

Universal Serial Bus
Communications Class
Subclass Specification for
Ethernet Control Model Devices

Revision 1.2

February 9, 2007

Revision History

Rev	Date	Filename	Comments
1.2	2/9/07		Final edits as per February 2007 CDC meeting

Please send comments or questions to : cdc@usb.org

Copyright © 2007, USB Implementers Forum, Inc.

All rights reserved.

A LICENSE IS HEREBY GRANTED TO REPRODUCE THIS SPECIFICATION FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, IS GRANTED OR INTENDED HEREBY.

USB-IF AND THE AUTHORS OF THIS SPECIFICATION EXPRESSLY DISCLAIM ALL LIABILITY FOR INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. USB-IF AND THE AUTHORS OF THIS SPECIFICATION ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE THE INTELLECTUAL PROPERTY RIGHTS OF OTHERS.

THIS SPECIFICATION IS PROVIDED "AS IS" AND WITH NO WARRANTIES, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE. ALL WARRANTIES ARE EXPRESSLY DISCLAIMED. NO WARRANTY OF MERCHANTABILITY, NO WARRANTY OF NON-INFRINGEMENT, NO WARRANTY OF FITNESS FOR ANY PARTICULAR PURPOSE, AND NO WARRANTY ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

IN NO EVENT WILL USB-IF OR USB-IF MEMBERS BE LIABLE TO ANOTHER FOR THE COST OF PROCURING SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA OR ANY INCIDENTAL, CONSEQUENTIAL, INDIRECT, OR SPECIAL DAMAGES, WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SPECIFICATION, WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

All product names are trademarks, registered trademarks, or service marks of their respective owners.

Contributors, v1.1

Paul E. Berg	Moore Computer Consultants, Inc.
Chuck Brabenac	Intel Corporation
Shelagh Callahan	Intel Corporation
Paul Chehowski	Mitel Corporation
Joe Decuir	Microsoft Corporation
Stefan Eder	Siemens Semiconductors
Ed Endejan	3Com Corporation
Randy Fehr	Northern Telecom
Diego Friedel	AVM
John Howard	Intel Corporation
Ken Lauffenburger	Efficient Networks, Inc.
Ron Lewis	Rockwell Semiconductors
Dan Moore	Diamond Multimedia Systems
Terry Moore	Moore Computer Consultants, Inc.
Andy Nicholson	Microsoft Corporation
Nathan Peacock	Northern Telecom
Dave Perry	Mitel Corporation
Kenny Richards	Microsoft Corporation
Charlie Tai	Intel Corporation
Mats Webjörn	Universal Access

Contributors, V1.2

Bruce Balden	Belcarra
Jun Guo	Broadcom
CC Hung	Mentor Graphics
Dale Self	Symbian
Janne Rand	Nokia
Joe Decuir	MCCI
Joel Silverman	K-Micro
John Turner	Symbian
Ken Taylor	Motorola
Morten Christiansen	Ericsson AB
Peter FitzRandolph	MCCI
Richard Petrie	Nokia
Saleem Mohammed	Synopsys
Tero Soukko	Nokia

Table of Contents

1	Introduction	1
1.1	Purpose	1
1.2	Scope.....	1
1.3	Related Documents	1
1.4	Terms and Abbreviations.....	2
2	Management Overview	3
3	Functional Overview	4
3.1	Function Model	4
3.2	USB Networking Models.....	4
3.3	Common Data Plane Characteristics	4
3.3.1	Segment Delineation	6
3.3.2	Segment Size	6
3.4	Ethernet Networking Control Model.....	7
4	Class-Specific Codes	8
4.1	Communications Class Code	8
4.2	Communications Class Subclass Code.....	8
4.3	Communications Class Protocol Code.....	8
5	Descriptors	9
5.1	Standard USB Descriptor Definitions	9
5.2	Class-Specific Descriptors.....	9
5.3	Functional Descriptors.....	9
5.4	Ethernet Networking Functional Descriptor.....	9
6	Communications Class Specific Messages	12
6.1	Overview.....	12
6.2	Management Element Requests	12
6.2.1	<i>SetEthernetMulticastFilters</i>	13
6.2.2	<i>SetEthernetPowerManagementPatternFilter</i>	13
6.2.3	<i>GetEthernetPowerManagementPatternFilter</i>	14
6.2.4	<i>SetEthernetPacketFilter</i>	14
6.2.5	<i>GetEthernetStatistic</i>	15
6.3	Management Element Notifications.....	17

List of Tables

Table 1 Class Subclass Code.....	8
Table 2 Class Protocol Code.....	8
Table 3: Ethernet Networking Functional Descriptor.....	9

Table 4: Ethernet Statistics Capabilities11

Table 5: Requests — Ethernet Networking Control Model*12

Table 6: Class-Specific Request Codes for Ethernet subclass.....13

Table 7: Power Management Pattern Filter Structure.....14

Table 8: Ethernet Packet Filter Bitmap.....15

Table 9: Ethernet Statistics Feature Selector Codes.....16

Table 10: Notifications — EthernetNetworking Control Model*17

Table 11: Class-Specific Notification Codes for Ethernet subclass.....17

List of Figures

Figure 1 - USB Network Device Example6

Figure 2: Ethernet Networking Model.....7

1 Introduction

1.1 Purpose

The USB Communications Device Class Specification 1.1 contains general Communications Class specifications, and details for seven device subclasses. That specification has been editorially reorganized into a common USB CDC 1.2 specification [USBCDC1.2] and a set of subclass specifications. This should help implementers understand what is necessary for each device subclass and facilitate specification maintenance by the USB Device Working Group.

This document is one of those subclass specifications. It contains material technically identical to that contained in USB CDC 1.1. It is intended to guide implementers of USB-connected devices of the types listed in the following section.

1.2 Scope

This document specifies one device subclass intended for use with Communication devices, based on the Universal Serial Bus Class Definitions for Communication Devices specification [USBCDC1.2]. It supports Ethernet (IEEE 802.3) and similar devices.

The intention of this specification is that all material presented here is technically compatible with the previous version of the USB CDC 1.1 Specification, from which it is derived. Numeric codes are defined for subclass codes, protocol codes, management elements, and notification elements.

In some cases material from [USBCDC1.2] is repeated for clarity. In such cases, [USBCDC1.2] shall be treated as the controlling document.

In this specification, the word ‘shall’ or ‘must’ is used for mandatory requirements, the word ‘should’ is used to express recommendations and the word ‘may’ is used for options.

1.3 Related Documents

Reference	Description
[USB2.0]	Universal Serial Bus Specification, revision 2.0 (also referred to as the USB Specification). http://www.usb.org
[USBCCS1.0]	Universal Serial Bus Common Class Specification, revision 1.0. http://www.usb.org
[USBCDC1.2]	Universal Serial Bus Class Definitions for Communication Devices, Version 1.2. http://www.usb.org .
[USBATM1.2]	Universal Serial Bus Subclass Specification for Asynchronous Transfer Mode (ATM) Devices. http://www.usb.org
[USBWMC1.1]	Universal Serial Bus Subclass Specification for Wireless Mobile Communications, Version 1.1. http://www.usb.org
IEEE 802.3	ISO/IEC 8802-3 (ANSI/IEEE Std 802.3): Information technology — Local and metropolitan area networks — Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, 1993
DOCSIS 1.0	Data Over Cable Service Interface Specifications (DOCSIS) Customer Premises to CPE Interface (CMCI) Interim Specification, SP-CMCI-I02-980317, 1998 http://www.cablemodem.com

1.4 Terms and Abbreviations

TERM	Description
802.3	Second generation networking cabling and signaling, commonly known as Ethernet II.(IEEE 802.3)
BYTE	For the purposes of this document, the definition of a byte is 8 bits.
ETHERNET FRAME	Generic term representing any data frames that may be exchanged over DIX or 802.3 networks.
COMMUNICATIONS INTERFACE	Refers to a USB interface that identifies itself as using the Communications Class definition.
DATA INTERFACE	Refers to a USB interface that identifies itself as using the Data Class definition.
DEVICE MANAGEMENT	Refers to requests and responses that control and configure the operational state of the device. Device management requires the use of a Communications Class interface.
DESCRIPTOR	Data structure used to describe a USB device capability or characteristic
DEVICE	A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical. When used as a non-specific reference, a USB device is either a hub or a function.
ETHERNET FRAME	Generic term representing any data frames that may be exchanged over DIX or 802.3 networks.
FUNCTION	Device capabilities exposed over the USB cable.
MANAGEMENT ELEMENT	Refers to a type of USB pipe that manages the communications device and its interfaces. Currently, only the Default Pipe is used for this purpose.
MASTER INTERFACE	A Communications Class interface, which has been designated the master of zero or more interfaces that implement a complete function in a USB Communication Communications device. This interface will accept management requests for the union.
MULTIFUNCTION DEVICE	A device of peripheral that exposes one or more functions or services to an end user. Exposed services can but do have to be exposed as USB functions.
NOTIFICATION ELEMENT	Refers to a type of USB pipe. Although a notification element is not required to be an interrupt pipe, a notification element is typically defined in this way.
NOTIFICATION ELEMENT	Refers to a type of USB pipe. Although a notification element is not required to be an interrupt pipe, a notification element is typically defined in this way.
PDU	Protocol Data Unit - A combination of the SDU and the current protocol layer's header and/or trailer.
SDU	Service Data Unit – User data and control information created at the upper protocol layers that is transferred transparently through a primitive by layer (N+1) to layer (N) and subsequently to (N-1).
UNION	A relationship between a collection of one or more interfaces that can be considered to form a functional unit.
UNIT	Entity that provides the basic building blocks to describe a protocol stack.

2 Management Overview

This subclass specification includes specifications for devices that connect to IEEE 802 family network devices, commonly referred to as Ethernet. Examples include:

- USB-to-Ethernet (IEEE 802.3) adaptors
- USB-connected cable modems (DOCSIS)
- USB-connected ADSL modems that support PPPoE emulation
- USB-connected Wi-Fi adaptors (IEEE 802.11-family)

Note that for USB-connected ADSL modems that expose the ATM services [USBATM1.2] is applicable.

Devices of these subclasses must conform to:

- USB Specification [USB2.0]
- Device Framework
- Communications Device Class 1.2 [USBCDC1.2]

3 Functional Overview

3.1 Function Model

[USB2.0] defines “function” as a “USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers”. Further, in section 5.2.3, it says “Multiple functions may be packaged into what appears to be a single physical device.... A device that has multiple interfaces controlled independently of each other is referred to as a composite device.” We therefore adopt the term “function” to describe a set of one or more interfaces which taken together provide a capability to the host.

Functions defined in this specification may be used as part of multifunction devices (e.g. [USBWMC 1.1]) or as single-function devices.

3.2 USB Networking Models

A USB Networking device has a type of Communications Class Interface that will be presented to the host for configuring and managing the networking device. Networking devices are typically “Always Connected”, spending all of their time with the “link up”. The Communications Class Interface is primarily used to configure and manage the networking device, not to place calls.

In contrast to a telecommunications device, a networking device will always have at least one associated Data Class interface to exchange network traffic. In a typical host software stack, the same driver that is responsible for configuring and managing the network device is also the client that is a source/sink of networking traffic. An example of such a host resident networking driver is either an ATM or Ethernet driver.

The usage of more than one pair of Communications/Data interfaces will be common for devices that expose more than one interface to the network. For example, an DOCSIS cable modem has two 48 bit MAC addresses. Its first MAC address is used to manage cable modem functions, while the second MAC address is used for subscriber access to the Internet. For a cable modem implementation where its management functions are migrated to the host, two different Communications Class and Data Class interfaces will be exposed to the host (one pair for each of the 2 MAC addresses).

Networking devices are differentiated by their SubClass code, which currently are comprised of the Ethernet Networking Control Model and ATM Networking Control Model [USBATM1.2].

3.3 Common Data Plane Characteristics

The core Data-In/Data-Out pipe mechanism is the same for all networking device models supported by this specification, independent of the media type (e.g., Cable, xDSL, Ethernet) or media data type (e.g., ATM cells, Ethernet frames).

Typical USB-based Networking devices will support bulk transfers as the default configuration to exchange data between a host and the USB device.

While each data packet of a bulk endpoint is limited to the maximum packet size defined in the associated endpoint descriptor, it should be noted that a host might request multiple bulk USB protocol

packets within a single USB frame. For maximum throughput, a Networking device must be prepared to transfer multiple bulk packets within a single USB frame.

Some USB-based Networking device implementations may support isochronous data transfers in addition to (or instead of) bulk transfers. Isochronous transfers guarantee data throughput and bounded latency, consistent with the needs of real-time streams (audio, video). Isochronous data errors are reported to receiver, but no data integrity (i.e., retransmission) is provided by the USB link.

The Data Class Interface Descriptor protocol code for all Networking Control Models is 00h.

USB provides no inherent flow control mechanism for isochronous pipes, and this specification defines no higher level mechanism for doing so. Instead, it is assumed that the host software is responsible for doing traffic shaping as necessary to match any end-to-end negotiation. If the networking device is performing traffic shaping, then either a bulk endpoint should be used, or the flow control methods should be provided using vendor-specific methods.

The Data Class interface of a networking device shall have a minimum of two interface settings. The first setting (the default interface setting) includes no endpoints and therefore no networking traffic is exchanged whenever the default interface setting is selected. One or more additional interface settings are used for normal operation, and therefore each includes a pair of endpoints (one IN, and one OUT) to exchange network traffic. The host shall select an alternate interface setting to initialize the network aspects of the device and to enable the exchange of network traffic.

To recover the network aspects of a device to known states, the host will first select the default interface setting (with no endpoints) and then select the appropriate alternate interface setting. In response to this action the device shall flush device buffers, clear any filters or statistics counters and will cause *NetworkConnection* and *ConnectionSpeedChange* notifications to be sent to the host. The effect of a "reset" on the device physical layer is media-dependent and beyond the scope of this specification.

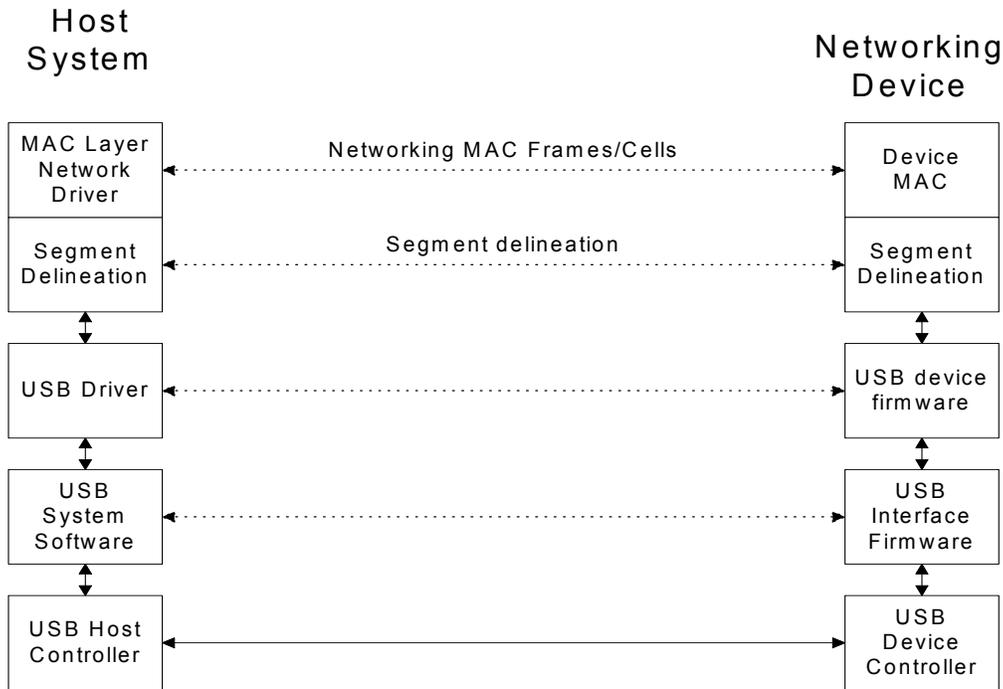


Figure 1 - USB Network Device Example

3.3.1 Segment Delineation

For almost any type of USB attached networking device, a mechanism is needed where both the networking device and the Host can delineate the beginning and ending of a *segment* within the data stream delivered by an endpoint. The meaning and cause of *segment* end is media dependent. E.g. the end of an Ethernet frame

This positive delineation is done using a USB short packet mechanism. When a segment spans N USB packets, the first packet through packet $N-1$ shall be the maximum packet size defined for the USB endpoint. If the N th packet is less than maximum packet size the USB transfer of this short packet will identify the end of the segment. If the N th packet is exactly the maximum packet size, it shall be followed by a zero-length packet (which is a short packet) to assure the end of segment is properly identified.

When transmitting data to the networking device, it is assumed that the client of the host USB driver takes the appropriate actions to cause a short packet to be sent to the networking device. For segments with lengths that are an even multiple of the pipe's "max packet size", the ability to write a buffer of zero length is required to generate this short packet.

3.3.2 Segment Size

The host and the attached network device must negotiate to establish the maximum segment size. The upper limit for this is usually a function of the buffering capacity of the attached device, but there may be other factors involved as well.

For networking devices that exchange Ethernet frames, the maximum size of a segment is also negotiable. Typical Ethernet frames are 1514 bytes or less in length (not including the CRC), but this can be longer (e.g., 802.1Q VLAN tagging).

3.4 Ethernet Networking Control Model

The Ethernet Networking Control Model is used for exchanging Ethernet framed data between the device and host. A Communications Class interface is used to configure and manage various Ethernet functions, where an "Ethernet Networking Control Model" SubClass code is indicated in the descriptor definition of its Communications Class interface.

A Data Class interface is used to exchange Ethernet encapsulated frames sent over USB. These frames shall include everything from the Ethernet destination address (DA) up to the end of the data field. The CRC checksum must not be included for either send or receive data. It is the responsibility of the device hardware to generate and check CRC as required for the specific media. Receive frames that have a bad checksum must not be forwarded to the host. This implies that the device must be able to buffer at least one complete Ethernet frame.

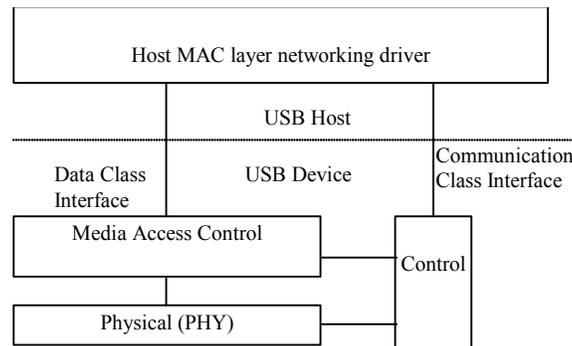


Figure 2: Ethernet Networking Model

Although a typical USB Networking device stays in an "always connected" state, some Networking device management requests are required to properly initialize both the device and the host networking stack. There also may be occasional changes of device configuration or state, e.g., adding multicast filters.

4 Class-Specific Codes

This section lists the codes for the Communications Device Class, Communications Interface Class and Data Interface Class, including subclasses and protocols. These values are used in the *DeviceClass*, *bInterfaceClass*, *bInterfaceSubClass*, and *bInterfaceProtocol* fields of the standard device descriptors as defined in chapter 9 of [USB2.0].

4.1 Communications Class Code

This is defined in [USBCDC1.2].

4.2 Communications Class Subclass Code

The following table defines the Communications Subclass code used in this subclass specification:

Table 1 Class Subclass Code

Code	Subclass
06h	Ethernet Control Model

4.3 Communications Class Protocol Code

[USBCDC1.2] defines Communications Class Protocols.

The following table lists the Protocol code used in this subclass specification:

Table 2 Class Protocol Code

Code	Protocol
00h	No class specific protocol required

If a Communications Class interface appears with multiple alternate settings, all alternate settings for that interface shall have the same *bInterfaceClass*, *bInterfaceSubclass* and *bInterfaceProtocol* codes.

5 Descriptors

5.1 Standard USB Descriptor Definitions

Devices that conform to this subclass specification need to implement the standard USB descriptors for the Communications Device Class, Communications Interface Class and Data Interface Class. These are defined in [USBCDC1.2].

5.2 Class-Specific Descriptors

Devices that conform to this subclass specification may need to implement class-specific descriptors for the Communications Interface Class and Data Interface Class. These are defined in [USBCDC 1.2].

5.3 Functional Descriptors

Functional descriptors describe the content of the class-specific information within an Interface descriptor. Functional descriptors all start with a common header descriptor, which allows host software to easily parse the contents of class-specific descriptors. Each class-specific descriptor consists of one or more functional descriptors. Although the Communications Class currently defines class-specific descriptor information, the Data Class does not.

[USBCDC1.2] specification describes functional descriptors that may be used in all Communications Subclasses. These include:

- Header Functional Descriptor
- Union Functional Descriptor
- Country Selection Functional Descriptor

The following Functional Descriptors are specific to Ethernet subclass devices.

5.4 Ethernet Networking Functional Descriptor

The Ethernet Networking functional descriptor describes the operational modes supported by the Communications Class interface, as defined in Section 3.4, with the SubClass code of Ethernet Networking Control. It can only occur within the class-specific portion of an Interface descriptor.

Table 3: Ethernet Networking Functional Descriptor

Offset	Field	Size	Value	Description
0	<i>bFunctionLength</i>	1	Number	Size of this functional descriptor, in bytes.
1	<i>bDescriptorType</i>	1	Constant	CS_INTERFACE
2	<i>bDescriptorSubtype</i>	1	Constant	Ethernet Networking functional descriptor subtype as defined in [USBCDC1.2]

Offset	Field	Size	Value	Description
3	<i>iMACAddress</i>	1	Index	Index of string descriptor. The string descriptor holds the 48bit Ethernet MAC address. The Unicode representation of the MAC address is as follows: the first Unicode character represents the high order nibble of the first byte of the MAC address in network byte order. The next character represents the next 4 bits and so on. The Unicode character is chosen from the set of values 30h through 39h and 41h through 46h (0-9 and A-F). <i>iMACAddress</i> can not be zero and the Unicode representation must be 12 characters long. For example, the MAC Address 0123456789ABh is represented as the Unicode string "0123456789AB".
4	<i>bmEthernetStatistics</i>	4	Bitmap	Indicates which Ethernet statistics functions the device collects. If a bit is set to 0, the host network driver is expected to keep count for the corresponding statistic (if able). See Table 4 for a detailed listing of possible Ethernet statistics. Support for any of these statistics is optional. If none of these bits are set, the device does not support the <i>GetEthernetStatistic</i> request.
8	<i>wMaxSegmentSize</i>	2	Number	The maximum segment size that the Ethernet device is capable of supporting. This is typically 1514 bytes, but could be extended (e.g., 802.1d VLAN)
10	<i>wNumberMCFilters</i>	2	Bitmap	Contains the number of multicast filters that can be configured by the host. D15: 0 - The device performs perfect multicast address filtering (no hashing). 1- The device uses imperfect multicast address filtering (hashing). Here, the host software driver must perform further qualification itself to achieve perfect filtering. D14..0: Indicates the number of multicast address filters supported by the device (0 to 32767). If the host finds the number of filters supported by the device to be inadequate, it may choose to set the device's Ethernet Packet Filter to forward all multicast frames to the host, performing all multicast filtering in software instead. If this value is 0, the device does not support the <i>SetEthernetMulticastFilters</i> request.
12	<i>bNumberPowerFilters</i>	1	Number	Contains the number of pattern filters that are available for causing wake-up of the host.

Table 4: Ethernet Statistics Capabilities

Offset	Field	Description
D0	XMIT_OK	Frames transmitted without errors
D1	RVC_OK	Frames received without errors
D2	XMIT_ERROR	Frames not transmitted, or transmitted with errors
D3	RCV_ERROR	Frames received with errors that are not delivered to the USB host.
D4	RCV_NO_BUFFER	Frame missed, no buffers
D5	DIRECTED_BYTES_XMIT	Directed bytes transmitted without errors
D6	DIRECTED_FRAMES_XMIT	Directed frames transmitted without errors
D7	MULTICAST_BYTES_XMIT	Multicast bytes transmitted without errors
D8	MULTICAST_FRAMES_XMIT	Multicast frames transmitted without errors
D9	BROADCAST_BYTES_XMIT	Broadcast bytes transmitted without errors
D10	BROADCAST_FRAMES_XMIT	Broadcast frames transmitted without errors
D11	DIRECTED_BYTES_RCV	Directed bytes received without errors
D12	DIRECTED_FRAMES_RCV	Directed frames received without errors
D13	MULTICAST_BYTES_RCV	Multicast bytes received without errors
D14	MULTICAST_FRAMES_RCV	Multicast frames received without errors
D15	BROADCAST_BYTES_RCV	Broadcast bytes received without errors
D16	BROADCAST_FRAMES_RCV	Broadcast frames received without errors
D17	RCV_CRC_ERROR	Frames received with circular redundancy check (CRC) or frame check sequence (FCS) error
D18	TRANSMIT_QUEUE_LENGTH	Length of transmit queue
D19	RCV_ERROR_ALIGNMENT	Frames received with alignment error
D20	XMIT_ONE_COLLISION	Frames transmitted with one collision
D21	XMIT_MORE_COLLISIONS	Frames transmitted with more than one collision
D22	XMIT_DEFERRED	Frames transmitted after deferral
D23	XMIT_MAX_COLLISIONS	Frames not transmitted due to collisions
D24	RCV_OVERRUN	Frames not received due to overrun
D25	XMIT_UNDERRUN	Frames not transmitted due to underrun
D26	XMIT_HEARTBEAT_FAILURE	Frames transmitted with heartbeat failure
D27	XMIT_TIMES_CR_S_LOST	Times carrier sense signal lost during transmission
D28	XMIT_LATE_COLLISIONS	Late collisions detected
D29-D31	RESERVED	Must be set to zero

6 Communications Class Specific Messages

6.1 Overview

The Communications Interface Class supports the standard requests defined in Chapter 9 of [USB2.0]. In addition, the Communications Interface Class has some class-specific requests and notifications. These are used for device and call management.

Requests for controlling the interface between the USB Ethernet device are presented in Section 6.2. There are also some additional signals that shall go back to the host as notifications, which are represented in Section 6.3. These requests and notifications are transported via the Communications Class interface for the device.

6.2 Management Element Requests

Devices conforming to this subclass specification use the following requests. Table 5 lists all of the class-specific request codes that are valid for a Communications Class interface with a Communications Class SubClass code of Ethernet Networking Control Model.

Table 5: Requests — Ethernet Networking Control Model*

Request	Description	Req'd/Opt	reference
<i>SendEncapsulatedCommand</i>	Issues a command in the format of the supported control protocol. The intent of this mechanism is to support networking devices (e.g. host-based cable modems) that require an additional vendor-defined interface for media specific hardware configuration and management.	Optional	[USBCDC1.2]
<i>GetEncapsulatedResponse</i>	Requests a response in the format of the supported control protocol.	Optional	[USBCDC1.2]
<i>SetEthernetMulticastFilters</i>	As applications are loaded and unloaded on the host, the networking transport will instruct the device's MAC driver to change settings of the Networking device's multicast filters.	Optional	6.2.1
<i>SetEthernetPowerManagementPatternFilter</i>	Some hosts are able to conserve energy and stay quiet in a "sleeping" state while not being used. USB Networking devices may provide special pattern filtering hardware that enables it to wake up the attached host on demand when something is attempting to contact the host (e.g. an incoming web browser connection). Primitives are needed in management plane to negotiate the setting of these special filters	Optional **	6.2.2
<i>GetEthernetPowerManagementPatternFilter</i>	Retrieves the status of the above power management pattern filter setting	Optional **	6.2.3
<i>SetEthernetPacketFilter</i>	Sets device filter for running a network analyzer application on the host machine	Required	6.2.4
<i>GetEthernetStatistic</i>	Retrieves Ethernet device statistics such as frames transmitted, frames received, and bad frames received.	Optional	6.2.5

* These requests are specific to the Communications Class.

** If the *SetEthernetPowerManagementPatternFilter* command is supported, then the *GetEthernetPowerManagementPatternFilter* request must be supported as well.

This following describes the requests that are specific to the Communications Interface Class ECM Subclass.

Table 6: Class-Specific Request Codes for Ethernet subclass

Request	Value
SET_ETHERNET_MULTICAST_FILTERS	40h
SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	41h
GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	42h
SET_ETHERNET_PACKET_FILTER	43h
GET_ETHERNET_STATISTIC	44h
RESERVED (future use)	45h-4Fh

6.2.1 SetEthernetMulticastFilters

This request sets the Ethernet device multicast filters as specified in the sequential list of 48 bit Ethernet multicast addresses.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_MULTICAST_FILTER	Number of filters (N)	Interface	N * 6	A list of N 48 bit Multicast addresses, in network byte order

If the host wishes to change a single multicast filter in the device, it must reprogram the entire list of filters using this request. This sequential programming method for the entire multicast list is well-suited for devices that use hashing techniques.

Although the Data field for this request might be quite large, devices with limited buffering capacity may use NAKs as necessary to process (e.g. hash) a small number of multicast addresses at a time.

6.2.2 SetEthernetPowerManagementPatternFilter

This request sets up the specified Ethernet power management pattern filter as described in the data structure.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	Filter number	Interface	Size of structure	Power management pattern filter structure (Table 7)

Some hosts are able to conserve energy and stay quiet in a “sleeping” state while not being used. USB Networking devices may provide special pattern filtering hardware that enables it to wake up the attached host on demand when something is attempting to contact the host (e.g. an incoming web browser connection).

NOTE: To enable remote wake-up, additional steps must be completed that are described in [USB2.0].

If the host simply wishes to clear (remove) any previous setting for the specified pattern filter, the value of wLength is set to Zero and no Data field (pattern filter structure) follows.

If the specified pattern is not able to fit into the device, any pattern previously loaded is considered destroyed, and the device must set a value of FALSE (0x0000) into the associated status that will be read by the *GetEthernetPowerManagementPatternFilter* request.

Table 7: Power Management Pattern Filter Structure

Field	Size	Value	Description
MaskSize	2	Number	Contains the size (in bytes) of the Mask.
Mask	MaskSize	Bitmask	Each byte of Mask contains 8 masking bits, where each one of them represents whether or not the associated byte of the Pattern should be compared with what is seen on the media by the networking device. The least significant bit (D0) of the first Mask byte is associated with the first byte of the pattern, which starts at the Destination Address (DA) of the Ethernet frame. If the last byte of Mask contains trailing zeros in its highest order bits, the associated bytes in the Pattern field are not sent in this request.
Pattern	Specified by Mask	Number	This is a string of bytes to perform pattern matching on, starting from offset 0 of the Ethernet frame (the Destination Address).

6.2.3 *GetEthernetPowerManagementPatternFilter*

This request retrieves the status of the specified Ethernet power management pattern filter from the device. If the device has an active pattern set for the specified filter, a TRUE (0x0001) will be returned. If a FALSE (0x0000) is returned, either no pattern has yet been set for the specified filter, or the prior attempt by the host software to set this filter was not successful (i.e., was not able to fit).

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	Filter number	Interface	2	Pattern active boolean

6.2.4 *SetEthernetPacketFilter*

This request is used to configure device Ethernet packet filter settings. The Packet Filter is the inclusive OR of the bitmap shown in Table 8. Though network adapters for faster buses (e.g., PCI) may offer other hardware filters, the medium speed networking devices (< 10Mbit/s) attached via USB are only required to support promiscuous and all multicast modes. The host networking software driver is responsible for performing additional filtering as required.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_PACKET_FILTER	Packet Filter Bitmap	Interface	Zero	None

Note that for some device types, the ability to run in promiscuous mode may be severely restricted or prohibited. For example, DOCSIS cable modems are only permitted to forward certain frames to its attached host. Even if forwarding of all frames were allowed, the raw cable modem downstream rate available on the RF interface can be many times the maximum USB throughput.

Table 8: Ethernet Packet Filter Bitmap

Bit position	Description
D15..D5	RESERVED (Reset to zero)
D4	PACKET_TYPE_MULTICAST 1: All multicast packets enumerated in the device's multicast address list are forwarded up to the host. (required) 0: Disabled. The ability to disable forwarding of these multicast packets is optional. ***
D3	PACKET_TYPE_BROADCAST 1: All broadcast packets received by the networking device are forwarded up to the host. (required) 0: Disabled. The ability to disable forwarding of broadcast packets is optional. ***
D2	PACKET_TYPE_DIRECTED 1: Directed packets received containing a destination address equal to the MAC address of the networking device are forwarded up to the host (required) 0: Disabled. The ability to disable forwarding of directed packets is optional. ***
D1	PACKET_TYPE_ALL_MULTICAST 1: ALL multicast frames received by the networking device are forwarded up to the host, not just the ones enumerated in the device's multicast address list (required) 0: Disabled.
D0	PACKET_TYPE_PROMISCUOUS: 1: ALL frames received by the networking device are forwarded up to the host (required) 0: Disabled.

*** Support for inhibiting (Dx = 0) the forwarding of "ordinary" directed, multicast and broadcast packets to the host is optional. Since there are no associated descriptors for the device to designate which filters are supported by a particular device, the host must blindly set these bits as desired, filtering out these undesired packets in host software should they appear.

6.2.5 GetEthernetStatistic

This request is used to retrieve a statistic based on the feature selector. The value returned indicates the number of matching frames with the specified statistic that have occurred since the device has been powered on or reset. This number is a 32 bit unsigned integer, which is incremented at each occurrence, and will be wrapped to 0 if reaching the maximum value.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	GET_ETHERNET_STATISTIC	Feature Selector	Interface	4	32 bit unsigned integer

Table 9: Ethernet Statistics Feature Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
XMIT_OK	01h	Interface	4	Frames transmitted without errors
RCV_OK	02h	Interface	4	Frames received without errors
XMIT_ERROR	03h	Interface	4	Frames not transmitted, or transmitted with errors
RCV_ERROR	04h	Interface	4	Frames received with errors
RCV_NO_BUFFER	05h	Interface	4	Frames missed, no buffers
DIRECTED_BYTES_XMIT	06h	Interface	4	Directed bytes transmitted without errors
DIRECTED_FRAMES_XMIT	07h	Interface	4	Directed frames transmitted without errors
MULTICAST_BYTES_XMIT	08h	Interface	4	Multicast bytes transmitted without errors
MULTICAST_FRAMES_XMIT	09h	Interface	4	Multicast frames transmitted without errors
BROADCAST_BYTES_XMIT	0Ah	Interface	4	Broadcast bytes transmitted without errors
BROADCAST_FRAMES_XMIT	0Bh	Interface	4	Broadcast frames transmitted without errors
DIRECTED_BYTES_RCV	0Ch	Interface	4	Directed bytes received without errors
DIRECTED_FRAMES_RCV	0Dh	Interface	4	Directed frames received without errors
MULTICAST_BYTES_RCV	0Eh	Interface	4	Multicast bytes received without errors
MULTICAST_FRAMES_RCV	0Fh	Interface	4	Multicast frames received without errors
BROADCAST_BYTES_RCV	10h	Interface	4	Broadcast bytes received without errors
BROADCAST_FRAMES_RCV	11h	Interface	4	Broadcast frames received without errors
RCV_CRC_ERROR	12h	Interface	4	Frames received with circular redundancy check (CRC) or frame check sequence (FCS) error
TRANSMIT_QUEUE_LENGTH	13h	Interface	4	Length of transmit queue
RCV_ERROR_ALIGNMENT	14h	Interface	4	Frames received with alignment error
XMIT_ONE_COLLISION	15h	Interface	4	Frames transmitted with one collision
XMIT_MORE_COLLISIONS	16h	Interface	4	Frames transmitted with more than one collision
XMIT_DEFERRED	17h	Interface	4	Frames transmitted after deferral
XMIT_MAX_COLLISIONS	18h	Interface	4	Frames not transmitted due to collisions
RCV_OVERRUN	19h	Interface	4	Frames not received due to overrun
XMIT_UNDERRUN	1Ah	Interface	4	Frames not transmitted due to underrun
XMIT_HEARTBEAT_FAILURE	1Bh	Interface	4	Frames transmitted with heartbeat failure
XMIT_TIMES_CR_S_LOST	1Ch	Interface	4	Times carrier sense signal lost during transmission
XMIT_LATE_COLLISIONS	1Dh	Interface	4	Late collisions detected

Refer to the ISO/IEC 8802-3 (ANSI/IEEE Std 802.3) specification for additional information on the meaning of each of these statistics.

6.3 Management Element Notifications

[USBCDC1.2] defines the common Communications Interface Class notifications that the device uses to notify the host of interface or endpoint events. These requests are sent over the management element and can apply to different device views as defined by the Communications Class interface codes.

The class-specific notification codes valid for a Communications Class interface with a Communications Class SubClass code of Ethernet Control Model are listed in the following Table 10, and the class codes are listed in Table 11.

Table 10: Notifications — EthernetNetworking Control Model*

Notification	Description	Req'd/Opt	reference
<i>NetworkConnection</i>	Reports whether or not the physical layer (modem, Ethernet PHY, etc.) link is up.	Required	[USBCDC1.2]
<i>ResponseAvailable</i>	Notification to host to issue a <i>GetEncapsulateResponse</i> request.	Optional	[USBCDC1.2]
<i>ConnectionSpeedChange</i>	Reports a change in upstream or downstream speed of the networking device connection.	Required	[USBCDC1.2]

* These notifications are specific to the Communications Class.

Table 11: Class-Specific Notification Codes for Ethernet subclass

Request	Value
NETWORK_CONNECTION	00h
RESPONSE_AVAILABLE	01h
CONNECTION_SPEED_CHANGE	2Ah