CCS Feature Specification: Logical-Devices

Revision 1.0

October 27, 1999

Revision History

Revision	Issue Date	Comments
1.0	10/27/1999	1.0 Release

Contributors

Mark Williams	Microsoft Corporation
John Dunn	Microsoft Corporation
Steve McGowan	Intel Corporation
Kenneth Ray	Microsoft Corporation
Husni Roukbi	Microsoft Corporation
Mats Webjörn	UniAccess AB

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

Table of Contents

1. Introduction

1.1 Purpose

This document describes requirements and specifications for enabling host system software to group sets of interfaces or create one or more logical devices on a single USB device.

1.2 Scope

The information in this document:

- Allows a USB device developer to tie multiple interfaces to a single class code.
- Allows the creation of multiple Logical-Devices in a single physical device.
- Defines methods that are compatible with existing system software implementations.
- Enables host system software to create one abstract data structure for each Logical-Device when the physical device is hot plugged into the host, and use the data structure to independently manage the Logical-Device until the physical device is hot unplugged from the host.

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: Interface Power Management, available at http://www.usb.org.

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

1.4 Terms and Abbreviations

CAPABILITY	A visible function provided by one or more interfaces, such as speakers, keypad, etc.
CCS	Common Class Specification
DEVICE	A physical USB device that performs one or more functions.
DLD	Dynamic Logical-Device
DYNAMIC LOGICAL-DEVICE	A dynamic logical-device capability provided to a host by a device through the use of CCS mechanisms. Dynamic Logical-Devices can be used to create Logical-Devices at run time. Refer to the CCS Feature Specification: Logical Device for more information.
COMPOSITE DEVICE	A USB device that provides more than one function to a host by the declaration of class codes at the interface level.
FUNCTION	A capability provided to a host by a device, such as an ISDN connection, a digital microphone, or speakers.
LOGICAL-DEVICE	A logical capability provided to a host by a device through the use of CCS mechanisms. Logical-Devices can be used to tie multiple interfaces to a single class code and to define multiple functions in a single device.
LD	Logical-Device
PNP	Plug and Play

2. Management Overview

This section provides the background that motivated this feature, an overview of the contents of this document and a brief summary of each of the subsequent sections. It does not establish any requirements or guidelines.

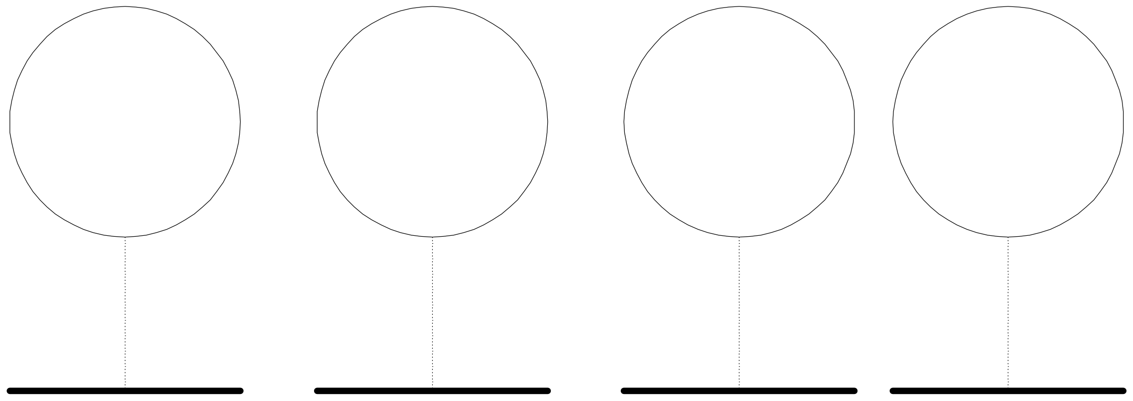
Device vendors can build multiple functions into a device. Each function is self-describing to host system software because the device is required to provide an Interface descriptor for each interface.

In general, when host system software enumerates a USB device, it creates an abstract data structure for each interface on a device. Host system software fills some of the elements of this abstract data structure, which represents the interface, with information it reads from the Interface descriptor. The host maintains dynamic information about the operational state of the device in other parts of the data structure.

A simple strategy on the part of the host would be to build and maintain one data structure for each interface on the device. This simple strategy is shown in the following diagram, where each horizontal solid line represents an interface in the USB device, each circle represents a data structure that represents an interface to the host, and each dotted line represents an association between data structure and interface. The diagram shows a device with four interfaces, and the host has associated one data structure with each interface.

However, a more flexible strategy can have advantages to both builders of system software and builders of devices.

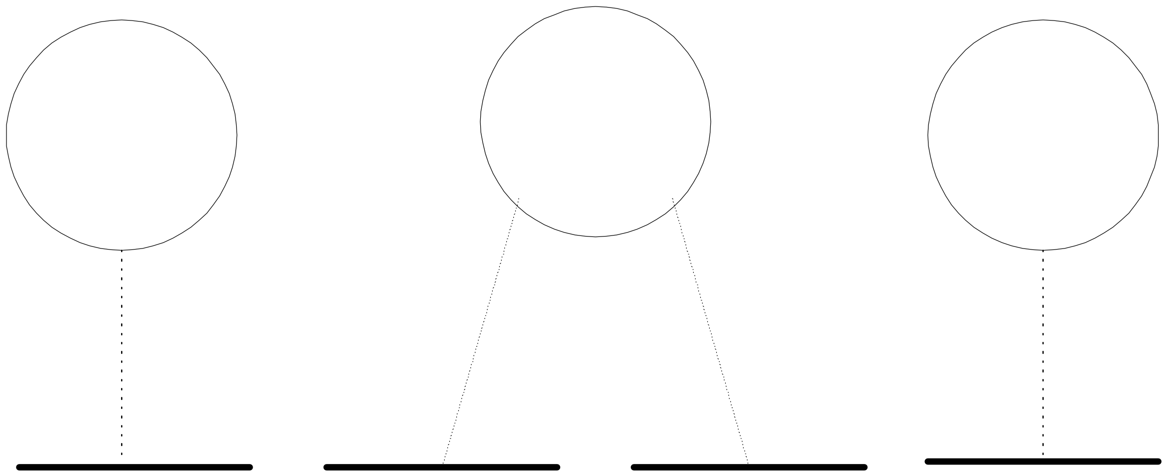
Useable host system software can be built that always uses this simple strategy.



The following diagram shows the strategy enabled by the Common Class Logical-Device feature specified in this document. Once again, a physical device is shown with four interfaces. However, there is a one-to-one association of data structure to interface for only two of the interfaces. The other two interfaces are associated with the same data structure; this is a 2-to-1 mapping of device interfaces to a host data structure.

There is no need to limit the many-to-one mapping of interfaces to a data structure to the 2-to-1 mapping shown in the example diagram. 3-to-1 mappings, 4-to-1 mappings, and so on, are possible.

Each data structure on the host can be called a “Logical-Device.” In order to carry out this scheme, the host requires the device to report the Logical-Devices it contains, which is using the Logical-Device (LD) mechanism described in this document.



The following sections of this specification are:

- Section 3, Functional Characteristics, describes the capabilities provided by the Logical-Device feature and lists the benefits to both the host and the device of the feature.
- Section 4, Operational Model, defines the interactions between the device and host when using the Logical-Device feature.
- Section 5, Descriptors, specifies the format of Logical-Device related descriptors and gives rules for the placement of those descriptors in the list of descriptors returned by a device.
- Section 6, Requests, lists the requests the host uses when using this feature.

3. Functional Characteristics

This section provides a description of each of the functional characteristics and the capabilities provided by the Logical-Device (LD) feature, and lists the benefits to both the host and the device of the feature.

A key requirement of LD's is backwards compatibility with existing system software. The best method of accomplishing this is to provide an out-of-band channel that for access to LD related descriptors and services.

3.1 Background

Device drivers for USB are searched for and located based on descriptor information from the USB device. The "keys" used in the driver search and their priorities are listed in a table in section 3.10 "Locating USB Drivers", of the USB Common Class Specification. Keys for this driver search are based on information from both the Device and Interface descriptors.

A simplified description of the search method follows: In the case of USB Class drivers, the search algorithm first examines the information in the Device descriptor to determine whether a single class driver controls the device as a whole. If a class code of 0 is found in the Device descriptor then the search will be extended to the Interface descriptors. Typically each class code found at the interface level is mapped to a separate device driver.

Problems arise when a single function requires the services of multiple USB classes. For instance, a Communications Device Class (CDC) function defines class codes in two interfaces: a Data class interface for data and a CDC class interface for control. These two interfaces are components of the same function. In order to allow a single driver to control both interfaces of the function, the current driver search algorithm requires that a class code be declared in the device descriptor.

A USB speaker function would declare 3 interfaces: an Audio Class [AudioStreaming Subclass] interface for data, an Audio Class [AudioControl Subclass] interface for control, and a HID class interface for user input. The Audio class does not declare a class code at the device level. Instead it when an Audio class interface is found the audio class driver assumes that all remaining interfaces in the device are related to the audio function. The audio class driver is invoked for both of the audio class interfaces and the driver must also be capable of locating the associated HID interface and taking it over.

Both approaches exclude the possibility of additional functions existing on the device and adopt class specific solutions to a common problem; how does a function tie together multiple interfaces.

LD's address both of these problems and a few more. LD's provide a standard method of tying multiple interfaces to a single function and allowing multiple functions to reside in a single USB device. LD's also allow a function to declare PnP information independent of the device that contains it, thus allowing it to appear as a device in it's own right.

3.2 Feature Requirements

The requirements of a LD feature are:

- A USB device can self-report groupings of interfaces into LD's.
- The host can obtain the LD information from the device.

- The host can identify LD’s provided by alternate vendors.
- The new feature has a minimal impact on the *Universal Serial Bus Specification*.
- The LD feature is backward compatibility with USB device and interface designs that comply with the *Universal Serial Bus Specification*.

3.3 Feature Specification

The defined LD feature has to following capabilities:

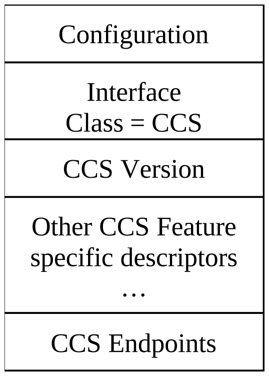
- Associate multiple interfaces into a single function
- Support multiple functions on a single physical device
- Functions to provide PnP enumeration information
- Allow functions to emulate physical devices to preserve existing device drivers

4. Operational Model

4.1 Overview

A USB device indicates its *CCS functionality* by exposing a configuration containing a single Interface descriptor with a CCS class code, immediately followed by a CCS Version descriptor, followed by other CCS Feature specific descriptors, and any permanent CCS related endpoints.

Figure 4. Operational Model-1: CCS Functionality Descriptors



This is to allow current implementations of USB enumerators that look at a Class-code either at a device's Device or Interface descriptor to determine the CCS driver that it should bind to the interface. Using any other method to invoke CCS functionality in a device would mean that the CCS driver could not be automatically invoked without modifying existing enumerators. The CCS Class code, which always must be declared in an Interface descriptor, allows the existing enumeration mechanism to support CCS functionality.

If a CCS capable device needs backward compatibility with existing non-CCS-capable hosts for some of its interfaces then it shall expose a "legacy" 1.0-compliant configuration as its first configuration, and expose CCS functionality on a higher numbered configuration. A CCS-capable host must during enumeration check all configurations on a device and select the one containing CCS functionality.

The CCS Version descriptor identifies the CCS features that the device requires and their versions. For a complete description of CCS Version descriptor refer to "CCS Feature Specification: Version Descriptor".

A device then indicates its capability of grouping interfaces together into Logical-Devices (LD's) by including a CCS Logical-Device descriptor (together with all other CCS Feature specific descriptors). But it is only used to identify the number of static and dynamic LD's that the device supports. The CCS driver must issue GET_DESCRIPTOR requests, overloaded with a Logical-Device ID, in order to obtain descriptors for each specific LD. See section 0 for details.

Each LD is described using the same descriptors used to describe a standard device, that is Device, Configuration, Interface, Endpoint and String descriptors. This way a LD contains all the PnP information the CCS driver needs to enumerate it just as if it was a standard device. See section 0 for details. Note, CCS functionality may not be declared in a LD.

A LD may also be dynamic, meaning that all LD´s may not be obtained initially, but at a later moment. See section 0 for details.

4.2 Addressability

Logical devices are addressed by specifying a Logical-Device ID in the *wIndex* field of a standard request addressed to the device. When *wIndex* is zero, the physical-device is the target. When *wIndex* is non-zero, the specified Logical-Device is the target.

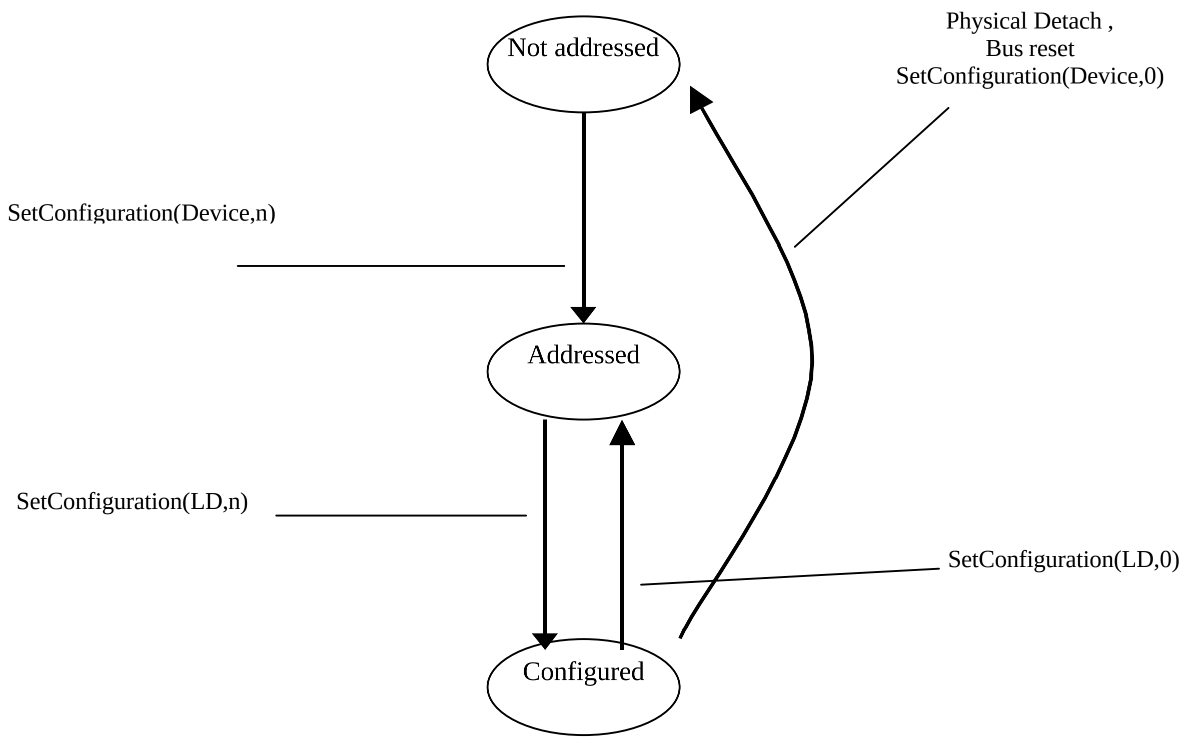
GET/SET-DESCRIPTOR requests for STRING descriptors present a special problem, because *wIndex* is used to specify the string index. A device using the Logical-Device Feature must allocate string indexes across the physical-device and all Logical-Devices without overlap.

See section 0 for a list of standard requests.

4.3 LD state diagram

The following state diagram displays the enumeration processes of static LD's.

Static Logical-Device Operations



5. Descriptors

This section defines the feature-specific descriptors for the Logical-Device feature.

These descriptors enable USB device developers to describe LD’s on their physical-device to the host system software. Host system software uses the information in the descriptors to create one abstract data structure for each LD when the physical-device is hot plugged into the host. Host system software can then use the data structure to independently manage the LD’s until the physical-device is hot unplugged from the host.

5.1 Physical-Device Descriptors

Physical-device descriptors are returned by a GET_DESCRIPTOR request with *wIndex* equal to 0.

5.1.1 Configuration Descriptor

A physical-device uses a standard USB Configuration descriptor.

If the physical-device supports remote wakeup then *bmAttributes* bit D5 (RemoteWakeup) is set to 1.. If any logical-device is expected to generate a wakeup event then the physical-device must set the RemoteWakeup bit in it's Configuration descriptor to 1.

The *MaxPower* field of the physical-device's configuration descriptor represents the maximum power consumption of the physical-device from the USB, in this specific configuration, when the physical-device and all logical-devices are fully operational.

Table 5.Descriptors-1: CCS Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	CONFIGURATION
2	*wTotalLength*	2	Number	See Core Spec.
4	*bNumInterfaces*	1	Number	See Core Spec.
5	*bConfigurationValue*	1	Number	See Core Spec.
6	*iConfiguration*	1	Index	See Core Spec.
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7 See Core Spec. D6 Self Powered D5 Remote Wakeup D4..0 See Core Spec. A physical-device configuration that uses power from the bus sets D6. If the physical-device configuration or any logical-device configurations support remote wakeup, D5 is set to 1.

Offset	Field	Size	Value	Description
8	*MaxPower*	1	mA	Maximum power consumption of USB physical-device from the bus in this specific configuration when the physical-device and all logical-devices are fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA). See Core Spec.

5.1.2 Interface Descriptor

The CCS Interface descriptor is a standard Interface descriptor with *bInterfaceClass* set to CCS Class code.

Table 5. Descriptors-2: CCS Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	Number	See Core Spec.
3	*bAlternateSetting*	1	Number	See Core Spec.
4	*bNumEndpoints*	1	Number	See Core Spec.
5	*bInterfaceClass*	1	Class	CCS Class code.
6	*bInterfaceSubClass*	1	SubClass	Subclass code. Always 0
7	*bInterfaceProtocol*	1	Protocol	Protocol code. Always 0
8	*iInterface*	1	Index	See Core Spec.

5.1.3 CCS Version Descriptor

A device that employs Logical Device functionality must declare a CCS Version Descriptor. The CCS Version descriptor will contain a Version Information Record for the CCS Logical Device feature.

The Feature ID (*bFeatureID*) for the LD feature is 01h.

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

This document describes Revision 1.0 of the Logical-Device Feature.

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

5.1.4 Logical-Device Descriptor

If LD support is indicated in the CCS Version descriptor then a single Logical-Device descriptor must follow the CCS Version descriptor. The Logical-Device descriptor identifies the maximum number of logical-devices supported by a physical-device. For convenience, static and dynamic logical-devices are identified in the LD descriptor. The properties of static LDs are known at enumeration time, while the properties of dynamic LDs are not. Refer to the CCS Feature Specification: Dynamic Logical-Devices for more information on use and operation of dynamic LDs.

Table 5. Descriptors-3: Logical-Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	T.B.D	LOGICAL_DEVICE Descriptor Type
2	*bLogicalDeviceCount*	1	Number	The number of Logical-Devices supported by the device. Logical-Device IDs start at 1 and consecutively increment to *bLogicalDeviceCount*.
3	*bFirstDynamicLD*	1	Number	This is Logical-Device ID of the first Dynamic Logical-Device. An value of 0 indicates that no Dynamic Logical-Devices exist.

bLogicalDeviceCount defines total number of LD's supported by the device. This allows the system to allocate any resources that may be required to support the LD's at device enumeration time.

A subset of the LD's may be dynamic, i.e the characteristics of the LD are not know at enumeration time. LD 1 to LD *bfirstDynamicLD*-1 are static and can be enumerated by retrieving their device and configuration descriptors at enumeration time. LD *bfirstDynamicLD* to LD *bLogicalDeviceCount* are dynamic and are enumerated after the receipt of a Dynamic Logical-Device notification. For a complete description of Dynamic Logical-Devices's refer to "CCS Feature Specification: Dynamic Logical-Device". If no Dynamic Logical-Devices exist *bfirstDynamicLD* equals 0 and the number of static Logical-Devices is defined by *bLogicalDeviceCount*.

5.2 LD Descriptors

The LD descriptors are returned by a GET_DESCRIPTOR request with *wIndex* not equal to 0. LD's can take any descriptors that comply with the *Universal Serial Bus Specification*, with restrictions listed below.

5.2.1 LD Device Descriptor

LDs use standard Device descriptors. The same rules that apply to standard device descriptor members, apply to the LD's device descriptor members.

5.2.2 LD Configuration Descriptor

LDs uses a standard Configuration descriptor. The same rules that apply to standard configuration descriptor members apply to the LD's configuration descriptor members, with the following exceptions.

For each LD *bmAttributes* bit D6 (SelfPower) indicates whether the LD is self powered or draws its power from the USB power provided to the physical device.

The *MaxPower* field of the logical-device's configuration descriptor is set to 0. It is up to the device vendor to ensure that the bus power consumption of all logical-devices in included in the MaxPower field of the physical-devices Configuration descriptor..

5.2.3 LD Interface Descriptor

LDs use a standard Interface descriptor. Note that interface numbers must be allocated across the physical-device and all LDs without overlap. When the recipient of a request is an interface the interface number, which is contained in wIndex, must be unique to unambiguously specify the target interface.

5.2.4 LD Endpoint Descriptor

LDs use a standard Endpoint descriptor. Note that a device can support, at most, 31 endpoint addresses, where endpoint address 0 is always the control endpoint. These endpoints must be allocated across the physical-device and all LDs without overlap. When the recipient of a request is an endpoint the interface number, which is contained in wIndex, must be unique to unambiguously specify the target endpoint.

5.2.5 LD String Descriptor

LDs use a standard String descriptor. Note that string indexes must be allocated across the physical-device and all LDs without overlap.

5.2.6 Supporting Plug and Play and Unknown or Proprietary Class Codes

The physical-device Device descriptor provides vendor and product IDs and optionally, serial number information. Logical-devices provide equivalent information, which can be used for equivalent purposes.

The vendor, product, and serial number information declared in the LD Device descriptor takes precedence over the same information in the physical-device Device descriptor. However a zero value in the *idVendor*, *idProduct*, *iManufacturer*, *iProduct* or *iSerialNumber* member of a LD Device descriptor allows the value to default back to the respective member declared in the physical-device Device descriptor.

Section 3.10 of the Universal Serial Bus Common Class Specification identifies "keys" used to locate device drivers. In many cases the Logical-Device ID (LDID) must be included in the key to identify a unique instance of a Logical-Device.

For instance, if all LDs have the same VID/PID, then VID+PID+LDID must be used for binding drivers, and VID+PID+Serial+LDID is used for device instance identification. Note that if a serial numbers are supplied for individual LDs they must be unique.

5.2.7 Alternate Settings

A LD may include alternate configurations and/or alternate interface settings.

Alternate configurations allow the characteristics of the LD to be varied after enumeration. The default setting for a LD is always configuration zero. The SET_CONFIGURATION request, with *wIndex* equal to the Logical-Device ID, is used to select a configuration for a LD or to return a LD to the default setting. The GET_CONFIGURATION request, with *wIndex* equal to the Logical-Device ID, returns the current selected configuration setting for the selected LD.

The alternate interface settings allow a LD's interfaces and endpoints and/or their characteristics to be varied after the LD has been configured. The semantics of alternate interfaces on LDs are identical to those for a standard device.

The rules for declaring alternate configurations and interfaces in a LD are identical to those for a standard device.

6. Requests

The *Universal Serial Bus Specification* defines a number of standard requests that all devices must support. This section defines how this feature uses those standard requests, if they differ from standard implementations. This section also defines any additional requests defined for this feature.

6.1 Summary – Uses of Standard Requests by a Logical-Device

The following table lists the standard USB requests and describes how each request is used by a LD.

In most cases if the recipient defined in *bmRequestType* is a device, then *wIndex* is overloaded to define the target LD. A *wIndex* value of 0 is always used to access the device itself.

Table 6.Requests-4:LD USB Device Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	CLEAR_FEATURE	Feature Selector	Zero or LD ID	Zero	None
00000001B	CLEAR_FEATURE	Feature Selector	Interface	Zero	None
00000010B	CLEAR_FEATURE	Feature Selector	Endpoint	Zero	None
10000000B	GET_CONFIGURATION	Zero	Zero or LD ID	One	Configuration Value
10000000B	GET_DESCRIPTOR	Descriptor Type (not String) and Descriptor Index	Zero or LD ID	Descriptor Length	Descriptor
10000000B	GET_DESCRIPTOR	Descriptor Type (String) and Descriptor Index	Language ID	Descriptor Length	Descriptor
10000001B	GET_INTERFACE	Zero	Interface	One	Alternate Interface
10000000B	GET_STATUS	Zero	Zero or LD ID	Two	Device Status
10000001B	GET_STATUS	Zero	Interface	Two	Interface Status
10000010B	GET_STATUS	Zero	Endpoint	Two	Endpoint Status
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_CONFIGURATION	Configuration Value	Zero or LD ID	Zero	None

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type (not string) and Descriptor Index	Zero or LD ID	Descriptor Length	Descriptor
00000000B	SET_DESCRIPTOR	Descriptor Type (string) and Descriptor Index	Language ID	Descriptor Length	Descriptor
00000000B	SET_FEATURE	Feature Selector	Zero or LD ID	Zero	None
00000001B	SET_FEATURE	Feature Selector	Interface	Zero	None
00000010B	SET_FEATURE	Feature Selector	Endpoint	Zero	None
00000001B	SET_INTERFACE	Alternate Setting	Interface	Zero	None
10000010B	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

6.2 Device Requests

Standard requests can be sent to a specific LD recipient by placing the Logical-Device ID in the *wIndex* field. There are a few exceptions to this rule and they are discussed below.

6.2.1 GET_DESCRIPTOR Request

This request returns the specified descriptor if the descriptor exists.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Logical-Device ID	Descriptor Length	Descriptor

The *wIndex* field determines whether the device's or a LD's descriptors are returned. A zero returns the device related descriptors and a non-zero value returns the descriptors related to the selected LD.

A host can use the GET_DESCRIPTOR request to get device or configuration information from a LD in the same way that it would retrieve the information from a physical-device. The only difference is the value of *wIndex*.

Since String descriptors are global for a device the syntax for obtaining them is unchanged.

6.2.2 GET_CONFIGURATION Request

This request returns the current device or LD configuration value.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_CONFIGURATION	Zero	Zero or Logical-Device ID	One	Configuration Value

The *wIndex* field determines whether the device's or a LD's configuration value is returned. A zero returns the device configuration value, and a non-zero value returns the configuration value related to the selected LD.

For a complete description on how to get device configuration value refer to chapter 9 in the *Universal Serial Bus Specification*.

A host can use the GET_CONFIGURATION request to identify the configuration value of a particular LD. A device must respond to a valid GET_ CONFIGURATION request by returning the configuration value of the selected LD. A returned value of 0 indicates that the LD is not configured and value of > 0 identifies the *bConfigurationValue* of the selected LD's current configuration.

6.2.3 SET_CONFIGURATION Request

This request sets the device or LD configuration.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONFIGURATION	Configuration Value	Zero or Logical-Device ID	Zero	None

The *wIndex* field determines whether the device's or a LD's configuration state is modified. A zero sets the device configuration state, and a non-zero value sets the configuration state related to the selected LD.

For a complete description on how to set device configuration refer to chapter 9 in the *Universal Serial Bus Specification*.

A host can use the SET_CONFIGURATION request to control the configuration state of a particular LD. A device must respond to a valid SET_ CONFIGURATION request by modifying the state of the target LD. A configuration value of 0 sets the LD to the *Addressed* state, and configuration value > 0 identifies the *bConfigurationValue* of the selected LD's configuration and sets the LD to *Configured* state.

A SET_CONFIGURATION(0) or a SET_CONFIGURATION(n) request to a LD before the device has received a SET_CONFIGURATION(n) request will generate a stall.

A SET_CONFIGURATION(0) or a SET_CONFIGURATION(n) request to a device where n is different than the device's current configuration sets all LD's to *Addressed* state.

Configuring the device does not configure any LD's that it supports.

7. Class Interactions

A feature may choose to make extensive use of other features' definitions to implement its capabilities. The requirements of such interactions are described here.

7.1 Interaction with the Interface Power Management Feature

System software can power manage the individual Logical-Devices by using the Common Class Interface Power Management feature. For the specification of the requests that are available to the host, see section 6.1 of *USB Feature Specification: Interface Power Management*, Revision 1.0 .

8. Example

Consider a USB device that supports a single LD.

The device is enumerated just like any other USB device, the enumerator reads the Device descriptor by issuing a GET_DESCRIPTOR (Device, 0)[1].

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

In the Interface descriptor returned by the following GET_DESCRIPTOR (Config, 0)[1] request the enumerator finds a CCS class code and starts up the CCS driver.

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

The CCS driver executes another GET_DESCRIPTOR (Config, 0) [1] request to obtain the CCS Version descriptor. In the CCS Version descriptor the CCS driver identifies the fact that the device requires LD support. The Logical-Device descriptor which was also returned by the GET_DESCRIPTOR (Config, 0) [1] request identifies that the device supports one LD.

To obtain the descriptors related to LD #1 the CCS driver executes a GET_DESCRIPTOR (Device, 1) [1] request to obtain the device descriptor for the LD, and a GET_DESCRIPTOR (Configuration, 1) [1] to obtain the Configuration, Interface and Endpoint descriptors for the LD.

Once these descriptors are retrieved the CCS driver enumerates the LD in the same way that a standard USB device would have been enumerated.

If the Logical-Device descriptor indicated that the device supported more than one LD then the CCS enumerator would have generated GET_DESCRIPTOR requests for each LD by incrementing the *wIndex*.

[1] Notation: GET_DESCRIPTOR(Descriptor type, *wIndex* value)