USB Feature Specification: Dynamic Logical-Device

INTEL® CORPORATION

Revision 1.0

October 27, 1999

Revision History

Revision	Issue Date	Comments
1.0	10/27/1999	Version 1.0 release.

Contributors

John Howard Intel Corporation
Steve McGowan Intel Corporation

Please send comments via electronic mail to ccscomments@usb.org.

Table of Contents

1. Introduction

The following feature specification describes a method of handling logical-devices" (as described in the "USB Feature Specification: Logical-Devices") which are not statically enumerated, but whose characteristics may change dynamically.

The class of a logical-device really describes the class of the data communicated by collection of interfaces that it contains. The exact nature of that data in some cases is not known until an external event occurs, e.g. a phone call. The characteristics of logical-devices dependent on external events may change drastically, depending on the requirements and capabilities associated with the particular external event. The changes in these characteristics can include any of the logical-device, standard interface and endpoint information which would normally be associated with that logical-device in a particular configuration, such as the class code, number and type of alternate settings and bandwidth capabilities. It can also include class-specific information such as the formatting of the actual stream.

Such logical-devices are known as "dynamic logical-devices". A change in a dynamic logical-device is indicated to system software through a logical-device notification.

Throughout this document the terms function and logical-device will be used interchangeably, however they both refer to a Logical-Device and the collection of descriptors that it defines.

1.1 Purpose

The purpose of this document is to provide a common specification of how the creation of, or changes in a logical-device can be identified at run-time.

1.2 Scope

This document fully describes the Common Class Dynamic Logical-Device Feature extension for USB Devices. It describes:

- Host and Device requirements for dynamic logical-device notification.
- The format of dynamic logical-device notification requests.

1.3 Related Documents

USB Specification, Version 1.1, available at http://www.usb.org.

USB Common Class Specification, Version 1.0, available at http://www.usb.org.

CCS Feature Specification: Version Descriptor, available at http://www.usb.org.

CCS Feature Specification: Logical-Devices, available at http://www.usb.org.

CCS Feature Specification: Notifications, available at http://www.usb.org.

1.4 Terms and Abbreviations

CCS	Common Class Specification
DLD	Dynamic Logical-Device
LD	Logical-Device

PNP	Plug and Play

2. Management Overview

Dynamic Logical-Device (DLD) Notifications allow for late binding of drivers where a Logical-Device's (LD) characteristics are not known until after enumeration or they are modified by an application. DLDs can also be used to indicate the attachment or removal of devices to the USB that bridge to alternate bus technologies.

For example, infrared control interfaces (e.g. IrDA-C) are designed to allow devices such as infrared keyboards, mice, joysticks and other input/output devices to communicate within a typical room in a home. IrDA-C provides a standard that allows IrDA-C devices to communicate with an IrDA-C host, even though different companies may have manufactured each device. Similar, RF technologies are also being developed.

Conceptually, an IrDA-C host device is similar to a USB hub. When an IrDA-C host detects a new device it enumerates the same way that USB device would when plugged into a hub. DLD notifications provide the equivalent of a "soft" hub capability for a IrDA-C devices attached to USB through a IrDA-C to USB host, allowing the devices to be "attached" and "removed" from the system.

3. Functional Characteristics

3.1 Requirements

DLD notification could theoretically be handled by fully enumerating all of the possibilities a particular function would ever display. This, however, is potentially an enormous list, expensive to implement in a device and difficult to accurately develop. Also, the particular characteristics may actually be a function of a remote connection or a removable sub-device, and not intimately tied to the device at time of manufacture.

Therefore, a method of handling such function devices must be developed which does not place such a burden on the device or on the imagination of the device designer. Also, it is extremely desirable that this method perturbs the version 1.0 methods of obtaining interface information as little as possible.

DLD notification support is required to exist on both the Host and Device.

A device that supports dynamic notification must also provide a Common Class Notification pipe to inform the host of changes in state of DLDs.

The intent this specification is to hide the dynamic nature of LDs from client device drivers that are unaware of this feature.

3.2 Specification

A device that requires DLD notification support will declare a CCS class interface with an Interrupt In endpoint to provide the notification pipe. The host will bind the USB system software to the CCS interface at device enumeration time. The CCS driver will recognize DLD notification requests and further enumerate other LDs on the USB device as the notifications declare them. When the characteristics of a DLD become known, the DLD will take on a particular class, with particular characteristics, and at that time be bound to a driver.

A device may support up to 255 DLDs.

The specification for DLD notifications includes the following:

- Definition of CCS protocols that identify the required support.
- Discussion of how the GetDescriptor request returns the LD descriptors when an announcement notification is received for a DLD.
- A new CCS notification message to inform the host of DLD state changes.
- New CCS features to allow the host to query and set the state of a DLD.

Each of these areas is discussed in more detail below.

3.3 Version Identification

Version of the CCS DLD feature implemented by the device can be obtained through the CCS Version Descriptor. See the “USB Feature Specification: Version Descriptor” for more information.

4. Operational Model

4.1 Managing DLDs

This section provides an overview of DLDs, layout rules for descriptors, the definition of the format for Announce Dynamic Logical-Device messages and DLD notification related features.

Static enumeration occurs when a device is first detected. At this time the CCS driver is loaded. The CCS driver is also an enumerator, which can load device drivers for other static logical-devices and interfaces on a per- LD or interface basis. The CCS driver knows how to enumerate a DLD, including the retrieval of the LD descriptor. In addition it locates, optionally loads and hands off the LD to the appropriate driver. Figure 4-1: illustrates the relationship between the enumeration driver, the function driver and the corresponding relationship between the drivers on the host and alternate interface settings on the device.

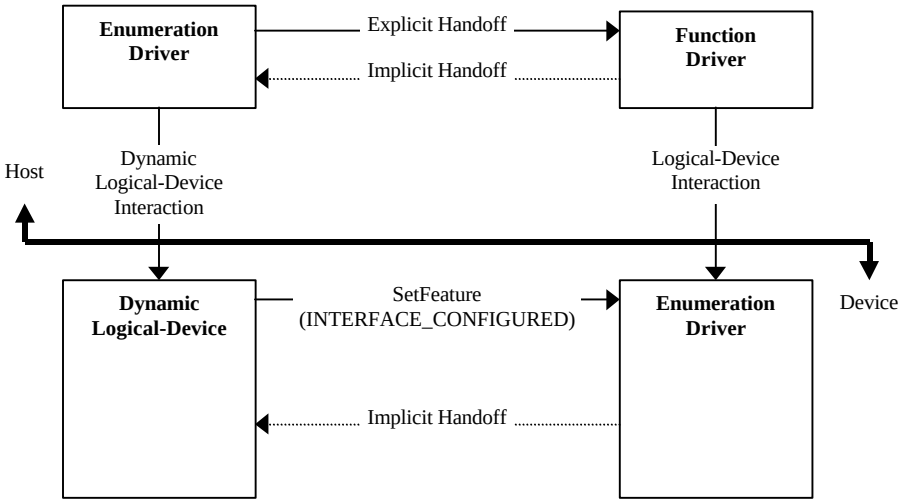


Figure 4-1: Driver Relationships

In response to a GetDescriptor(CONFIGURATION) request, a device that is unconfigured and supports DLDs will return the descriptors for the CCS notification interface and any other permanently defined LDs and interfaces. Once configured, a device will return the descriptors for the CCS notification interface, all permanent and all currently defined LDs and interfaces. Descriptors associated with DLDs can be accessed with the appropriate GetDescriptor() request after an Announce DLD Notification has been received.

If the capabilities of a DLD change, the device will send an Announce DLD Notification to the host which identifies the number of the logical-device that changed. The host will respond with a series GetDescriptor() requests to the identified logical-device which will return a new set of logical-device descriptors.

Announce DLD Notifications inform the host that the state of a DLD has changed. If the announced DLD is already configured (LD_CONFIGURED is set), the host must compare newly retrieved LD descriptors with it's current image of the LD descriptors to determine the aspects of the LD that have changed. In the interval between receiving the Announce DLD Notification and retrieving the LD descriptor the state could change again. The LD descriptors returned by a device always represents the instantaneous state of a LD.

This specification does not define a minimum or maximum number of DLDs that a device may declare. A device must expect the possibility that a CCS driver may not enumerate a DLD, announced by a notification. This can

occur if the CCS driver runs out of resources. It is suggested that a CCS driver implementation be capable of supporting at least 16 dynamic LDs per device.

The LD Descriptor will identify all static LDs and the maximum number of DLDs that the device can support. After a device is configured, a device may send an Announce DLD Notification at any time. When the enumerator driver detects that a DLD needs to be enumerated, it retrieves new LD descriptors via GetDescriptor() requests to the target LD, then derives from the new LD's device descriptor an appropriate function driver and hands off ownership of the LD. The enumerator will assign device handles to LDs and DLDs that are identical to the device handle that a physical device would receive, making all three types of devices identical as far as a function driver was concerned.

Due to timing constraints, it may be necessary for the device to respond to external stimuli before the Host has opportunity to respond to the need for re-enumeration. It is acceptable for the device to respond to the external stimuli, however it is not allowed to respond to requests on pipes defined within the LD provided to the Host until the Host has issued a SetConfiguration(LD ID) request. It is up to the operating system to ensure that the abstraction seen by the function driver is consistent regardless of device's state: not addressed, not attached or configured. The execution of a SetConfiguration(LD ID) request to the LD by the host indicates to the device that the new LD is CONFIGURED. Note: a dynamic logical-device can also define interfaces with alternate interfaces.

When a function driver is finished with the LD it can inform the enumeration driver, by issuing a SetConfiguration(0) request to the CCS LD enumerator assigned handle. When this occurs, the enumeration driver, which traps all SetConfiguration requests, will convert the device handle into the appropriate LD ID and issue a SetConfiguration(LD ID,0) request to the target LD.

If the capabilities of a LD are no longer valid, the LD will send an Announce DLD Notification to the enumeration driver. The enumeration driver must make it appear to the function driver as though the device associated with the DLD was disconnected.

In this case, the enumeration driver will respond with a GetDescriptor(DEVICE, LD ID) request to which the device will respond with a zero length device descriptor because the DLD no longer exists.

The host may issue a GetDescriptor() or other request to a LD at any time to query the current capabilities of a LD. A GetDescriptor() request to an undefined DLD will return zero length data, all other requests will stall.

The device may issue the Announce DLD Notification message whether a DLD is in a Not Attached, Addressed or Configured state. The recipient of a Announce DLD Notification message is always the enumeration driver.

A DLD in an Addressed or Configured state may also notify the Host of the need for re-enumeration. For example, the DLD's pipe characteristics may be changing or it may be terminating. The enumeration driver would then make the new DLD appear to be attached to the USB. If a DLD is currently set to a Addressed or Configured state the only allowable enumeration change is to a Not Attached state. The system software services must provide the correct mapping of requests to enforce a single view to the function driver.

Announcement DLD Notifications are transmitted to the host over an interrupt in endpoint declared in a CCS class interface, using the format defined in the "CCS Feature Specification: Notifications". The CCS driver pareses the all interrupt messages received from the device. Note that the notification endpoint may be implemented as a shared endpoint. In some devices, the notification interface may be the only interface statically declared and all the dynamic function devices may share a single interrupt endpoint with the with the CCS interface. This means that immediately after enumeration, the notification endpoint will be "shared" but there will be no other logical endpoints to share it with, until after the first Announcement DLD Notification is processed.

The "USB Feature Specification: Logical-Device" states that the LD IDs start at the value 1. The value 0 is reserved and must never be used. The LD feature specification does not require that LDs use sequential LD IDs, only that they are unique on a device and are within the range defined in the LD Descriptor. Due to the nature of

DLDs, the LD IDs may appear out of sequence (or some may disappear why others remain, creating holes). The maximum LD ID is defined in the LD Descriptor.

The “USB Core Specification” states that interface numbers must be zero-based. The LD feature specification does not require that interface use sequential interface numbers, only that they are unique on a device. Due to the nature of DLDs, an interface number may appear out of sequence (or some interfaces may disappear why others remain, creating holes in the interface numbering). The valid set of interface numbers can be derived by reading the Configuration Descriptors from the physical device and all LDs. Any unattached LDs will stall on the GetConfiguration request.

The “USB Core Specification” states that the physical endpoint numbers must be zero-based. The LD feature specification does not require that interface use sequential endpoint numbers, only that they are unique on a device. Due to the nature of DLDs, an endpoint number may appear out of sequence (or some endpoints may disappear why others remain, creating holes in the endpoint numbering). The valid set of endpoint numbers can be derived by reading the Configuration Descriptors from the physical device and all LDs. Any unattached LDs will stall on the GetConfiguration request.

5. Descriptors

This section defines the feature-specific descriptors for the DLD feature.

5.1 Interface Descriptor

To invoke the Common Class driver and provide DLD support the device must declare a CCS Interface Descriptor. See the CCS Logical-Device Feature Specification for a detailed description of the CCS Interface Descriptor.

5.2 CCS Version Descriptor

A device that employs DLD functionality must declare a CCS Version Descriptor. The CCS Version descriptor will contain a Version Information Records for the Logical Device and CCS DLD features.

The Feature ID (*bFeatureID*) for the DLD feature is 0x04.

All Feature Flags (*bmFeatureFlags*) are reserved and set to 0.

See the CCS Version Descriptor Feature Specification for a detailed description of the CCS Version Descriptor.

5.3 CCS Logical-Device Descriptor

The *bfirstDynamicLD* member of the LD descriptor identifies the total number of DLDs that are supported by the device.

5.4 Interface and Endpoint Descriptor

A device that employs DLD functionality must also declare a Notification pipe using the interface and endpoint descriptors as described in the “USB Feature Specification: Notifications”.

5.5 Sub-Device Registration

A host system uses device-specific information in a device or interface descriptor to associate a device with a driver. A device that supports removable sub-devices where the sub-devices can be manufactured by a vendor other than the device vendor may use dynamic logical-devices. It can be assumed that the Vendor ID, Product ID, device release number, and manufacturer, product and Serial number strings defined in the LD's device descriptor are associated with the sub-device. And that the host system may use the sub-device-specific information in a LD descriptor to associate a driver with the particular LD.

The rules for locating USB sub-device drivers are identical to those defined in the USB Common Class Specification except that the LD supplies the device or interface descriptor.

6. Requests and Notifications

6.1 DLD Requests

DLD descriptor retrieval follows the same rules as LD descriptor retrieval. The LD ID is provided by the DLD announcement. See Section 6 of the "CCS Feature Specification: Logical Device".

6.2 Announce DLD Notification

bmNotificationType	bNotification	wValue	wIndex	wLength	Data
10100001B	ANNOUNCE DLD	Zero	LD ID	zero	N/A

The *bmNotificationType* field contains the binary value 10100001B to indicate that the request is a class specific notification, originating from the CCS interface. The *wIndex* field contains the LD ID of the target LD in the low order 8 bits. The high order 8 bits of *wIndex* are reserved and set to 0. When an external event has occurred in which the characteristics of one or more DLDs has become defined or undefined, the device sends a CCS Announce Notification to the host for each LD that changes state. CCS DLD Notifications take place on a permanent interrupt pipe provided by the device and use a format similar to that used by USB device requests over the default pipe.

The host acknowledges the receipt of each Announce DLD Notification to the device by issuing a GetDescriptor(Device, LD ID) request targeted at the LD identified in the CCS Announce DLD Notification.

The format of the GetDescriptor request follows:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type = Device and Descriptor Index = 0	LD ID	Descriptor Length	Descriptor

More than one CCS DLD Notification may be concatenated into a single interrupt response. The CCS driver will look for CCS DLD Notifications until the interrupt buffer is depleted. The interrupt buffer will not contain partial CCS DLD Notifications.

The CCS driver responds to an Announce DLD Notification by requesting a GetDescriptor(DEVICE, LD ID). If a zero length LD device Descriptor is returned, then the CCS driver can assume that the requested LD is no longer defined. Under these circumstances the CCS driver must cleanly terminate the enumeration process. Otherwise, the CCS driver will enumerate the new LD. After the information is obtained the CCS Driver will issue SetConfiguration requests as appropriate and bind specific device drivers to the function device(s) as necessary.

Note that announcements are asynchronous events. It is possible that in the interval between receiving the announcement and reading the LD's device descriptor or, reading the LD's other standard descriptors and sending a SetConfiguration request to a LD, the state of an LD may change again.

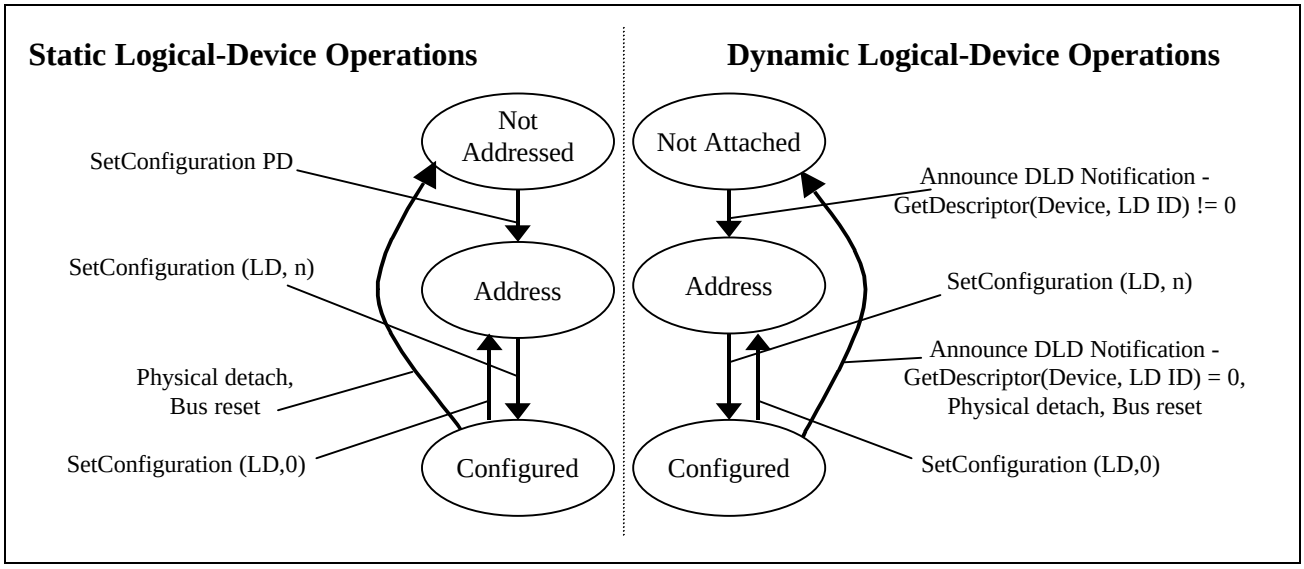
Table 6-1: CCS DLD Notification Request Code

CCS Notification bRequest	Value	Data
NO_OP	0	None
ANNOUNCE_DYNAMIC_LOGICAL_DEVICE	1	None
reserved	2-FF	N/A

NOTE: The CCS DLDs are the first feature to use a CCS notification. To support compatibility with future CCS features that may need notifications, all features must use the same CCS notification format and define new bRequest codes.

7. Dynamic Logical-Device States

The following state diagram displays the similarities between the enumeration processes of static and dynamic logical-devices. The 5 enumeration states of a statically enumerated LD are on the left while the 3 states on the right demonstrate the enumeration of a dynamic LDs.



Note that a static LD only transitions from the *Configured* to *Not Addressed* state when the physical-device is detached from the USB or the physical-device receives a bus reset. DLDs transition through similar states, however the key difference is a *Not Attached* state replaces the static LD's *Not Addressed* state. And DLD Announcements are used to transition in and out of the *Not Attached* state.