupt |
|  | 7..2: Reserved |  | 000000.. |  |
| 4 | wMaxPacketSize | 2 | 0010h | 16 bytes |
| 6 | bInterval | 1 | 01h | 1 ms |

1. **Endpoint Descriptor 01** 1 Out, Interrupt, 1 ms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bLength | 1 | 07h |  |
| 1 | bDescriptorType | 1 | 05h | Endpoint |
| 2 | bEndpointAddress | 1 | 01h | 1 Out |
| 3 | bmAttributes | 1 | 03h | Interrupt |
|  | 1..0: Transfer Type |  | ......11 | Interrupt |
|  | 7..2: Reserved |  | 000000.. |  |
| 4 | wMaxPacketSize | 2 | 0010h | 16 bytes |
| 6 | bInterval | 1 | 01h | 1 ms |

1. **Interface 2 HID Report Descriptor** Vendor-Defined 1

|  |  |
| --- | --- |
| Usage Page (Vendor-Defined 1) | 06 00 FF |
| Usage (Vendor-Defined 1) | 09 01 |
| Collection (Application) | A1 01 |
| Usage Minimum (Vendor-Defined 1) | 19 01 |
| Usage Maximum (Vendor-Defined 16) | 29 10 |
| Logical Minimum (0) | 15 00 |
| Logical Maximum (255) | 26 FF 00 |
| Report Size (8) | 75 08 |
| Report Count (16) | 95 10 |
| **Input** (Data,Ary,Abs) | 81 00 |
| Usage Minimum (Vendor-Defined 1) | 19 01 |
| Usage Maximum (Vendor-Defined 16) | 29 10 |
| **Output** (Data,Ary,Abs,NWrp,Lin,Pref,NNul,NVol,Bit) | 91 00 |
| End Collection | C0 |

As I understood from all googling about composite devices, two interfaces usually mean that in Windows device manager

we will see composite device and two other devices that corespond to each interface.

If you need one device, then most likely you will need to use device class 0xEF and USB Interface Association Descriptor,

but I'm not sure if it will do what you need.

<http://msdn.microsoft.com/en-us/library/windows/hardware/ff540054(v=vs.85).aspx>

<http://www.microsoft.com/taiwan/whdc/archive/iad.mspx>

**usbccgp.sys hangs Windows to death when a descriptor is malformed?**

Yes, it's true. Even for full-speed and low-speed devices. So double-check your descriptors:

* The first Configuration Descriptor has ID #1
* The first Interface Descriptor has ID #0 and are contiguous numbered.   
  The Interface count in Configuration Descriptor must be set correctly.
* The first Alternate Setting has ID #0 and are contiguous numbered

Problems arise when you change your single-interface USB device into a multi-interface (multi-function) while keeping your USB VID+PID pair!

Even when your change or add some more USB interface with the same VID+PID pair, problems occur.

The Windows behaviour of assigning a driver is as follows:

* If the VID+PID is already known, Windows loads the known driver regardless whether it's a multi-function device or not.   
  (I.e. when you loaded a driver before for the non-multifunction device with PnP ID e.g. USB\VID\_16C0&PID\_06B3, that driver is always used.)   
  As a consequence, your driver will fail (typically with a blue screen) when it is surprisingly bound to the - now - multifunction USB device.
* If VID+PID is not known, Windows tries to load a Class Driver.   
  This is "usbccgp.sys" for the top of multifunction devices which is a very special class (not USB\Class\_00).
* Windows remembers the Class-Driver-To-VID+PID(+MI)-binding for the next plug-in event persistently, probably for performance purposes.   
  Therefore, when you swap your Interfaces keeping VID+PID, you earn errors.   
  But you can get rid easily by uninstalling your device in Device Manager. This will remove the memorized bindings.
* "usbccgp.sys" itself retries the enumeration process for the child devices in the same way,   
  i.e. preferes VID+PID+MI bindings over USB Class bindings.
* In case of no match, no driver is loaded, and the entry is marked in Device Manager.

So you may conclude:

* VID+PID, and MI numbers (like USB\VID\_16C0&PID\_06B3&REV\_4621&MI\_00) have precedence,   
  allowing manufacturers to deliver their own multifunction device drivers (replacing "usbccgp.sys")
* USB CLASS bindings (like USB\Class\_07) have less precedence, allowing manufacturers   
  to deliver their own specialized device driver replacing one of the system-supplied class drivers (like "usbprint.sys")
* REV numbers don't matter Windows, so manufacturers are free to (mis)use this entry,  
  and there is no option to get more possible USB device bindings "for free" using the REVnumber as an extra binding criterion

An example:

This INF file snippet is wrong because you have no control whether your driver is bound to the Interface #0   
or the entire USB device (except your driver is very smart and can handle this):

* %DevDesc%=Dev,USB\VID\_16C0&PID\_06B3&REV\_4621&MI\_00  
  %DevDesc%=Dev,USB\VID\_16C0&PID\_06B3&REV\_4620

(Windows will work as expected most of the time, but users are able to bind multi-function devices to entire devices,   
and produce overwhelming trouble with your support.)

As a result:

If you change your single-interface USB device to multi-interface and vice versa keeping VID+PID,   
make sure that you cleaned your Device Manager entries beforehand.   
Or at least before you plug your changed device.

Google for **devmgr\_show\_nonpresent\_devices**and **devmgr\_show\_details**to get a clue.

Better you change VID+PID too, providing you have a free pair available.

If you are in doubt offering a single-interface or multi-interface USB device,

simply decide to multi-interface at the first step,

and implement an extra HID or CDC interface to make usbccgp.sys happy.

It's useful for debugging purposes later. At least two childs are needed.

Note that CDC interface (which is itself a composite USB device by design and handled in a special way by usbccgp.sys)

will not run properly in elsewhere-multifunction devices on Windows 2000 and earlier.

[**USB Composite Device**](http://blog.siliconlabs.com/t5/8-bit-MCU/USB-Composite-Device/td-p/70519)

USB composite device  
  
Examples  
  
[USB\_CDC\_HID\_IAD\_10.zip](http://homepage2.nifty.com/Notus/USB_CDC_HID_IAD_10.zip)

This example implements a composite device of CDC (with IAD) and HID

The CDC interface is based on the code of [USB CDC implementation for 'F32x and 'F34x](http://www.cygnal.org/ubb/Forum9/HTML/000945.html) In either interface, CDC and HID, the device loops back the OUT transfer to IN.  
  
Passed USBCV R1.3.1, the ch9 and HID test  
Tested on these Windows version,  
- Windows Vista SP1  
- WinXP SP3 (usbser.sys: 5.1.2600.5508, usbccgp.sys: 5.1.2600.5512)  
  
On the original WinXP SP2 (usbser.sys, usbccgp.sys: 5.1.2600.2180),  
the HID interface of the device works well, but CDC causes BSOD when a test app opens the VCP.  
  
[USB\_HID\_composite\_01.zip](http://homepage2.nifty.com/Notus/USB_HID_composite_01.zip)

This example implements three HID interfaces to SiLabs 'F320/'F340 DK.  
Each interface has an interrupt IN EP and an interrupt OUT EP.  
The firmware loops back the data sent to the OUT EP to the IN EP, on each interface.  
  
Passed the USB compliance test, USBCV 1.3, ch9 and HID test.  
Tested on HClient host app example on WinDDK.  
  
[USB\_Mouse\_INT\_01.zip](http://homepage2.nifty.com/Notus/USB_Mouse_INT_01.zip)

This is a demonstration which shows a composite device combined SiLabs USB\_HID\MouseExample and USB\_Interrupt (USB\_INT)

* When you are asked a device driver, specify the INF folder attached to this zip file.
* USB\_Interrupt host application and device driver works with this implementation without any change.

**1. What is composite device**

Composite device is defined in the USB spec as follows (usb\_20.pdf 5.2.3),  
A device that has multiple interfaces controlled independently of each other is referred to as a composite device.  
Using composite device, multiple functions are combined into a single device.  
Ex.  
- Keyboard + Mouse  
- Video + USB Hard disk  
- I/O device (HID + USB\_bulk)  
  
Another advantage of composite device is that it eases the device driver development.  
OS assigns a separate device driver to each Interface of the composite device as follows.   
Therefore, a dedicated monolithic driver is not required for newly designed device;   
you can realize it using existing drivers.

Code:

+----------------------------+ +----------------------  
 | Composite Device | | Host PC  
 | | |  
 | Function 0 -- Interface 0 --------- Device driver A <--->  
 | | |  
 | Function 1 -- Interface 1 --------- Device driver B <--->  
 +----------------------------+ +-----------------------

When OS has some required drivers as built-in, they are available for the composite device.

These OS built-in device drivers are called as USB class driver.  
Approved Class Specification Documents from USB.org  
<http://www.usb.org/developers/devclass_docs#approved>  
  
Windows have these built-in class drivers.  
USB FAQ: Introductory Level - USB Class Drivers from MS WHDC  
<http://www.microsoft.com/whdc/system/bus/usb/USBFAQ_intro.mspx>  
  
Please note, available drivers for a composite device are not limited only to class drivers.

Any driver can be applied, as long as it doesn't require a device class (class defined in device descriptor).

For example, SiLabs USB\_INT and USB\_bulk drivers are also applicable for composite devices.

**2. How Windows handles a composite device**

When a device satisfies these three requirement, Windows system recognizes the device as a composite device.

1. The class field of the device descriptor equals to zero: (bDeviceClass) = (0x00)
2. Single Configuration
3. Multiple Interfaces

[Note]  
WinXP SP2, Win2k3 SP1 and Vista supports this alternative requirement.  
1'. The class, subclass and protocol fields of the device are that of Interface Association Descriptor:  
(bDeviceClass, bDeviceSubClass, bDeviceProtocol) = (0xEF, 0x02, 0x01)

When an USB device is plugged in to a PC, the system reads out the device descriptor of the device and makes these Device ID.

USB\VID\_vvvv&PID\_pppp  
USB\VID\_vvvv&PID\_pppp&REV\_rrrr  
(vvvv, pppp, rrrr: four digit hex number for the VID, PID, device release number.   
Matches to idVendor/ idProduct/ bcdDevice, defined in the device descriptor)

The system searches device database on the registry, installed by INF files.   
When the system finds the Device ID in a device record, it assigns the device driver specified by the record.   
However, when it cannot find any matched record, and the device Configuration satisfies above criteria,   
**the generic composite parent driver is loaded.**   
This parent driver parses the Configuration of the device, and assigns this Device ID to each Interfaces.

USB\VID\_vvvv&PID\_pppp&MI\_mm  
USB\VID\_vvvv&PID\_pppp&REV\_rrrr&MI\_mm  
(mm: Interface number of the corresponding function, two digit hex number)

As of the Interface which specifies a class, the system also assigns this Compatible ID.

USB\CLASS\_cc  
USB\CLASS\_cc&SUBCLASS\_ss  
USB\CLASS\_cc&SUBCLASS\_ss&PROT\_pp  
(cc/ ss / pp: two digits hex number. bInterfaceClass/ bInterfaceSubClass/ bInterfaceProtocol, **from the Interface descriptor**)

The system searches these Device ID and Compatible ID in the device database again.   
When it finds a matched record, it assigns the specified device driver to the Interface.   
However, when it cannot find any matched record, it shows 'New Hardware Wizard' to users and asks them to install the device driver.  
  
Enumeration of the Composite Parent Device from MSDN  
<http://msdn2.microsoft.com/en-us/library/aa476434.aspx>

**3. Implementation**

On USB application, firmware, device driver and host application are closely related.   
It is desirable to make separate prototypes of firmwares and host applications for each Interface first.   
After confirming them to work properly, combine them together.

**3.1 Device firmware**

When the source code is well organized, modification of the firmware is easy.   
Based on one of the prototypes, copy a part of code from other prototypes and insert it to corresponding part of the base source code.  
In other word, the source codes should be organized considering to make it a composite device.   
When these parameters are defined by #define macro or enum instead of direct number in each prototype,   
the combination of prototypes is done smoothly.

* Interface number
* Endpoint address
* Endpoint status (IDLE / HALT)

3.1.1 Descriptors

3.1.1.1 Device descriptor

* **bDeviceClass: Must be assigned to zero**
* idVendor, idProduct: VID/PID must be unique, to avoid conflict to other devices.

3.1.1.2 Configuration descriptor

* wTotalLength: The total number of bytes of Configuration set, includes all of Interface sets
* bNumInterfaces: Number of Interfaces included in this Configuration

Configuration set means these descriptors, for example.

Configuration descriptor  
- Interface descriptor (0)  
- - accessory descriptor, such as HID class descriptor, if any  
- - Endpoint descriptors  
- Interface descriptor (1)  
- - accessory descriptor, such as HID class descriptor, if any  
- - Endpoint descriptors

HID report descriptor and String descriptors are not included in the Configuration set.  
  
3.1.1.3 Interface descriptors

* bInterfaceNumber: Index of Interfaces, starting from zero
* bInterfaceClass: Specify class code for this Interface

If any specific class code is not assigned to the Interface, set bInterfaceClass to 0xFF (Vendor specific). 0x00 (Reserved) would work, but 0xFF is better.  
  
3.1.1.4 Endpoint descriptors

* bEndpointAddress: must be unique on each Endpoint descriptor

Any duplicated Endpoint across Interfaces is not allowed in a composite device.

**3.1.2 Standard requests**

3.1.2.1 Additional Descriptor handling  
Get\_Descriptor must support additional descriptors asked by the host.

* Configuration descriptor: return full Configuration set (see 3.1.1.2)
* Class-specific descriptors:

When the descriptor type in request (MSB of wValue) is not Device(1), Configuration(2) or String(3),  
the request may be for class-specific descriptor (in full-speed devices).   
In this case, wIndex field of the Setup data shows the Interface number in question.   
According to the class specified by the Interface (wIndex),   
Get\_Descriptor must return the class-specific descriptor specified by the MSB of wValue.  
When the class supports Set\_Descriptor request, it must be handled similarly to Get\_Descriptor.  
Interface and Endpoint descriptors cannot be directly accessed with Get\_Descriptor or Set\_Descriptor.  
Therefore, Get\_Descriptor and Set\_Descriptor have no need to support them.

3.1.2.2 Additional Interfaces handling  
wIndex field of the Setup data shows the Interface number in question.   
When this Interface number matches to the additional Interfaces, handle the requests as follows.

* Get\_Status: return Zero
* Get\_Interface: return current alternate Interface number
* Set\_Interface: set current alternate Interface to one specified by the request

When the Interface in question doesn't have any alternate Interface, Get\_Interface returns Zero.   
And Set\_Interface succeeds when the wValue is Zero, otherwise return STALL.  
  
3.1.2.3 Additional Endpoints handling

* Get\_Status: return HALT condition of the Endpoint
* Clear\_Feature: recover the Endpoint from HALT
* Set\_Feature: set the Endpoint to HALT
* Set\_Configuration: Setup additional Endpoints

When Get\_Status, Clear\_Feature and Set\_Feature are issued to an Endpoint, wIndex field of the Setup data indicates the Endpoint address.  
When the Interfaces doesn't have any alternate Interface, set up the Endpoints in Set\_Configuration request.   
As of the Interface with any alternate Interface, set up the Endpoints belonging to the Interface in Set\_Interface.

**3.1.3 Class-specific requests**

wIndex field of the Setup data of the request indicates the Interface to which this request is issued.   
Therefore, dispatch the request by wIndex first, and copy the each handler for class-specific requests from the prototypes under each case.

**3.1.4 Endpoint handling**

When the Endpoint address and Endpoint status are defined by macro,   
modification on this part finishes by copying the Endpoint handler of each prototype to the base one.

**3.2 Device driver and host application**

OS built-in class drivers are designed to work with composite devices.   
In most case, these drivers are applicable to composite devices without any change of default INF file.   
However, device drivers provided by vendors are not always designed to work with composite devices.   
INF file and host application code should be modified for these drivers.   
The device driver itself should work without any change.   
Of course, rare exception may exist.

**3.2.1 INF file**

When the device driver requires an INF file even for single use, the INF file is required as a part of composite device.   
The INF file defines the Device ID as follows.

USB\VID\_vvvv&PID\_pppp  
USB\VID\_vvvv&PID\_pppp&REV\_rrrr

For a composite device, the Interface number must be added to this Device ID.

USB\VID\_vvvv&PID\_pppp&MI\_mm  
USB\VID\_vvvv&PID\_pppp&REV\_rrrr&MI\_mm

For example, when you add the USB\_Bulk function (Interface with bulk IN/OUT Endpoints)   
to your composite device as the Interface number 1, the Device ID in the INF file (SilabsBulk.inf) is modified as follows.

USB\VID\_vvvv&PID\_pppp&MI\_01  
(vvvv, pppp: VID/PID must be unique)

**3.2.2 Endpoint address and pipe name / device path name**

When two or more devices are combined into a single composite device,   
the Endpoint addresses must be often changed to fit to the newly designed device.   
Usually, the pipe name and device path name from device drivers are designed to hide the Endpoint address.   
Therefore, in most case, Endpoint address reassignment doesn't affect to the host application.  
  
However, there are some drivers which expose the Endpoint address directly to the pipe name.   
For these drivers, the host application must be modified to reflect the Endpoint address assignment.  
Confirmation for this point is desirable before planning a new device.

* OS built-in class drivers hide Endpoint address behind its device path name.
* MS WinDDK bulkusb and isousb example driver hide Endpoint address behind the pipe name (a part of device path name).
* SiLabs USB\_INT and USB\_Bulk device drivers hide it behind the pipe name (a part of device path name).
* Cypress ezusb.sys driver hides Endpoint address behind its pipe number.
* Cypress CyUSB.sys driver exposes Endpoint address directly.   
  But when the code follows the example of CCyUSBDevice::BulkInEndPt in CyAPI.chm,   
  the Endpoint address is hidden behind the index of the Endpoint array.

When a device applies the same class to multiple Interfaces, the host application should be modified to distinguish these Interfaces.

**Enumeration of USB Composite Devices**

When a new USB device is connected to a host machine, the USB bus driver creates a physical device object (PDO)

for the device and generates a PnP event to report the new PDO.   
The operating system then queries the bus driver for the hardware IDs associated with the PDO.

For all USB devices, the USB bus driver reports a [device ID](https://msdn.microsoft.com/en-us/library/ff556277.aspx#wdkgloss.device_id) with the following format:

USB\VID\_xxxx&PID\_yyyy

**Note**  xxxx and yyyy are taken directly from **idVendor** and **idProduct** fields of the device descriptor, respectively.

The bus driver also reports a compatible identifier (ID) of USB\COMPOSITE, if the device meets the following requirements:

* The device class field of the device descriptor (**bDeviceClass**) must contain a value of zero,   
  or the class (**bDeviceClass**), subclass (**bDeviceSubClass**), and protocol (**bDeviceProtocol**) fields   
  of the device descriptor must have the values 0xEF, 0x02 and 0x01 respectively,   
  as explained in [USB Interface Association Descriptor](https://msdn.microsoft.com/en-us/library/ff540054.aspx).
* The device must have multiple interfaces.
* The device must have a single configuration.

The bus driver also checks the device class (**bDeviceClass**), subclass (**bDeviceSubClass**), and protocol (**bDeviceProtocol**) fields of the device descriptor.

If these fields are zero, the device is a composite device, and the bus driver reports an extra compatible identifier (ID) of USB\COMPOSITE for the PDO.

After retrieving the hardware and compatible IDs for the new PDO, the operating system searches the INF files.

If one of the INF files contains a match for the device ID, Windows loads the driver that is indicated by that INF file   
and the generic parent driver does not come into play.

If no INF file contains the device ID, and the PDO has a compatible ID, Windows searches for the compatible ID.

This produces a match in Usb.inf and causes the operating system to load the [USB Generic Parent Driver (Usbccgp.sys)](https://msdn.microsoft.com/en-us/library/ff539234.aspx).

If you want the generic parent driver to manage your device, but your device does not have the characteristics necessary   
to ensure that the system will generate a compatible ID of USB\COMPOSITE,   
you will have to provide an INF file that loads the generic parent driver.

The INF file should contain a needs/includes section that references Usb.inf.

If your composite device has multiple configurations, the INF file you provide must specify which configuration the generic parent should use in the registry.

The necessary registry keys are described in [Configuring Usbccgp.sys to Select a Non-Default USB Configuration](https://msdn.microsoft.com/en-us/library/ff538059.aspx).

**USB Generic Parent Driver (Usbccgp.sys)**

 This section describes the Usbccgp.sys driver provided by Microsoft for composite devices.

Many USB devices expose multiple *USB interfaces*.

In USB terminology, these devices are called *composite devices*.

Microsoft Windows 2000 and Windows 98 operating systems include a generic parent facility in the USB bus driver

(Usbhub.sys) that exposes each interface of the composite device as a separate device.

In Microsoft Windows XP and Windows Me, this facility is streamlined and improved by transferring it

to an independent driver called the *USB generic parent driver*(Usbccgp.sys).

Using the features of the generic parent driver, device vendors can make selective use of Microsoft-supplied driver support for some interfaces.

The interfaces of some composite device operate independently.

For example, a composite USB keyboard with power buttons might have one interface for the keyboard

and another interface for the power buttons.

The USB generic parent driver enumerates each of these interfaces as a separate device.

The operating system loads the Microsoft-supplied keyboard driver to manage the keyboard interface, a

nd the Microsoft-supplied power keys driver to manage the power keys interface.

If the composite device has an interface that is not supported by native Windows drivers,

the vendor of the device should provide a driver for the interfaces and an INF file.

The INF file should have an INF *DDInstall* section that matches the device ID of interface. T

he INF file must not match the device ID for the composite device itself, because this prevents the generic parent driver from loading.

For an explanation of how the operating system loads the USB generic parent driver, see

[**Enumeration of USB Composite Devices**](https://msdn.microsoft.com/en-us/library/ff537109.aspx)**.**

Some devices group interfaces into *interface collections* that work together to perform a particular *function*.

When interfaces are grouped in interface collections, the generic parent driver treats each collection,

rather than each individual interfaces, as a device.

For more information on how the generic parent driver manages interface collections,

see [Enumeration of Interface Collections on USB Composite Devices](https://msdn.microsoft.com/en-us/library/ff538828.aspx).

After the operating system loads the client drivers for the interfaces of a composite device,

the generic parent driver multiplexes the data flow from the client drivers,

combining these separate interactions into a single data stream for the composite device.

The generic parent is power policy owner for the entire composite device and all of its interfaces.

It also manages synchronization and PnP requests.

The generic parent driver can simplify the task for vendors of composite hardware,

if Microsoft-supplied drivers support some interfaces but not others.

Vendors of such devices need only supply drivers for the unsupported interfaces,

because the generic parent driver facilitates the use of Microsoft-supplied drivers for the supported interfaces.

The following sections describe the features and functions of the generic parent driver:

[Enumeration of USB Composite Devices](https://msdn.microsoft.com/en-us/library/ff537109.aspx)

[Descriptors on USB Composite Devices](https://msdn.microsoft.com/en-us/library/ff537071.aspx)

[Enumeration of Interfaces on USB Composite Devices](https://msdn.microsoft.com/en-us/library/ff537102.aspx)

[Enumeration of Interface Collections on USB Composite Devices](https://msdn.microsoft.com/en-us/library/ff538828.aspx)

[Content Security Features in Usbccgp.sys](https://msdn.microsoft.com/en-us/library/ff537063.aspx)

**Enumeration of Interface Collections on USB Composite Devices**

 Interfaces on a composite USB device can be grouped in collections. The [USB Generic Parent Driver (Usbccgp.sys)](https://msdn.microsoft.com/en-us/library/ff539234.aspx) can enumerate interface collections in four ways.

These four methods of enumeration of interface collections are arranged hierarchically in the following manner:

1. **Vendor-supplied callback routines**

If the vendor has registered a callback routine with the [USB Generic Parent Driver (Usbccgp.sys)](https://msdn.microsoft.com/en-us/library/ff539234.aspx), the generic parent driver gives precedence to the callback routine, and allows the callback routine to group interfaces rather than using some other method. For more information on the enumeration of interface collection using vendor-supplied callback routines, see [Customizing Enumeration of Interface Collections for Composite Devices](https://msdn.microsoft.com/en-us/library/ff537066.aspx).

1. **Union Functional Descriptors**

. If the vendor has enabled CDC and WMCDC enumeration in the USB generic parent driver, the generic parent driver uses *union functional descriptors* (UFDs) to group interfaces into collections. When enabled, this method has precedence over all other methods, except for vendor-supplied callback routines. For more information on the enumeration of devices with UFDs, see [Support for Wireless Mobile Communications Device Class](https://msdn.microsoft.com/en-us/library/ff538833.aspx).

1. **Interface Association Descriptors**

If *interface association descriptors* (IADs) are present, the USB generic parent driver always groups interfaces by using IADs rather than by using legacy methods. Microsoft recommends that vendors use IADs to define interface collections. For more information on the enumeration of devices with IADs, see[Enumeration of Interface Collections on USB Devices with IADs](https://msdn.microsoft.com/en-us/library/ff537107.aspx).

1. **Legacy audio method.**

The USB generic parent driver is able to enumerate interface collections by using legacy techniques that are reserved for audio functions. The generic parent driver does not use this method if there are any IADs on the device. For more information on the legacy audio method of enumeration, see [Enumeration of Interface Collections on Audio Devices without IADs](https://msdn.microsoft.com/en-us/library/ff537105.aspx).

**Descriptors on USB Composite Devices**

 As described by the USB specification, every USB device provides a set of hierarchical descriptors that define its functionality. At the top level, each device has one or more USB configuration descriptors, each of which has one or more interface descriptors. For further information about USB configuration descriptors, see [USB Configuration Descriptors](https://msdn.microsoft.com/en-us/library/ff539242.aspx). Configurations are mutually exclusive, so only one configuration can be selected to operate at a time.

Prior to Windows Vista, Microsoft-supplied drivers only select configuration 1. In Windows Vista and the later versions of Windows, you can set a registry value to specify which configuration the [USB Generic Parent Driver (Usbccgp.sys)](https://msdn.microsoft.com/en-us/library/ff539234.aspx) will use. For more information about selecting the device configuration on composite devices, see [How to select a configuration for a USB device](https://msdn.microsoft.com/en-us/library/gg615081.aspx).

Within a configuration, interfaces and interface collections are managed independently. Each interface is represented, at the descriptor level, by a unique value in the**bInterfaceNumber** member of its [**USB\_INTERFACE\_DESCRIPTOR**](https://msdn.microsoft.com/en-us/library/ff540065.aspx) structure.

The function of an interface is indicated by the **bInterfaceClass**, **bInterfaceSubClass**, and **bInterfaceProtocol** members of the same structure, along with the class-specific descriptors that might follow it.

For more information on descriptors, see [USB Descriptors](https://msdn.microsoft.com/en-us/library/ff539267.aspx).

**Enumeration of Interfaces on USB Composite Devices**

Interfaces on a composite USB device can be grouped in collections or represent one USB function individually.

When the interfaces are not grouped in collections, the generic parent driver creates a PDO

for each interface and generates a set of hardware IDs for each PDO.

The [*device ID*](https://msdn.microsoft.com/en-us/library/ff556277.aspx#wdkgloss.device_id) for an interface PDO has the following form:

USB\VID\_v(4)&PID\_p(4)&MI\_z(2)

In these IDs:

* *v(4)* is the four-digit vendor code that the USB standards committee assigns to the vendor.
* *p(4)* is the four-digit product code that the vendor assigns to the device.
* *z(2)* is the interface number that is extracted from the **bInterfaceNumber** field of the interface descriptor.

The generic parent driver also generates the following compatible IDs by using the information

from the interface descriptor ([**USB\_INTERFACE\_DESCRIPTOR**](https://msdn.microsoft.com/en-us/library/ff540065.aspx)):

USB\CLASS\_d(2)&SUBCLASS\_s(2)&PROT\_p(2)

USB\CLASS\_d(2)&SUBCLASS\_s(2)

USB\CLASS\_d(2)

In these IDs:

* *d(2)* is the class code (**bInterfaceClass**)
* *s(2)* is the subclass code (**bInterfaceSubClass**)
* *p(2)* is the protocol code (**bInterfaceProtocol**)

Each of these codes is a four-digit number.

**USB common class generic parent driver (Usbccgp.sys)**

The USB common class generic parent driver is the Microsoft-provided parent driver for composite devices.

The hub driver enumerates and loads the parent composite driver if **deviceClass** is 0 or 0xef and **numInterfaces** is greater than 1 in the device descriptor.

The hub driver generates the compatible ID for the parent composite driver as "USB\COMPOSITE".

Usbccgp.sys uses Windows Driver Model (WDM) routines.

The parent composite driver enumerates all functions in a composite device and creates a PDO for each one.

This causes the appropriate class or client driver to be loaded for each function in the device.

Each function driver (child PDO) sends requests to the parent driver, which submits them to the USB hub driver.

Usbccgp.sys is included with Windows XP with SP1 and later versions of the Windows operating system.

In Windows 8, the driver has been updated to implement function suspend and remote wake-up features as defined in the USB 3.0 specification.

For more information, see [USB Generic Parent Driver (Usbccgp.sys)](https://msdn.microsoft.com/en-us/library/ff539234.aspx).

**USB Composite Device Operation**

As a new member, let me say thanks to all of you for all of the valuable posts.

Without the help of this forum, it would have taken me much more time to get up to speed.  
  
I'm trying to make a composite USB device based off of the Microchip CDC firmware and the HID firmware.

I believe I've found two ways to do this since the CDC firmware was written to use two interfaces.  
  
1. The first is to group all three endpoints under the first interface by changing CDC\_DATA\_INTF\_ID to point to the first interface.

The device descriptor can then have the class, subclass, and protocol set to 0x00 to allow each interface to specify a different driver.

This works fine with just the CDC class in it.

As soon as a second class (HID) is added to the CDC class that was working alone,

Windows XP recognizes the device as a "Composite Device".

As such, the device hardware ID changes to USB\Vid\_xxxx&Pid\_yyyy&MI\_ww format.    
  
2. The second way I believe is to use the Interface Association Descriptor to group the two interfaces together.

This just became available with Windows XP SP2.

This allows for a generic “Composite Device” class, subclass, and protocol to be used.

The driver can then be assigned at the associated interface level allowing mixed classes in a device.

When using the IAD, the device hardware ID changes to USB\Vid\_xxxx&Pid\_yyyy&MI\_ww format.  
  
When using the first method, the PIC only sometimes gets data from HyperTerminal and never is able to send data to HyperTerminal from the PIC.

Using the second method I am unable to get any communication between the PIC and HyperTerminal to work.

In both cases, after modifying the driver to accept the new hardware ID, the drivers installed fine and appear to be function correctly.

Using a software USB sniffer, the PIC seems to respond to most commands during enumeration, but has issues when it polls the pic to for data.  
  
I suspect I am just not handling the new Composite Device communications correctly or I have not merged the two demos correctly.

Unfortunately, I haven’t found much documentation on either.

I’ve ordered an ICD2 to aid with debugging, but with the fast timeouts on the USB, this may not be as useful as I’d like.    
   
The next thing I am going to try is to combine the Demo code with the HID code

into a composite device as both of them already use a single interface and are already configured

to specify the class, subclass, and protocol at the interface level.

Perhaps this will be easier as I’ve seen some postings on other forums that didn’t think that the CDC driver could easily be part of a composite device.  
  
Has anyone else used composite device firmware for which I could see some example code?    
   
Where do you recommend that I could find more information about how to work with Composite Devices with the PIC?

(USB Complete is already on its way.)  I’ve searched the forum, but wasn’t able to find much on Composite Devices or anything on IAD.  
  
Thanks for any pointers you may have!  
  
-John  
   
Windows XP Pro SP2  
MPLAB v7.41, C18 v3.02  
Homebuilt PICDEM FS USB Demo Board  
18F4550 running Microchip’s USB Boot loader

First, thanks to everyone for their posts, as they have been very helpful.

Special thanks to ‘sirnoname’ who pointed me in the right direction with their post.

I am currently developing a USB Composite Device with a Serial Port (CDC) and Mass Storage Device (MSD) target for use with Windows XP.  
   
Let me first say that “theory” would suggest that the right approach is to use the IAD as a way to associate the two interfaces a CDC requires.

However after following all the information in the specs and several examples I simply could not get this to work.

By everything I have read I think it “should” work, but it doesn’t.

I have a current open ticket with Microsoft regarding this issue, but I am not going to hold my breath.

It appears that Windows is either not handling the IAD correctly or the usbser.sys driver was not built to use it.

Either case, it doesn’t work.

I also tried installing the HotFix described at <http://support.microsoft.com/kb/918365> but this does not help either.

We’ll see if Microsoft comes up with anything.  
   
In any case using the suggestions from ‘sirnoname’ I was able to pull this off however,

but not in a way I would have expected. Below is what I needed to do to get my device to enumerate properly with Windows.  
   
The key was to define the CDC interface as a single interface with 3 Endpoints

(one interrupt endpoint for notification, 1 endpoint for bulk-in and 1 endpoint for bulk-out).

EP0 is always assumed.

This is contrary to what you normally see.

Normally you see examples with a COMMUNICATIONS\_CLASS interface with 1 endpoint (notification)

and a second DATA interface with 2 end-points.

So I am not sure why my configuration works, I am just happy that it does.

Perhaps someone can shed some light as to why this works.  
   
Anyway here it goes;  
   
1)       Device Descriptor  
   
You must define the device descriptor as so:  
   
Device Descriptor   
bLength                                     0x12   
bcdUSB                                    0x200   
bDeviceClass                             0x0   
bDeviceSubClass                       0x0   
bDeviceProtocol                         0x0   
bMaxPacketSize0                      0x8   
idVendor                                   0xXXXX   
idProduct                                  0xYYYY   
bcdDevice                                 0x1   
iManufacturer                             0x1   
iProduct                                    0x2   
iSerialNumber                            0x3   
bNumConfigurations                   0x1   
   
   
By setting the bDeviceClass, bDeviceSubClass, and bDeviceProtocol to zero this indicates the device will describe its functions via the interfaces.

Also Windows picks this up as a “Composite Device” which is good.

Normally with an IAD you would think to use the “Multi-Function” values described in the USB spec.  
   
2)       Configuration Descriptor  
   
Pretty much standard stuff you need to fill out for your own device, but the key here is the bNumInterfaces which in this case is 0x2

(1 interface for CDC, 1 interface for MSD). Normally it is 2 interfaces for CDC (1 for COMMUNICATION, second for DATA).  
   
Configuration Descriptor   
bLength                         0x9   
bDescriptorType             USB\_CONFIGURATION\_DESCRIPTOR\_TYPE   
wTotalLength                 0x4C   
bNumInterfaces             0x2   
iConfiguration                 0x0   
bmAttributes                  0xE0 ( Bus\_Powered Self\_Powered Remote\_Wakeup )   
MaxPower                     0x0   
   
3)       CDC Interface and Endpoint Descriptors  
   
This is they key. The CDC Interface Descriptor is a single interface with 3 endpoints.  
   
Interface Descriptor   
bLength                         0x9   
bInterfaceNumber           0x0   
bAlternateSetting           0x0   
bNumEndpoints             0x3   
bInterfaceClass             0x2 (Communications and CDC Control)   
bInterfaceSubClass        0x2 (Abstract Control Model)   
bInterfaceProtocol          0x0   
iInterface                       0x0   
   
Endpoint Descriptor   
bLength                         0x7   
bEndpointAddress          0x83 [IN]   
bmAttributes                  0x3 (USB\_ENDPOINT\_TYPE\_INTERRUPT)   
wMaxPacketSize           0x8   
bInterval                        0x10   
   
Endpoint Descriptor   
bLength                         0x7   
bEndpointAddress          0x1 [OUT]   
bmAttributes                  0x2 (USB\_ENDPOINT\_TYPE\_BULK)   
wMaxPacketSize           0x40   
bInterval                        0x0   
   
Endpoint Descriptor   
bLength                         0x7   
bEndpointAddress          0x82 [IN]   
bmAttributes                  0x2 (USB\_ENDPOINT\_TYPE\_BULK)   
wMaxPacketSize           0x40   
bInterval                        0x0   
   
4)       CDC Functional Descriptors  
   
The functional descriptors should be included as usual:  
   
Header  
Abstract Control Model  
Call Management  
   
Notably the Call Management Functional Descriptor has 2 nuances;   
a)       The bmCapabilities field should have bit D0 clear as this indicates that Device sends/receives call management information only   
          over the Communication Class interface (interface 0).  
b)       Per the spec bDataInterface is optional, but probably good to set it to 0.  
   
Also note that I did not include the Union Functional Descriptor as it seems unnecessary anyway with this configuration.  
   
5)       MSD Interface and Endpoint Descriptors  
   
Pretty much normal stuff here. Just that this interface is #1  
   
Interface Descriptor   
bLength                         0x9   
bInterfaceNumber           0x1   
bAlternateSetting           0x0   
bNumEndpoints             0x2   
bInterfaceClass             0x8 (Mass Storage)   
bInterfaceSubClass        0x6 (SCSI Transparent Command Set)   
bInterfaceProtocol          0x50 (Bulk-Only Transport)   
iInterface                       0x0   
   
Endpoint Descriptor   
bLength                         0x7   
bEndpointAddress          0x4 [OUT]   
bmAttributes                  0x2 (USB\_ENDPOINT\_TYPE\_BULK)   
wMaxPacketSize           0x40   
bInterval                        0x0   
   
Endpoint Descriptor   
bLength                         0x7   
bEndpointAddress          0x85 [IN]   
bmAttributes                  0x2 (USB\_ENDPOINT\_TYPE\_BULK)   
wMaxPacketSize           0x40   
bInterval                        0x0   
   
6)       INF File  
   
The INF file you create for the serial port must contain the following:  
   
USB\Vid\_xxxx&Pid\_yyyy&MI\_ww  
   
Where xxxx, yyyy are extracted from respective device descriptor fields. ww is the CDC interface number (0 in this case).  
   
   
That’s it!  
   
I hope this helps someone out there.  
   
Enjoy!

Creating a composite USB device that contains a CDC and another independent function can be tricky under Windows.

Using a single interface for both the CDC communication and data interfaces apparently works under Windows

but doesn’t comply with the USB CDC spec and thus might not work on other OS’s, including other Windows editions.   
  
From the CDC spec, Section 3.6.2, Abstract Control Model:   
The device uses both a Data Class interface and a Communication Class interface.  
  
Here are two other options:  
  
**Option 1. Composite: CDC + independent interface**  
  
In the device descriptor:  
bDeviceClass = 02h.   
  
CDC interfaces:  
communication interface  
data interface to carry application data  
A union functional descriptor specifies the CDC interfaces.  
  
Additional interfaces:  
1 or more independent interfaces (HID, mass storage, etc.)  
  
Windows:

Must provide an INF file for the CDC interface (not the entire device).

The hardware ID in the INF file must include the interface number.

In this example, the interface number is 00h (&MI\_00):  
  
USB\VID\_0925&PID\_1000&MI\_00  
  
Must also provide an INF file for the device to tell Windows to load the generic parent driver.

The INF file must contain a needs/includes section that references usb.inf.   
  
See:  
Enumeration of the Composite Parent Device  
<http://msdn2.microsoft.com/en-us/library/aa476434.aspx>  
  
Support for the Wireless Mobile Communication Device Class (WMCDC)

(The Registry Settings Section shows example INF file sections.)  
<http://msdn2.microsoft.com/en-us/library/aa476416.aspx>  
  
**Option 2. Composite: CDC + independent interface with IAD.**  
  
In the device descriptor:  
bDeviceClass = EFh, bDeviceSubClass = 02h, bDeviceProtocol = 01h.   
  
CDC Interfaces:  
communication interface  
data interface to carry application data  
  
An interface association descriptor (IAD) specifies the interfaces that belong to the CDC function.

The bDeviceClass code tells the OS that the device contains an IAD and enables loading the correct drivers.  
A union functional descriptor also specifies the CDC interfaces.  
  
Additional interfaces  
1 or more independent interfaces (HID, mass storage, etc.)  
  
Windows: Must provide an INF file for the CDC interface (see Option 1 above).  
  
Works only on Windows SP2 with hotfix, Vista  
  
More information:  
  
The Usbser.sys driver may not load when a USB device uses an IAD  
<http://support.microsoft.com/kb/918365>  
  
Interface Association Descriptor Engineering Change Notice (ECN)  
<http://www.usb.org/developers/docs/InterfaceAssociationDescriptor_ecn.pdf>  
  
USB Interface Association Descriptor Device Class Code and Use Model   
<http://www.usb.org/developers/whitepapers/iadclasscode_r10.pdf>  
  
I haven’t personally tested either of these options and would be interested to hear any reports.  
  
Jan Axelson  
[www.Lvr.com](http://www.lvr.com/)

Hello Jan Axselson  
  
Thanks for joining the MC forumboard  
  
I've collected these existing ones  
[Composite device won't initialize](http://forum.microchip.com/tm.aspx?m=251546)  
[Composite Device Problems](http://forum.microchip.com/tm.aspx?m=245361)  
[Composite device (HID + MSD) with PIC18F4550](http://forum.microchip.com/tm.aspx?m=245903)  
[Composite Device Problems](http://forum.microchip.com/tm.aspx?m=245361)