

USB 列舉教學，詳解

<https://wssllabcd.github.io/2012/11/28/usb-emulation/>

當 USB 插上 HUB 時，HUB 此時正在不斷的 Polling port 的狀態，一旦偵測到電位改變時，Hub 就會通知 Host 有新的裝置，當然會有一些 USB 全速與高速的判別，這部份請參考[全速和低速識別](#)，就不再提了，這邊只針對 CATC 所錄到的 USB 列舉作解析。

Host 對 Address 0 發送 GetDescriptor(Device Descriptor) 的請求

Packet	Dir	H	SETUP	ADDR	ENDP	CRC5	Pkt Len	Idle	Time S
1022	-->	S	0xB4	0	0	0x08	8	233.330 ns	00001.702
Packet	Dir	H	DATA0	Data				CRC16	Pkt Len
1023	-->	S	0xC3	80	06	00	01 00 00 40 00	0xBB29	16
Packet	Dir	H	ACK	Pkt Len	Time		Time Stamp		
1024	<--	S	0x4B	8	145.533 μs		00001.7026 3419		

addr0_desc

- BYTE 0, 1: 80, 06 代表 GetDescriptor
- BYTE 2, 3: 代表 Descriptor 索引，這裡是 1，代表 Device Descriptor
- BYTE 4, 5: 代表語言的 ID(應該是編碼，ex big5..etc ?)
- BYTE 6, 7: 代表 Length，代表 Device 要回傳多少 Data 回來，這裡是 0x40，而 Device 傳的 Data 只能小於等於這個長度

Device 回應 Host 所下的 GetDescriptor

而 Device 回的 Device Descriptor 是

Transaction	H	IN	ADDR	ENDP	T	Data													
1456	S	0x96	0	0	1	0: 12 01 00 02 00 00 00 40 76 0C 05 00 00 01 01 02 16: 00 01													

device_desc.png

- BYTE 0 :0x12，代表 Length，為 0x12 個 BYTE
- BYTE 1 :0x01，代表 Descriptor 的種類，01 代表 Device Descriptor
(註：第一個 BYTE 為描述資訊的長度，第二個 BYTE 為描述資訊的類別，長度太少，host 會不接受，過常 Host 會忽略)
- BYTE 2, 3 代表 USB 版本，value = 0200，也就是 USB 2.0
- BYTE 4 代表 Device class(裝置類別)，這裡是 0x00，代表這個 Device 的 Class 是在介面描述元中定義，(如果是 0x09，代表 Hub，參考下面的 Class code，USB-IF)
- BYTE 5 代表 Sub Device 類別，這裡是 0

- BYTE 6 代表 Protocol，這裡是 0x00 (如果是 0x01，代表 Hi-speed hub with single TT(參考各 Device 的 Class code))
- BYTE 7 代表 bMaxPacketSize，這裡是 0x40，也就是端點 0 最大的封包大小 (非端點 1，端點 2，其他端點的封包大小由 endPoint Descriptor 描述)
- BYTE 8,9 代表 vid，這裡是 0x0C76 代表 JMtek，LLC.，(如果這裡是 05E3，代表 Genesys (創唯))(VID 列表 <http://www.linux-usb.org/usb.ids>)
- BYTE 10,11 代表 PID，這裡是 0x0005(0005 Transcend Flash disk) (如果是 0608，代表 USB-2.0 4-Port HUB)
- BYTE 12,13 代表 Deive 序號，0x0001
- BYTE 14: 代表 Manufacturer 號，這裡為 1
- BYTE 15: 代表 Product 號，這裡為 2
- BYTE 16: 代表 SN 號，這裡為 0
- BYTE 17 numConfiguartions: Number of possible configuration，這裡是 1

注意 iManufacturer，iProduct，iSerialNumber，這個決定了後面 GetString 請求裡 wValue 掉低字節的值和所獲得 string 的關係。

註：如果是 Bulk-only 的裝置的話，DeviceClass，DeviceSubClass，DeviceProtocal 應全為 0

Host 對 Address 0 發送 SetAddress 的請求

因為每個 USB Device 初始的 Address 都是 0，所以一旦 Host 找到了 Device，就會把他從 Address 0 移開，以免跟後來新的 Device 衝突，此時，Host 把這個 device 設在 Address 3 的位置

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len	Idle	Time Stamp
8713	-->	S	0xC3	00 05 03 00 00 00 00 00	0x57E3	16	300.000 ns	00004.5159 4433

host_set_addr.png

- BYTE 0: 因為不用 Data-in，所以不是 0x00
- BYTE 1: 代表 Command 種類，這裡是 0x05，代表 SetAddress (GetDescriptor 這裡是 0x06)
- BYTE 2,3 : 代表 Address，範圍為 0~127
- BYTE 4~7: fixed 0

Host 對 Address 3 發送 GetDescriptor 的請求

Host 對新的 Address 下 GetDescriptor，目的可能是想要觀察 Device 有沒有正確的被配到新的位置，理論上來說，回傳的 Data 應該要一樣才對，而接下來所有發送的位置，都會以新的位置(也就是 Address 3)來發送

Host 發送 GetDescriptor(Config 類) 的請求

再看 CATC 圖之前先說明一下，此時 Host 要取回 Device 的 Configuration
而 Configuration Descriptor 中有包含好幾種 Descriptor，概述如下

- 組態描述元(Configuration Descriptor)
- 介面描述元(Interface Descriptor)
- 端點描述元(Endpoint Descriptor)
- HID 描述元(HID Descriptor)
- etc..

這邊有個問題，每支 Device 的 interface 與 endpoint 的數量都不一樣，也就是說，每支 Device 的 GetDescriptor(Config 類)的全部長度是不一樣的，那 host 要叫 Device 回傳多少長度呢？

這邊 Host 把 GetDescriptor(Config 類)這個動作分成兩步驟

1. 只先取 Configuration Descriptor(也就是前 9 BYTE)
2. 根據 Configuration Descriptor 中的 BYTE 2,3，來得知 GetDescriptor(Config 類)的總長

請看下圖，Host 發送 GetDescriptor(Config 類)，雖然整個 config 我不知道多長，但最起碼的 Configuration Descriptor 是固定 9 Byte 的，所以 host 先叫 device 回傳長度為 9，代表 Host 只先取回 Configuration Descriptor

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
9179	-->	S	0xC3	80 06 00 02 00 00 09 00	0x7520	18

host_GetDesc.png

Device 回傳 GetDescriptor(Config 類) 的請求

Device 回傳 9 BYTE，而前面 9 BYTE 固定是 Configuration Descriptor

Transaction	H	IN	ADDR	ENDP	T	Data
1773	S	0x96	3	0	1	0: 09 02 20 00 01 01 00 80 32 09 04 00 00 02 08 06 16: 50 00 07 05 81 02 00 02 FF 07 05 02 02 00 02 FF

dev_configuartion.png

- BYTE 0: 0x09，代表這個 Descriptor 的長度
- BYTE 1: 0x02，代表這 9 BYTE 是屬於 Configuration Descriptor
- BYTE 2 :0x20，wTotalLength_L，0x20(該配置返回的數據總長度，包括其下 interface，Endpoint descriptor 的總長度)
- BYTE 3 :0x00，wTotalLength_H

- BYTE 4: 0x01, 代表有幾個 interface, 這裡為 1 (猜測, 一個 USB_interface 對應一種 USB 邏輯設備, 比如鼠標、鍵盤、音頻流。所以, 在 USB 範疇中, device 一般就是指一個 interface。一個驅動只控制一個 interface。)
- BYTE 5: 0x01, 代表這個 Configuration 的編號 (Host 可由這個編號, 下 SetConfiguration 來做 配置的切換)
- BYTE 6: 0x00, 代表這個 configuration 的 String Descriptor 編號, 如果為 0x00, 則代表沒有 String Descriptor
- BYTE 7: 0x80, 代表這個配置的一些 attribute, Bit7 fixed 1(for historical reason), Bit6=1, 代表從 Usb Bus 供電, Bit5=1 代表 Remote wake up
- BYTE 8: 0x32, 所用電流, 單位為 2mA, 這邊代表這個裝置需要 100mA (USB Hub 最大為 500mA)

Host 再一次發送 GetDescriptor(Config 類) 的請求

雖然 Device 告訴 host 總長為 0x20, 但這裡 Host 很闊氣的下 0xFF

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
9234	-->	S	0xC3	80 06 00 02 00 00 FF 00	0x9725	16

Host_getConfiguration.png

Device 再一次回傳 GetDescriptor(Config 類)

Transaction	H	IN	ADDR	ENDP	T	Data
1773	S	0x96	3	0	1	0: 09 02 20 00 01 01 00 80 32 09 04 00 00 02 08 06 16: 50 00 07 05 81 02 00 02 FF 07 05 02 02 00 02 FF

device_configuration.png

這裡面一共有四組 (可藉由計算長度 (第一個 BYTE) 來判斷)

- 第一組為 Configuration, 共 9 BYTE, 而 0x02 代表 組態描述元(Configuration Descriptor)
- 第二組為 Interface, 共 9 BYTE, 而 0x04 代表 介面描述元(Interface Descriptor)
- 第三組為 Endpoint, 共 7 BYTE, 而 0x05 代表 端點描述元(Endpoint Descriptor)
- 第四組為 Endpoint, 共 7 BYTE, 而 0x05 代表 端點描述元(Endpoint Descriptor)

根據 USB 規定, 當指令要取得組態描述元時, 則裝置必須要將裝置描述元, 介面描述元, HID 描述元, 端點描述元的資料全都回傳給主機, 並且按照規定的順序排列

而 Configuration 前面已經有講過了

接下來會剖析 介面描述元(Interface Descriptor) 與 端點描述元(Endpoint Descriptor)

Device Configuration 中的介面描述元(Interface Descriptor)

以下這張圖同 device_configuration，貼在這邊只是方便對照，interface Descriptor 請從 offset 0x9 開始看起

Transaction	H	IN	ADDR	ENDP	T	Data
1773	S	0x96	3	0	1	0: 09 02 20 00 01 01 00 80 32 09 04 00 00 02 08 06 16: 50 00 07 05 81 02 00 02 FF 07 05 02 02 00 02 FF

device_configuration.png

- BYTE 0: 0x09，代表長度
- BYTE 1: 0x04，代表 Interface descriptor
- BYTE 2: 0x00，Interface Number
- BYTE 3: 0x00，AlternateSetting，代表交替設定 (linux 會有機會交替的使用不同的 AlternateSetting)
- BYTE 4: 0x02，代表此介面所使用的端點數 (對應到 linux unsigned num_altstting)
- BYTE 5: 0x08，Class Code (群組代碼)，代表 Interface Class，0x08 代表 mess storage(see http://www.usb.org/developers/defined_class)
- BYTE 6: 0x06，SubClass code(次群組代碼)：主群組代碼如果有細分的話，就在這裡描述，而 0x06 代表 SCSI (0x02 代表 CD-ROM)(對應到 linux kernel drivers/usb/storage/protocol.h)
- BYTE 7: 0x50，interfaceProtocol(介面協定)，0x50 代表 Bulk-Only Transfer(對應到 linux kernel ，drivers/usb/storage/transport.h)
- BYTE 8: 0x00，Interface 的 String Descriptor，0x00 代表沒有

Mass Storage specifications Overview 節錄 (對應到 BYTE 6)

Table 2.1 – SubClass Codes Mapped to Command Block Specifications

SubClass Code	Command Block Specification	Comment
01h	Reduced Block Commands (RBC) T10 Project 1240-D	Typically, a Flash device uses RBC command blocks. However, any Mass Storage device can use RBC command blocks.
02h	SFF-8020i, MMC-2 (ATAPI)	Typically, a C/DVD device uses SFF-8020i or MMC-2 command blocks for its Mass Storage interface.
03h	QIC-157	Typically, a tape device uses QIC-157 command blocks.
04h	UFI	Typically a floppy disk drive (FDD) device
05h	SFF-8070i	Typically, a floppy disk drive (FDD) device uses SFF-8070i command blocks. However, an FDD device can be in another subclass (for example, RBC) and other types of storage devices can belong to the SFF-8070i subclass.
06h	SCSI transparent command set	
07h – FFh	Reserved for future use.	

Usb_Spec_SubClassCode.png

Mass Storage specifications Overview 節錄 (對應到 BYTE 7)

Table 3.1 – Mass Storage Transport Protocol

<i>bInterfaceProtocol</i>	Protocol Implementation	Comment
00h	Control/Bulk/Interrupt protocol (with command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
01h	Control/Bulk/Interrupt protocol (with no command completion interrupt)	USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
50h	Bulk-Only Transport	USB Mass Storage Class Bulk-Only Transport
02h – 4Fh	Reserved	
51h – FFh	Reserved	

Usb_Spec_MS_trans_proto.png

Device Configuration 中的端點描述元(Endpoint Descriptor)

共 7 BYTE，描述端點的屬性及位置（不描述端點 0），端點的數目是由 Interface descriptor 的 BYTE 4 得知每個端點在 Device 中，都是代表某個記憶體位置，而各個端點有不同的傳輸型態及最大傳輸值

PS: 以下這張圖同 device_configuration，貼在這邊只是方便對照，Endpoint Descriptor 請從 offset 0x12 開始看起

Transaction	H	IN	ADDR	ENDP	T	Data
1773	S	0x96	3	0	1	0: 09 02 20 00 01 01 00 80 32 09 04 00 00 02 08 06 16: 50 00 07 05 81 02 00 02 FF 07 05 02 02 00 02 FF

device_configuration.png

- BYTE 0: 0x07，代表長度
- BYTE 1: 0x05，Descriptor Type，0x05 代表端點描述元
- BYTE 2: 0x81，End Point Address，Bit[3:0] 代表端點號，， Bit[7] 1:In，0: out，所以 0x81 代表 Endpoint 1，負責 data-In，而控制端點是雙向的，所以不用看 Bit7
- BYTE 3: 0x02，Attribute(端點屬性): Bit[1:0] 代表支援的傳輸模式，10b(即 2) 代表 Bulk 類
- BYTE 4: 0x00，wMaxPacketSize_L: 最大傳輸量，如果是 HighSpeed，則是 512，也就是 0x200
- BYTE 5: 0x02，wMaxPacketSize_H (在 linux 中，kernel 會根據 wMaxPacketSize 的大小，使用 kalloc 配置一塊記憶體給該 endptr 使用而有趣的是，由於是 LSB 的關係，linux 會使用一個 le16_to_cpu 函數把 0x0002 轉成 0x0200)
- BYTE 6: 0xFF，Interval，代表多少間隔內，Device 最多只發一次 NAK 封包（或是在中斷傳輸中，host 間隔多少時間來 polling device)

Host 下 Get String Descriptor

Host Get string descriptor(wValue MSB: Descriptor Type =0x3)

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len
104521	-->	S	0xC3	80 06 00 03 00 00 FF 00	0x2B26	18

Host_GetStringDesc.png

- BYTE 2: index，這裡的 index=0，代表只取 LanguageID
- BYTE 3: 代表 GetString Descriptor
- BYTE 4, 5 : 代表 Language ID，這裡為 0，代表沒有特別的 ID
- BYTE 6, 7: 代表 Length

Device 回傳 String Descriptor

Transaction	H	IN	ADDR	ENDP	T	Data	ACK
44837	S	0x96	3	0	1	04 03 09 04	0x4B

Device_strDesc.png

- BYTE 0: Length，長度為 4
- BYTE 1: DescriptorType，為 03
- BYTE 2, 3: bString，0409，代表 Language ID = 0x0409，代表使用的 Language 為 English (United States)

PS: 有關 Lang ID，可見 USB Language Identifiers(LANGIDs).PDF

Host 再下一次 GetString Descriptor (Get Product String)

接下來，Host 會在下次 GetString Descriptor 下，與上次不同的是，wValue 的 LSB(BYTE 2)，也就是 Descriptor Value 變成了 2

Packet	Dir	H	DATA0	Data	CRC16	Pkt Len	Idle	Time Stamp
104569	-->	S	0xC3	80 06 02 03 09 04 FF 00	0xE9DB	18	300.000 ns	00005.6498 3221

Host_GetStrDesc_2.png

這裡 wValue 低八位 =2，對應 Product String.

而 wIndex，也就是 Language ID 變成了 0x0409(Byte 4, 5)

Device 回傳 GetString Descriptor (Product String)

Packet	Dir	H	DATA1	Data	CRC16	Plt Len	Idle
104597	<--	S	0xD2	0: 22 03 55 00 53 00 42 00 20 00 4D 00 61 00 73 00 16: 73 00 20 00 53 00 74 00 6F 00 72 00 61 00 67 00 32: 65 00	0x323C	42	366.660 ns

Device_ProductStr.png

把這些數據按照 ASCII 編碼翻譯過來，就是現在 Host 端的 device name 了

Offset:	
00000	" ?U ?S ?B ?
00008	?M ?a ?s ?
00010	s ? ?S ?t ?
00018	o ?r ?a ?g ?
00020	e ?

ProductStr_ascii.png

到此基本上 USB Emulation 就已經結束了，接下來就是 UFI 的 inquiry 了