USB Interface Association Descriptor Device Class Code and Use Model

Revision 1.0

July 23, 2003

Contributors

John S. Howard	Intel Corporation
Brad W. Hosler	Intel Corporation
Abdul R. Ismail	Intel Corporation
Geert Knapen	Philips

Table of Contents

1. Overview

From day 1 (of the USB specification) there has been ambiguity with respect to whether multi-function devices should be allowed to use more than one interface per logical function. The core specification did not provide any specific framework support for multiple interfaces per function, but several Device Working Groups (DWG) defined device classes using them, with different methods for identifying how the interfaces should be grouped together. There was a trailing effort in the DWG Common Class group to define a standard method, but it was late in definition and was never adopted and eventually was decommissioned.

The recent USB 2.0 ECN *Interface Association Descriptor* (IAD) solves the problem by defining a standard method in the USB device framework for describing associations of interfaces (and their alternate settings) that should be bound to the same instance of a device driver.

There exists a legacy issue for new devices that use the IAD (implying interface level binding to device drivers as opposed to device-level binding) when connected to systems where USB system software does not understand the IAD. Although the IAD will be ignored, the device may not work as expected because the USB system software will not properly bind the interfaces with drivers.

The USB core team has allocated a device-level class code that must be included with device implementations that use the IAD. This provides for the easiest detection of IAD-enabled devices during device enumeration that will allow installation of a special-purpose function driver that has the capability of correctly parsing the configuration and locating the appropriate drivers for IAD-enabled devices.

Devices that use the IAD must use the device class, subclass and protocol codes as defined in the example device descriptor illustrated in Table 1-1. This set of class codes is defined as the *Multi-Interface Function Device Class Codes*.

Table 1-1. Example Device Descriptor Using Class Codes for IAD

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	See Table 9-8, USB Specification, Revision 2.0
2	bcdUSB	2	BCD	See Table 9-8, USB Specification, Revision 2.0
4	*bDeviceClass*	1	**EFH**	**Miscellaneous Device Class**
5	*bDeviceSubClass*	1	**02H**	**Common Class**
6	*bDeviceProtocol*	1	**01H**	**Interface Association Descriptor**
7	bMaxPacketSize0	1	Number	See Table 9-8, USB Specification, Revision 2.0
8	idVendor	2	ID	See Table 9-8, USB Specification, Revision 2.0
10	idProduct	2	ID	See Table 9-8, USB Specification, Revision 2.0
12	bcdDevice	2	BCD	See Table 9-8, USB Specification, Revision 2.0
14	iManufacturer	1	Index	See Table 9-8, USB Specification, Revision 2.0
15	iProduct	1	Index	See Table 9-8, USB Specification, Revision 2.0
16	iSerialNumber	1	Index	See Table 9-8, USB Specification, Revision 2.0
17	bNumConfigurations	1	Index	See Table 9-8, USB Specification, Revision 2.0

Section 2 illustrates how the IAD and *Multi-Interface Function Class Codes* should be used in the device framework of IAD-enabled devices.

2. IAD Use Model Example

This section provides an illustrative example of how the IAD should be used in a typical device implementation. The core USB specification does not specify many organizational (positional relationship) requirements on how the set of descriptors that are returned from a GetDescriptor(Configuration) request should be constructed. Figure 2-1 illustrates the recommended layout for organizing these descriptors so that host software can easily parse them and definitively know which descriptors to provide to each function driver. The general methodology for organizing the descriptor sets is to group them 'by device function'. This basically means that all descriptors for a particular device function should always be located 'together' (see Figure 2-1 for an example).

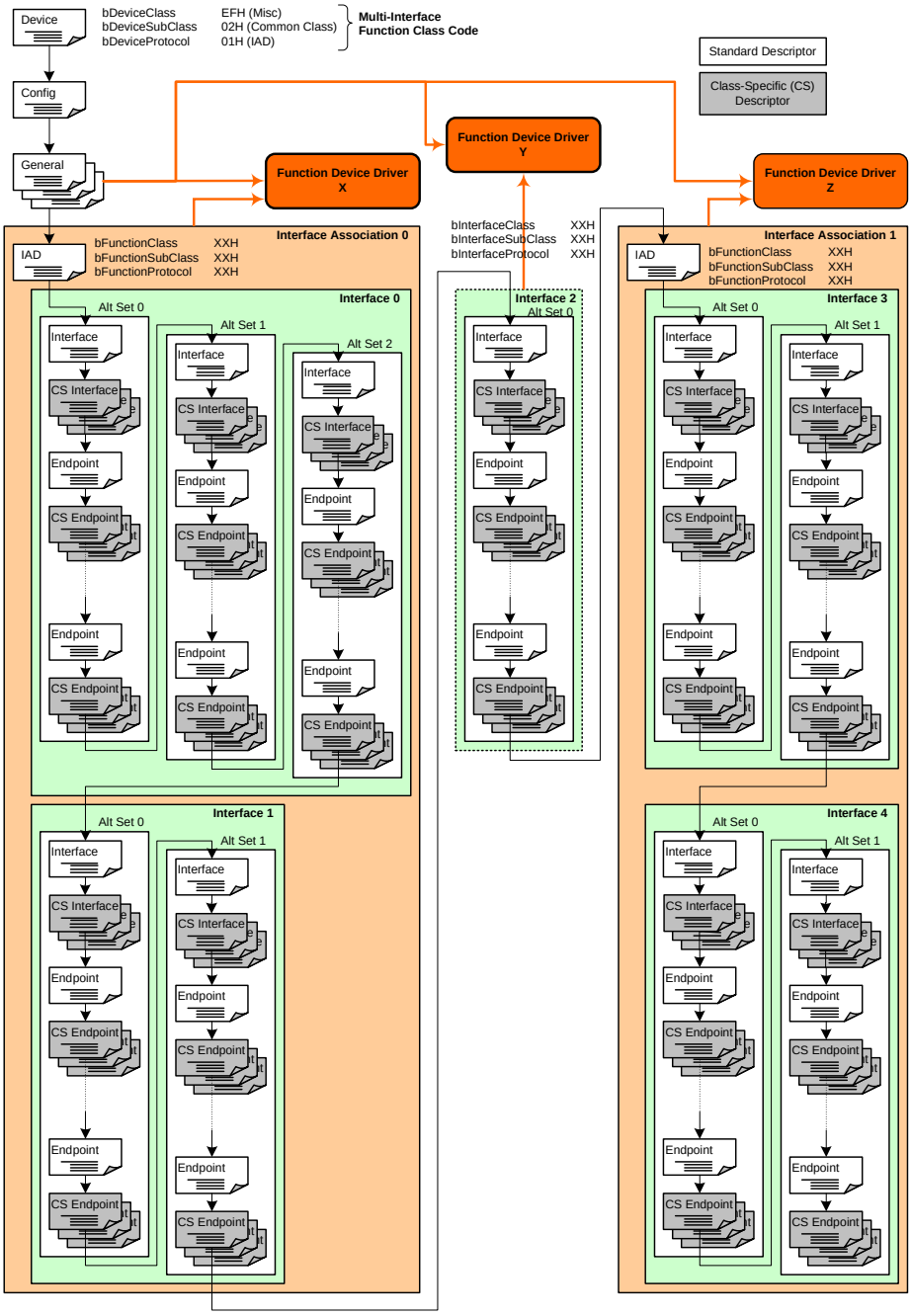

Figure 2-1. Example Device Framework Using Interface Association Descriptors

Additionally, Figure 2-1illustrates how the sets of descriptors should be bound to device drivers. Note there may be specific (additional) requirements for how class-specific descriptors should be grouped with IAD associations. Those would be specified in specific device class documents.

At the top of the framework example, the Device Descriptor includes device class, subclass and protocol codes for a *Multi-Interface Function Device Class* device.

Next is the configuration descriptor set. At a high level, this particular configuration includes three device functions. Any descriptors between the configuration descriptor and the first interface or IAD descriptor should be considered as "global" and be delivered to every function device driver (see example).

The first functional association includes two interfaces (including all class-specific descriptors and alternate settings). The second device function is a single-interface function and does not require an IAD. The third device function is similar to the first. For each device function, system software must provide the device driver all of the descriptors in the 'association' as well as the 'global' descriptors described above.

The value in the *bInterfaceCount* field must include all of the interfaces in the intended set. For example, assume a multi-interface function with interfaces numbered N through M. The *bFirstInterface* field gets the value of N and *bInterfaceCount* gets the value of (M-N)+1.[1] Note that each interface can have zero or more alternate settings, but alternate settings don't figure into the calculation for *bInterfaceCount*.

For device functions that use an IAD (like the 1st and 3rd functions in the example), USB system software should construct 'hardware identifiers' used to locate and load a device driver using the idVendor and idProduct from the Device Descriptor and the bFirstInterface field from the IAD. Further, system software should construct 'compatibility identifiers' using the class code fields (bFunctionClass, bFunctionSubClass, bFunctionProtocol) from the IAD.

[1] Recall that all of the interfaces in the association must be number contiguously, so (M-N)+1 arithmetic always works.