Universal Serial Bus Communications Class Subclass Specifications for Network Control Model Devices

Revision 1.0 (Errata 1)

November 24, 2010

Revision History

Rev	Date	Filename	Comments
1.0	4/30/09		Initial release
1.0 (Errata 1)	11/24/10		Merged Errata from “NCMErrata-2010-11-19a.docx”

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

Copyright © 2009-2010, 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

Bruce Balden	Belcarra
Stuart Lynne	Belcarra
Morten Christiansen	Ericsson
Alan Berkema	Hewlett Packard
Joel Silverman	K-Micro
Paul E. Berg	MCCI
Joe Decuir	MCCI, CSR
Peter FitzRandolph	MCCI
Terry Moore	MCCI
Thirumalai Rajan	MCCI
Dan Sternglass	MCCI
Ken Taylor	Motorola
David Willcox	Motorola
Kenji Oguma	NEC
Richard Petrie	Nokia
Janne Rand	Nokia
Tero Soukko	Nokia
Takashi Ninjouji	NTT DoCoMo
Dale Self	Symbian
John Turner	Symbian
Saleem Mohammad	Synopsis

Table of Contents

List of figures

List of Tables

1 Introduction

1.1 Purpose

The Communications Class Network Control Model (NCM) Subclass is a protocol by which USB hosts and devices can efficiently exchange Ethernet frames. These Ethernet frames may convey IPv4 or IPv6 datagrams that are transported over a communications network. NCM is intended to be used with high-speed network attachments such as HSPA and LTE data services.

This specification builds upon the USB Communications Device Class subclass specification for Ethernet Control Model devices [USBECM12], with improvements to support much higher data rates:

- Multiple Ethernet frames can be aggregated into single USB transfers.
- In order to minimize overhead when processing the Ethernet frames within the USB device, NCM functions can specify their preferences for how Ethernet frames may be best placed within a USB transfer.

1.2 Scope

This document specifies a new control and data plane for networking devices, as a subclass based on the Universal Serial Bus Class Definitions for Communications Devices specification [USBCDC12]. It supports Ethernet [IEEE802.3] and similar networking techniques.

This specification defines the following material applicable to NCM functions:

- The class-specific contents of standard descriptors.
- Additional class-specific descriptors.
- Required interface and endpoint structure.
- Class-specific commands.
- Class-specific notifications.
- The format of data that is exchanged with the host.
- The required behavior.

Behavior that is required of all USB devices and hosts is defined by [USB30] and [WUSB10]. Behavior that is required of all Communications devices is defined by [USBCDC12]. Some commands and notifications are based on material defined in [USBECM12].

1.3 Other USB Networking Specifications

At time of writing, three other USB networking subclasses were defined:

1. Ethernet Control Model (ECM) [USBECM12]
2. Ethernet Emulation Model (EEM) [USBEEM10]

3. ATM Networking Control Model [USBATM12]

ECM and NCM are both applicable to IEEE 802.3 type Ethernet networking functions that can carry IP traffic to an external network. ECM was designed for USB full speed devices, especially to support DOCSIS 1.0 Cable Modems. Although ECM is functionally complete, it does not scale well in throughput or efficiency to higher USB speeds and higher network speeds. NCM draws on the experience gained from ECM implementations, and adjusts the data transfer protocol to make it substantially more efficient.

EEM is intended for use in communicating with devices, using Ethernet frames as the next layer of transport. It is not intended for use with routing or Internet connectivity devices, although this use is not prohibited.

[USBATM12] is applicable to USB-connected devices like ADSL modems which expose ATM traffic directly. Rather than transporting Ethernet frames, [USBATM12] functions send and receive traffic that is broken up into ATM cells and encoded using AAL-2, AAL-4 or AAL-5. Although some of the insights that led to the design of NCM came from experience with [USBATM12] implementations, the two specifications are addressing substantially different target applications.

1.4 Editorial Notes

In some cases material from [USBCDC12] or [USBECM12] is repeated for clarity. In such cases, [USBCDC12] or [USBECM12] shall be treated as the controlling document, unless a change is specifically indicated by this NCM specification.

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

1.5 Related Documents

[ECMA368]	Ecma-368, High Rate Ultra Wideband PHY and MAC Standard, 2005
[IEC60027-2]	IEC 60027-2, Second edition, 2000-11, *Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics*
[IEEE802.11]	IEEE 802.11 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999
[IEEE802.16]	IEEE 802.16 Part 16: Air Interface for Fixed Broadband Wireless Access Systems, 2004
[IEEE802.3]	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
[USB20]	Universal Serial Bus Specification, revision 2.0. **http://www.usb.org**
[USB30]	Universal Serial Bus Specification, revision 3.0. **http://www.usb.org**. Unless otherwise specified, any reference to [USB30] includes [USB20] by reference, especially when referring to full- and high-speed devices.
[USBATM12]	USB Subclass Specification for ATM Control Model, Revision 1.2 **http://www.usb.org**
[USBCCS10]	Universal Serial Bus Common Class Specification, revision 1.0. **http://www.usb.org**
[USBCDC12]	Universal Serial Bus Class Definitions for Communications Devices, Revision 1.2. **http://www.usb.org**.
[USBECM12]	USB Subclass Specification for Ethernet Control Model, Revision 1.2 **http://www.usb.org**
[USBEEM10]	USB Subclass Specification for Ethernet Emulation Model, Revision 1.0 **http://www.usb.org**
[USBWMC11]	Universal Serial Bus Subclass Specification for Wireless Mobile Communications, Version 1.1. **http://www.usb.org**
[WUSB10]	Wireless Universal Serial Bus Specification, version 1.0, **http://www.usb.org/wusb**

1.6 Terms and Abbreviations

Term	Description
802.3	Second generation networking cabling and signaling, commonly known as Ethernet II.(See [IEEE802.3])
Communications Interface	A USB interface that has *bInterfaceClass* set to the class code defined for Communications Class. (See [USBCDC12])
Composite Device	A device or peripheral that exposes two or more functions, each such function being associated with one or more USB Interfaces.
Data Interface	A USB interface that has *bInterfaceClass* set to the class code defined for Data Class. (See [USBCDC12])
Datagram	A collection of bytes forming a single item of information, passed as a unit from source to destination.
Descriptor	Data structure used to describe a USB device capability or characteristic.
Device	A logical or physical entity that receives a device address during enumeration.
Ethernet frame	Generic term representing the various kinds of datagrams that may be exchanged over DIX or 802.3 networks.
Function	A collection of one or more interfaces in a USB device, which taken together present a specific capability of the device to the host.
GiB	Gigabinary Bytes: 2^{30} bytes [IEC60027-2]
KiB	Kilobinary bytes, 1024 bytes [IEC60027-2]
LAN	Local Area Network, e.g. [IEEE802.3].
NCM Communications Interface	A Communications Interface with *bInterfaceSubclass* set to the code defined in Table 4-1.
NCM Data Interface	A Data Interface that is identified as a subordinate interface in a UNION descriptor that is associated with an NCM Communications Interface.
NDP	NCM Datagram Pointer: NTB structure that delineates Datagrams (typically Ethernet frames) within an NTB, see Table 3-3. Depending on the NTB format, an NDB may use 16-bit or 32-bit offsets.
NDP Entry	Data structure in an NDP, giving the offset and the length of a single datagram. See section 3.3.
NTB	NCM Transfer Block: a data structure for efficient USB encapsulation of one or more datagrams. Each NTB is designed to be a single USB transfer. See Figure 3-1 and Figure 3-2.
NTB Format	NTBs have two formats. A “16 bit NTB” primarily uses fields that are 16 bits wide; therefore, such an NTB is limited to a maximum size of 64 KiB. A “32-bit NTB” primarily uses fields that are 32 bits wide; therefore, a 32-bit NTB may be as long as 4 GiB.
NTH	NTB Header: a data structure at the front of each NTB, which provides the information needed to validate the NTB and begin decoding. Depending on the NTB format, this may be either a 16-bit NTH16 (Table 3-1) or a 32-bit NTH32 (Table 3-2).
Union	A relationship among a collection of one or more interfaces that can be considered to form a functional unit.
WUSB	Wireless USB, as defined by [WUSB10].

2 Overview

This subclass specification includes specifications for USB-connected external data network adaptors that model IEEE 802 family Layer 2 networking functionality.

Devices of these subclasses shall conform to:

- USB Specification [USB30], or
- Wireless USB Specification [WUSB10], and
- Communications Device Class 1.2 [USBCDC12]

The principal advantage of using NCM lies in its method of transporting multiple datagrams inside single USB bulk transfers. In addition to reducing interrupt overhead, the NCM specification allows the sender of data to arrange the datagrams within the transfer so that the receiver need do minimal copying after receipt.

This specification defines two ways of encapsulating datagrams, one allowing transfers up to 64KiB (up to forty (40) 1514-byte [IEEE802.3] Ethernet frames), and another for transfers of up to 4GiB, supporting thereby both [USB20] High Speed and [USB30] SuperSpeed data rates.

Devices implementing the NCM function may be composite devices as described by [USBWMC11].

An NCM function is implemented by an NCM Communications Interface and an NCM Data Interface. The NCM Communications Interface is used for configuring and managing the networking function. The NCM Data Interface is used for transporting data, using the endpoints defined by that interface. Generally, the NCM Communications Interface and the NCM Data Interface are managed by a single driver on the USB host. The logical connections between host driver and NCM function are shown in Figure 2-1, and the control and data connections are shown schematically in Figure 2-2.

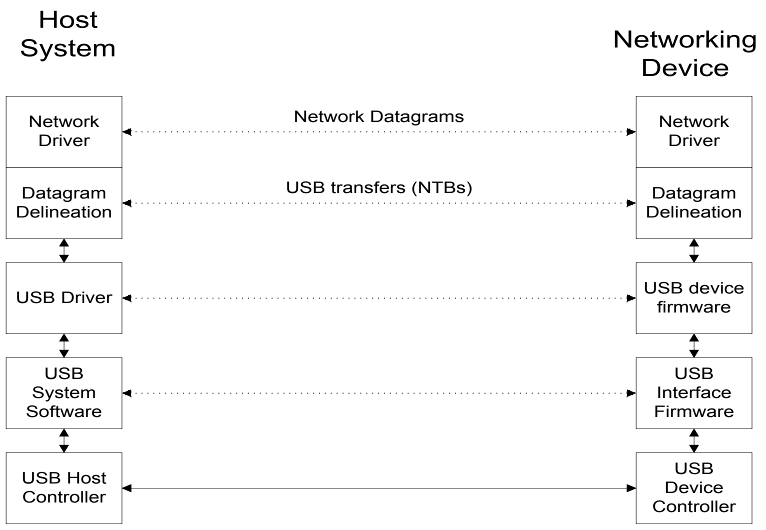

Figure 2-1 - NCM Function Example

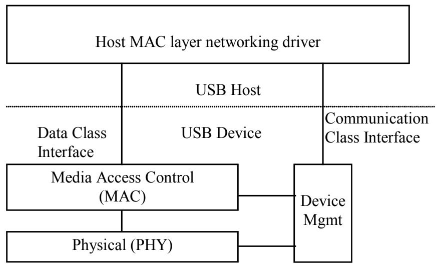
Host MAC layer networking driver
USB Host
Communication Class Interface
Data Class Interface
USB Device
Media Access Control (MAC)
Device Mgmt
Physical (PHY)

Figure 2-2 - Network Control Model

Although an NCM function may stay in an "always connected" state, some management requests may be required to properly initialize both the function and the host networking stack. There also may be occasional changes of function configuration or state, e.g., adding multicast filters in response to changes in the host networking stack.

3 Data Transport

3.1 Overview

NCM allows device and host to efficiently transfer one or more Ethernet frames using a single USB transfer. The USB transfer is formatted as a NCM Transfer Block (NTB).

Figure 3-1 and Figure 3-2 outline the structure of the NTB. Each NTB consists of several components.

- It begins with an NCM Transfer Header ("NTH"). This identifies the transfer as an NTB, and provides basic information about the contents of the NTB to the receiver (3.3.3.1).
- The NTH effectively points to the head of a list of NDP structures (NCM Datagram Pointers). Each NDP in turn points to one or more Ethernet frames encapsulated within the NTB (3.3.3.2).
- Finally, the NTB contains the Ethernet frames themselves (3.3.3.3).

Within any given NTB, the NTH always must be first; but the other items may occur in arbitrary order.

There are two kinds of NTB. NTBs that are shorter than 65,536 bytes in length can be represented as "sixteen bit NTBs" (NTB-16). NTBs of up to 4 GiB in size can be represented as "thirty-two bit NTBs" (NTB-32). The structures are abstractly the same, but NTB-16 form primarily uses 16-bit fields. The two formats are shown in Figure 3-1 and Figure 3-2, respectively.

Although two formats are defined, a function only uses one format or the other at a given time. The same format is used for IN and OUT transfers. The host selects the format to be used.

NTBs cannot be of arbitrary size; functions normally advertise their upper limits to the host. NCM allows functions to have different maximum NTB sizes for transmit and receive. The sender of an NTB may vary the size of NTBs as needed.

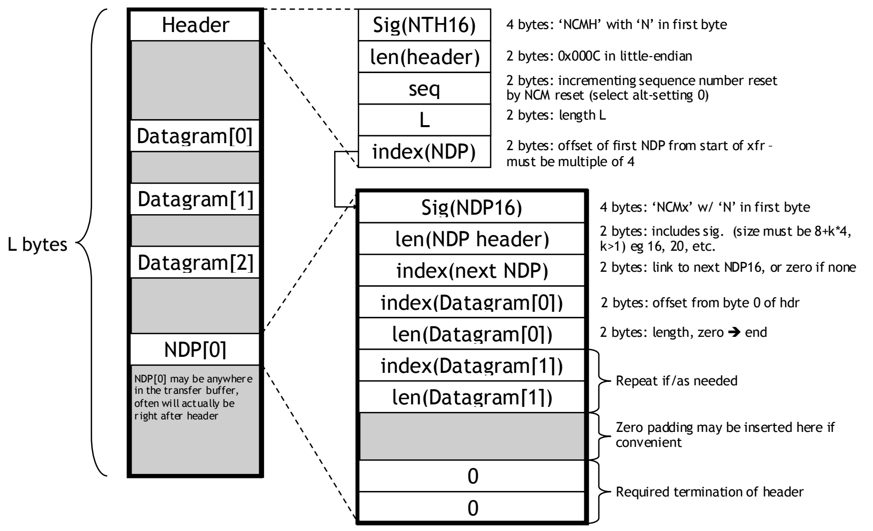
L bytes
Header
Datagram[0]
Datagram[1]
Datagram[2]
NDP[0]
NDP[0] may be anywhere in the transfer buffer, often will actually be right after header
Sig(NTH16)
4 bytes: 'NCMH' with 'N' in first byte
len(header)
2 bytes: 0x000C in little-endian
seq
2 bytes: incrementing sequence number reset by NCM reset (select alt-setting 0)
L
2 bytes: length L
index(NDP)
2 bytes: offset of first NDP from start of xfr – must be multiple of 4
Sig(NDP16)
4 bytes: 'NCMx' w/ 'N' in first byte
len(NDP header)
2 bytes: includes sig. (size must be 8+k*4, k>1) eg 16, 20, etc.
index(next NDP)
2 bytes: link to next NDP16, or zero if none
index(Datagram[0])
2 bytes: offset from byte 0 of hdr
len(Datagram[0])
2 bytes: length, zero → end
index(Datagram[1])
len(Datagram[1])
Repeat if/as needed
Zero padding may be inserted here if convenient
0
0
Required termination of header

Figure 3-1 - NTB layout details (16 bit)

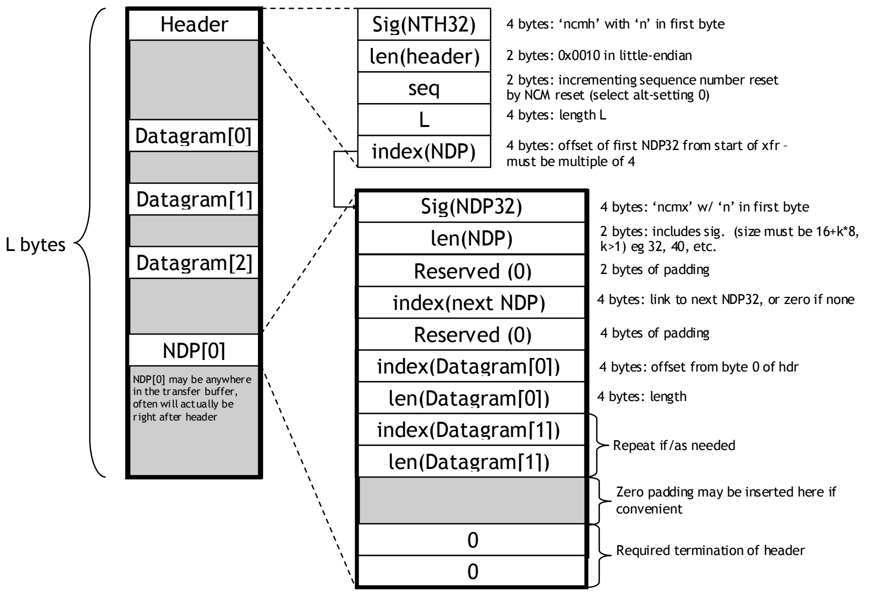
L bytes
Header
Datagram[0]
Datagram[1]
Datagram[2]
NDP[0]
NDP[0] may be anywhere in the transfer buffer, often will actually be right after header
Sig(NTH32)
4 bytes: 'ncmh' with 'n' in first byte
len(header)
2 bytes: 0x0010 in little-endian
seq
2 bytes: incrementing sequence number reset by NCM reset (select alt-setting 0)
L
4 bytes: length L
index(NDP)
4 bytes: offset of first NDP32 from start of xfr – must be multiple of 4
Sig(NDP32)
4 bytes: 'ncmx' w/ 'n' in first byte
len(NDP)
2 bytes: includes sig. (size must be 16+k*8, k>1) eg 32, 40, etc.
Reserved (0)
2 bytes of padding
index(next NDP)
4 bytes: link to next NDP32, or zero if none
Reserved (0)
4 bytes of padding
index(Datagram[0])
4 bytes: offset from byte 0 of hdr
len(Datagram[0])
4 bytes: length
index(Datagram[1])
len(Datagram[1])
Repeat if/as needed
Zero padding may be inserted here if convenient
0
0
Required termination of header

Figure 3-2 - NTB layout details (32 bit)

3.2 NCM Transfer Headers

3.2.1 NTH for 16-bit NTB (NTH16)

A 16-bit NTB must begin with an NTH16 structure, described in Table 3-1.

Table 3-1: Sixteen Bit NCM Transfer Header (NTH16)

Offset	Field	Size	Value	Description
0	dwSignature	4	Number (0x484D434E)	Signature of the NTH16 Header. This is transmitted in little-endian form, i.e., as 0x4E, 0x43, 0x4D, 0x48, or as the character sequence "NCMH"
4	wHeaderLength	2	Number (0x000C)	Size in bytes of this NTH16 structure, in little-endian format.
6	wSequence	2	Number	Sequence number. The transmitter of a block shall set this to zero in the first NTB transferred after every "function reset" event, and shall increment for every NTB subsequently transferred. The effect of an out-of-sequence block on the receiver is not specified. The specification allows the receiver to decide whether to check the sequence number, and to decide how to respond if it's incorrect. The sequence number is primarily supplied for debugging purposes.
8	wBlockLength	2	Number	Size of this NTB in bytes ("L" in Figure 3-1). Represented in little-endian form. NTB size (IN/OUT) shall not exceed *dwNtbInMaxSize* or *dwNtbOutMaxSize* respectively; see Table 6-3 in 6.2.1. If *wBlockLength* = 0x0000, the block is terminated by a short packet. In this case, the USB transfer must still be shorter than *dwNtbInMaxSize* or *dwNtbOutMaxSize*. If exactly *dwNtbInMaxSize* or *dwNtbOutMaxSize* bytes are sent, and the size is a multiple of *wMaxPacketSize* for the given pipe, then no ZLP shall be sent. *wBlockLength*= 0x0000 must be used with extreme care, because of the possibility that the host and device may get out of sync, and because of test issues. *wBlockLength* = 0x0000 allows the sender to reduce latency by starting to send a very large NTB, and then shortening it when the sender discovers that there's not sufficient data to justify sending a large NTB.
10	wNdpIndex	2	Number	Offset, in little endian, of the first NDP16 from byte zero of the NTB. This value must be a multiple of 4, and must be >= 0x000C.

Deleted: wFpIndex

3.2.2 NTH for 32-bit NTB (NTH32)

The 32-bit form of the NTH is described in Table 3-2.

Table 3-2: Thirty-two bit NCM Transfer Header (NTH32)

Offset	Field	Size	Value	Description
0	dwSignature	4	Number (0x686D636E)	Signature of the NTH32 Header. This is transmitted in little-endian form, i.e., as 0x6E, 0x63, 0x6D, 0x68, or as the character sequence "ncmh"
4	wHeaderLength	2	Number (0x0010)	Size in bytes of this NTH32 structure, in little-endian format.
6	wSequence	2	Number	Sequence number. The transmitter of a block shall set this to zero in the first NTB transferred after every "function reset" event, and shall increment for every NTB subsequently transferred. The effect of an out-of-sequence block on the receiver is not specified. The specification allows the receiver to decide whether to check the sequence number, and to decide how to respond if it's incorrect. The sequence number is primarily supplied for debugging purposes.
8	dwBlockLength	4	Number	Size of this NTB in bytes ("L" in Figure 3-2). Represented in little-endian form. NTB size (IN/OUT) shall not exceed *dwNtbInMaxSize* or *dwNtbOutMaxSize* respectively; see Table 6-3 in 6.2.1. If *dwBlockLength* = 0x0000, the block is terminated by a short packet. In this case, the USB transfer must still be shorter than *dwNtbInMaxSize* or *dwNtbOutMaxSize*. If exactly *dwNtbInMaxSize* or *dwNtbOutMaxSize* bytes are sent, and the size is a multiple of *wMaxPacketSize* for the given pipe, then no ZLP shall be sent. *dwBlockLength* = 0x0000 must be used with extreme care, because of the possibility that the host and device may get out of sync, and because of test issues. *dwBlockLength* = 0x0000 allows the sender to reduce latency by starting to send a very large NTB, and then shortening it when the sender discovers that there's not sufficient data to justify sending a large NTB.
12	dwNdpIndex	4	Number	Offset, in little endian, of the first NDP32 from byte zero of the NTB. This value must be a multiple of 4, and must be >= 0x0010 (because the first NDP32 had to be after the end of the NTH32).

Deleted: dwFpIndex

3.3 NCM Datagram Pointers (NDPs)

NCM Datagram Pointers (NDPs) describe the Ethernet datagrams that are embedded in an NDP. As with NTH structures, two forms are defined. One form (the NDP16) is used for 16-bit NTBs; a second form (the NDP32) is used for 32-bit NTBs. These forms are architecturally equivalent, but differ in that many fields are 16-bits wide in the NDP16, but are 32-bits in the NDP32.

3.3.1 NDP for 16-bit NTBs (NDP16)

The layout of the NDP16 is given in Table 3-3. It has the following overall structure:

- 8 bytes of header information
- 1 or more NCM Datagram Pointer Entries (4 bytes per entry)
- A terminating zero NCM Datagram Pointer Entry (4 bytes).

Table 3-3: Sixteen-bit NCM Datagram Pointer Table (NDP16)

Offset	Field	Size	Value	Description
0	dwSignature	4	Number (0x304D434E, 0x314D434E)	Signature of this NDP16. This is transmitted in little-endian form, i.e., as 0x4E, 0x43, 0x4D, 0x30 or 0x4E, 0x43, 0x4D, 0x31. or as the character sequences “NCM*0*”,or “NCM1 where “0” or ‘1” has the meaning given in Table 3-5.
4	wLength	2	Number	Size of this NDP16, in little-endian format. This must be a multiple of 4, and must be at least 16 (0x0010).
6	wNextNdpIndex	2	Reserved (0)	Reserved for use as a link to the next NDP16 in the NTB
8	wDatagramIndex[0]	2	Number	Byte index, in little endian, of the first datagram described by this NDP16. The index is from byte zero of the NTB. This value must be >= the value stored in *wHeaderLength* of the NTH16 (because it must point past the NTH16).
10	wDatagramLength[0]	2	Number	Byte length, in little endian, of the first datagram described by this NDP16. For Ethernet frames, this value must be >= 14.
12	wDatagramIndex[1]	2	Number	Byte index, in little endian, of the second datagram described by this NDP16. If zero, then this marks the end of the sequence of datagrams in this NDP16.
14	wDatagramLength[1]	2	Number	Byte length, in little endian, of the second datagram described by this NDP16. If zero, then this marks the end of the sequence of datagrams in this NDP16.
…				
wLength-4	wDatagramIndex[(wLength-8) / 4 - 1]	2	Number (0)	Always zero
wLength – 2	wDatagramLength[(wLength-8) / 4 - 1]	2	Number(0)	Always zero

Deleted: wNextFpIndex

Deleted: must be

3.3.2 NDP for 32-bit NTBs (NDP32)

The layout of the NDP32 is given in the Table 3-4. It has the following overall structure:

- 16 bytes of header information
- 1 or more NCM Datagram Pointer Entries (8 bytes per entry)
- A terminating zero NCM Datagram Pointer Entry (8 bytes).

Table 3-4: Thirty-two bit NCM Datagram Pointer Table (NDP32)

Offset	Field	Size	Value	Description
0	dwSignature	4	Number (0x306D636E, 0x316D636E)	Signature of this NDP32. This is transmitted in little-endian form, i.e., as 0x6E, 0x63, 0x6D, 0x30 or 0x6E, 0x63, 0x6D, 0x31. These are equivalent to the character sequences "ncm0" and "ncm1", where "0" and "1" have the meaning given in Table 3-5.
4	wLength	2	Number	Size of this NDP32, in little-endian format. This must be a multiple of 8, and must be at least 32 (0x0020).
6	wReserved6	2	Reserved (0)	Reserved for future use.
8	dwNextNdpIndex	4	Number	Reserved for use as a link to the next NDP32 in the NTB
12	dwReserved12	4	Reserved (0)	Reserved for future use.
16	dwDatagramIndex[0]	4	Number	Byte index, in little endian, of the first datagram described by this NDP32. The index is from byte zero of the NTB. This value must be >= 0x0010 (because it must point past the NTH).
20	dwDatagramLength[0]	4	Number	Byte length, in little endian, of the first datagram described by this NDP32. For Ethernet frames, this value must be >= 14.
24	dwDatagramIndex[1]	4	Number	Byte index, in little endian, of the second datagram described by this NDP32. If zero, then this marks the end of the sequence of datagrams in this NDP32.
28	dwDatagramLength[1]	4	Number	Byte length, in little endian, of the second datagram described by this NDP32. If zero, then this marks the end of the sequence of datagrams in this NDP32.
		…		
wLength – 8	dwDatagramIndex[(wLength-8) / 8 - 1]	4	Number (0)	Always zero
wLength – 4	dwDatagramLength[(wLength-8) / 8 - 1]	4	Number(0)	Always zero

Deleted: must be

Deleted: NDP16

3.3.3 Datagram Formatting

All the datagrams described by a given NDP share a common format. The datagram always starts with a 14-byte [IEEE802.3] header, and then continues with the appropriate payload. The preparer of an NDP16 or NDP32 must choose whether a CRC-32 will be appended to the payload. If a CRC-32 is appended, and the header and payload combined are less than the minimum specified in 802.3, the datagram must be padded appropriately before calculating and appending the CRC-32.

Table 3-5: NDP Datagram Formatting Codes

Value	Datagram Formatting
0x30 ('0')	[IEEE802.3], no CRC-32
0x31 ('1')	[IEEE802.3], with CRC-32. If CRC-32 is appended, transmitter is responsible for padding the datagram to the appropriate minimum length for 802.3 prior to calculating and appending CRC-32.

3.3.4 NCM Ethernet Frame Alignment

It is well known that many network stacks in embedded devices benefit through careful alignment of the payload to system defined memory boundaries. NCM allows a function to align transmitted datagrams on any convenient boundary within the NTB. Functions indicate how they intend to align their transmitted datagrams to the host in the NTB Parameter Structure (Table 6-3)

Similarly, for data transmitted from the host, functions indicate their preferred alignment requirements to the host. The host then formats the NTBs to satisfy this constraint.

NCM assumes that hosts are more flexible and powerful than devices. Therefore, the host shall always honor the constraints given by the device when preparing OUT NTBs.

Alignment requirements are met by controlling the location of the payload (the data following the Ethernet header in each datagram). This alignment is specified by indicating a constraint as a divisor and a remainder. The agent formatting a given NTB aligns the payload of each datagram by inserting padding, such that the offset of each datagram satisfies the constraint:

Offset % *wNdpInDivisor* == *wNdpInPayloadRemainder* (for IN datagrams)

Or

Offset % *wNdpOutDivisor* == *wNdpOutPayloadRemainder* (for OUT datagrams)

Two use cases are anticipated (although the specification doesn't restrict the user to these two cases).

In one use case, the function wants to align the beginning of an IP packet to a cache line boundary. Cache lines are generally much smaller than the maximum Ethernet frame size. In this case, the *wNdpXxDivisor* is set to the size of a cache line (in bytes), and the *wNdpXxPayloadRemainder* is set to zero. The effect is shown schematically in Figure 3-3.

In another use case, the function wants to place each IP packet in a fixed sized buffer. (This is primarily intended for use on the OUT pipe.) For this to work, each buffer must be bigger than the maximum Ethernet frame size. In this case, *wNdpXxDivisor* is set to the size of the buffer, and *wNdpXxPayloadRemainder* is set to the desired offset of the IP packet in the fixed size buffer. This situation is shown schematically in Figure 3-4.

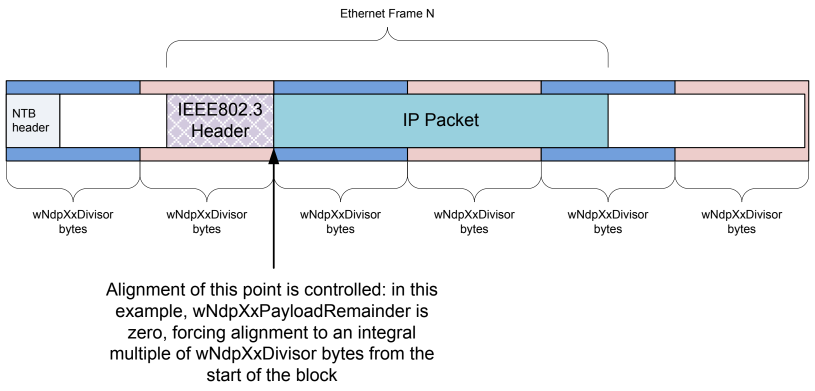
Ethernet Frame N
NTB header
IEEE802.3 Header
IP Packet
wNdpXxDivisor bytes
wNdpXxDivisor bytes
wNdpXxDivisor bytes
wNdpXxDivisor bytes
wNdpXxDivisor bytes
wNdpXxDivisor bytes
Alignment of this point is controlled: in this example, wNdpXxPayloadRemainder is zero, forcing alignment to an integral multiple of wNdpXxDivisor bytes from the start of the block

Figure 3-3 - Alignment to a cache line

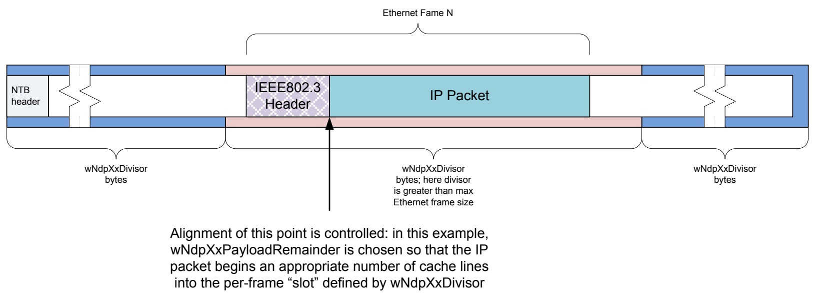
Ethernet Fame N
NTB header
IEEE802.3 Header
IP Packet
wNdpXxDivisor bytes
wNdpXxDivisor bytes; here divisor is greater than max Ethernet frame size
wNdpXxDivisor bytes
Alignment of this point is controlled: in this example, wNdpXxPayloadRemainder is chosen so that the IP packet begins an appropriate number of cache lines into the per-frame "slot" defined by wNdpXxDivisor

Figure 3-4 - Alignment for fixed-size internal buffers

3.4 NTB Maximum Sizes

NCM functions determine the maximum permitted size of NTB that they can process. The upper limit is usually determined by the buffering capacity of the attached function, but there may be other factors involved as well, such as latency.

For OUT pipes, the host determines the actual size of the NTB data structures sent to the device. Hosts can discover the maximum size supported by the device from the NTB Parameter Structure, and shall not send NTBs larger than the device can support.

Deleted: NTP

For IN pipes, the host tells the device the size of NTB data structures that it wishes the device to send using the *SetNtbInputSize* command. Hosts can discover the maximum size supported by the device from the field *dwNtbInMaxSize* in the NTB Parameter Structure (Table 6-3). Devices shall not send NTBs larger than the host has requested. The host shall not select a maximum NTB size that is not supported by the device.

Deleted: NTP

3.5 NTB format support

Functions conforming to this specification shall support 16-bit NTB structures (Table 3-1 and Table 3-3). Functions may also support 32-bit NTB structures (Table 3-2 and Table 3-4).

3.6 Ethernet frame Datagram Maximum Size

The maximum size of an Ethernet frame datagram can be dynamically adjusted by the host using the *SetMaxDatagramSize* request. Commonly, Ethernet frames are 1514 bytes or less in length (not including the CRC), but for many applications a larger maximum frame size is needed (e.g., 802.1Q VLAN tagging, jumbo frames). Hosts can discover the maximum Ethernet frame size supported by a device from the value *wMaxSegmentSize* in the Ethernet Networking Functional Descriptor, and shall not select a size larger than the device can support, nor shall it send a frame larger than the device can support.

Host or function may append CRCs to datagrams. These four-byte CRCs are *not* included when determining the "maximum segment size", but they are counted when specifying the datagram size in the NDP. For example, if the maximum datagram size is currently 1514, and the NDP header indicates that CRCs are being appended, then the maximum *wDatagramLength* value for any datagram in that NDP is 1518.

3.7 Null NCM Datagram Pointer Entries

Any NCM Datagram pointer entry with an index field (*wDatagramIndex, dwDatagramIndex*) of zero or with a length field (*wDatagramLength, dwDatagramLength*) of zero, or with both index and length fields set to zero, shall be treated as a Null entry. Receivers shall process datagram pointer entries sequentially from the first entry in the NTB. The first Null entry shall be interpreted as meaning that all following NCM Datagram Pointer Entries in the NDP are to be ignored.

Deleted: offset

Deleted: offset

The rules given in sections 3.3.1 and 3.3.2 specify that every NDP shall end with a Null entry. It is an error for a transmitter to format an NDP without a terminating Null entry. Receivers MAY discard such NDPs (and all associated frames) entirely.

Transmitters are allowed to send a properly-formatted NTB containing an NDP whose datagram pointer entries are all zero. Receivers shall ignore such NTBs.

It is an error for a transmitter to send an NDP with non-Null NCM Datagram Pointer Entries following the first Null. Receivers MAY process datagrams up to the first Null NCM Datagram Pointer Entry, and MAY ignore the remaining non-Null entries in the NDP.

4 Class-Specific Codes

This section lists the codes for the Communications Class and Data Class, specifically subclasses and protocols. These values are used in the *bInterfaceSubClass* and *bInterfaceProtocol* fields of the standard device descriptors as defined in chapter 9 of [USB30]. Values for *bDeviceClass*, *bDeviceSubClass*, and *bDeviceProtocol* in the Device Descriptor are defined in [USBCDC12].

4.1 NCM Communications Interface Subclass Code

Table 4-1 defines the interface subclass code used in the NCM Communications Interface descriptor.

Table 4-1 NCM Communications Interface Subclass Code

Code	Subclass
0Dh	Network Control Model

4.2 NCM Communications Interface Protocol Code

Table 4-2 lists the Protocol code used in the NCM Communications Interface Descriptor. This code indicates the format of commands sent, and responses received, via *SendEncapsulatedCommand* and *GetEncapsulatedResponse*. If *SendEncapsulatedCommand* and *GetEncapsulatedResponse* are not supported (as stated by the value in bit D2 of *bmNetworkCapabilities* in the NCM Functional Descriptor, Table 5-2), then *bInterface-Procotol* shall be set to zero.

Regardless of the format of encapsulated commands and responses, all NCM functions shall implement the default pipes requests and notifications as specified by Table 6-1 and Table 6-4.

Table 4-2 NCM Communications Interface Protocol Code

Code	Protocol	Defined By	Encapsulated Command/Response Payload Format
00h	None	[USBCDC12]	No encapsulated commands / responses.
01h	N/A	[USBCDC12]	Do not use.
02h	N/A	[USBCDC12]	Do not use
03h	N/A	[USBCDC12]	Do not use
04h	N/A	[USBCDC12]	Do not use
05h	N/A	[USBCDC12]	Do not use
06H	N/A	[USBCDC12]	Do not use
06h-FDh	Reserved		Reserved for future use
FEh	OEM defined	[USBWMC11]	External Protocol: Commands defined by Command Set Functional Descriptor following the NCM Communications Interface Descriptor.
FFh	N/A	[USBCDC12]	Do not use

4.3 NCM Data Class Interface Protocol Codes

[USBCDC12] defines Data Class Protocols. Alternate settings 0 and 1 of an NCM function's Data Interface shall use the protocol code defined in Table 4-3.

Table 4-3 NCM Data Class Protocol Code

Code	Protocol
01h	Network Transfer Block (3.3.3)

4.4 NCM Functional Descriptor Code

Table 4-4: NCM Functional Descriptor Code

Code	Name	Descriptor
1Ah	NCM_FUNC_DESC_CODE	NCM Functional Descriptor

5 Descriptors

5.1 Standard USB Descriptor Definitions

Refer to [USBCDC12].

5.2 NCM Communications Interface Descriptor Requirements

NCM functions must implement class-specific descriptors ("functional descriptors") for the NCM Communications Interface. The framework for these is defined in [USBCDC12].

NCM Communications Interfaces must implement the following functional descriptors described in [USBCDC12]:

- Header Functional Descriptor (describing the level of compliance to [USBCDC12])
- Union Functional Descriptor (containing the interface numbers of the Communications Interface and the Data interface).

NCM functions must implement an Ethernet Network Functional Descriptor, as defined in [USBECM12]

NCM functions must implement a NCM Functional Descriptor, as described in section 5.2.1.

If *bInterfaceProtocol* is FEh, then the NCM function must provide a Command Set Functional Descriptor, and may provide additional Command Set Detail Functional Descriptors.

The class-specific descriptors must be followed by an Interrupt IN endpoint descriptor.

Descriptor requirements are summarized in Table 5-1.

Table 5-1: NCM Communication Interface Descriptor Requirements

Descriptor	Description	Req'd/Opt	Order	Reference
HEADER	CDC Header functional descriptor	Required	First	[USBCDC12], 5.2.3.1
UNION	CDC Union functional descriptor	Required	Arbitrary	[USBCDC12] 5.2.3.2
ETHERNET	CDC Ethernet Networking Functional Descriptor	Required	Arbitrary	[USBECM12], 5.4
NCM	NCM Functional Descriptor	Required	Arbitrary	5.2.1
COMMAND SET	Command Set Functional Descriptor	Required if NCM Communications Interface bInterfa-ceProtocol is 0FEh	Arbitrary	5.2.2
COMMAND SET DETAIL	Command Set Detail Functional Descriptor	Optional if Com-mand Set Func-tional Descriptor is present; other-wise prohibited	After Command Set Func-tional De-scriptor	5.2.3

5.2.1 NCM Functional Descriptor

This descriptor provides information about the implementation of the NCM function. It is mandatory, and must appear after the Header Functional Descriptor.

Table 5-2: NCM Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	6	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE (0x24)
2	bDescriptorSubtype	1	Constant (1Ah)	NCM Functional Descriptor subtype, as defined in Table 4-4
3	bcdNcmVersion	2	Number 0x0100	Release number of this specification in BCD, with implied decimal point between bits 7 and 8. 0x0100 == 1.00 == 1.0. This is a little-endian constant, so the bytes will be 0x00, 0x01.
5	bmNetworkCapabilities	1	Bitmap	Specifies the capabilities of this function. A bit value of zero indicates that the capability is not supported. D7..D6: Reserved (zero) D5: Function can process 8-byte forms of *GetNtbInputSize* and *SetNtbInputSize* requests D4: Function can process *SetCrcMode* and *GetCrcMode* requests D3: Function can process *SetMaxDatagramSize* and *GetMaxDatagramSize* requests. D2: Function can process *SendEncapsulatedCommand* and *GetEncapsulatedResponse* requests. D1: Function can process *GetNetAddress* and *SetNetAddress* requests. D0: Function can process *SetEthenetPacketFilter* requests, as defined in [USBECM12]. If not set, broadcast, directed and multicast packets are always passed to the host.

Deleted: 0x22

Deleted: D5

5.2.2 Command Set Functional Descriptor

If the NCM Communications Interface has *bInterfaceProtocol* set to "External Protocol", as given in Table 4-2, then the command set transported by *SendEncapsulatedCommand* and *GetEncapsulatedResponse* is governed by a specification external to this document. The specification is identified by a GUID given in a Command Set Functional descriptor, which must appear associated with the NCM Communications Interface descriptor. This descriptor is defined in [USBWMC11], section 8.1.2.2. The GUID is defined by the appropriate external specification. The GUID identifies the format and contents of the command set. The command set may be, but is not required to be, AT commands and responses. This descriptor is required if *bInterfaceProtocol* is set to "External Protocol".

If the NCM Communications Interface has *bInterfaceProtocol* set to any other value, then the Command Set Functional Descriptor shall not appear, and the host shall ignore any such descriptors.

5.2.3 Command Set Detail Functional Descriptor

If a Command Set Functional Descriptor appears, it may be followed by one or more Command Set Detail Functional Descriptors, as described in [USBWMC11], section 8.1.2.3. If the Command Set Functional Descriptor is not present, Command Set Detail Functional Descriptors shall not appear, and the host shall ignore any such descriptors.

5.3 Data Interface Descriptor Requirements

The Data Interface of an NCM networking function shall have two alternate settings. The first alternate setting (the default interface setting, alternate setting 0) shall include no endpoints and therefore no networking traffic can be exchanged when the default interface setting is selected. The second alternate setting (alternate setting 1) is used for normal operation, and shall include one bulk IN endpoint and one bulk OUT endpoint.

The interface descriptors for alternate settings 0 and 1 shall have *bInterfaceSubClass* set to 0, and *bInterfaceProtocol* set to 01h (see section 4.3).

6 Communications Class Specific Messages

6.1 Overview

A Communications Interface shall support the standard requests defined in chapter 9 of [USB30]. In addition, an NCM Communications Interface shall support class- and subclass-specific requests and notifications. These are used to manage the function.

It is an error to send requests to an NCM Data Interface. Device behavior in this case is not specified.

6.2 Network Control Model Requests

Table 6-1 lists all of the class-specific requests that are valid for an NCM Communications Interface. This table includes those defined in [USBECM12]. Commands marked as "required" must be implemented by any conforming NCM function. Commands marked as "optional" must be implemented in some circumstances; see the Reference section for each command for details.

Table 6-1: Networking Control Model Requests

Request	Description	Req'd/Opt	reference
SendEncapsulatedCommand	Issues a command in the format of the supported control protocol. The intent of this mechanism is to support networking functions (e.g. host-based cable modems) that require an additional vendor-defined interface for media specific hardware configuration and management.	Optional	[USBCDC12]
GetEncapsulatedResponse	Requests a response in the format of the supported control protocol.	Optional	[USBCDC12]
SetEthernetMulticastFilters	Controls the receipt of Ethernet frames that are received with "multicast" destination addresses.	Optional	[USBECM12]
SetEthernetPowerManagementPatternFilter	Some hosts are able to conserve energy and stay quiet in a "sleeping" state while not being used. NCM functions may provide special pattern filtering hardware that enables the function to wake up the attached host on demand when something is attempting to contact the host (e.g. an incoming web browser connection). This command allows the host to specify the filter values that detect these special frames.	Optional	[USBECM12]
GetEthernetPowerManagementPatternFilter	Retrieves the status of the above power management pattern filter setting	Optional	[USBECM12]
SetEthernetPacketFilter	Controls the types of Ethernet frames that are to be received via the function.	Optional	[USBECM12]
GetEthernetStatistic	Retrieves Ethernet statistics such as frames transmitted, frames received, and bad frames received.	Optional	[USBECM12]
GetNtbParameters	Requests the function to report parameters that characterize the Network Control Block	Required	6.2.1
GetNetAddress	Requests the current EUI-48 network address	Optional	6.2.2
SetNetAddress	Changes the current EUI-48 network address	Optional	6.2.3
GetNtbFormat	Get current NTB Format	Optional	6.2.4
SetNtbFormat	Select 16 or 32 bit Network Transfer Blocks	Optional	6.2.5
GetNtbInputSize	Get the current value of maximum NTB input size	Required	6.2.6
SetNtbInputSize	Selects the maximum size of NTBs to be transmitted by the function over the bulk IN pipe.	Required	6.2.7

Deleted: 6.3.1

Deleted: 0

Deleted: 6.2.4

Deleted: 6.2.6

Request	Description	Req'd/Opt	reference
GetMaxDatagramSize	Requests the current maximum datagram size	Optional	6.2.8
SetMaxDatagramSize	Sets the maximum datagram size to a value other than the default	Optional	6.2.9
GetCrcMode	Requests the current CRC mode	Optional	6.2.10
SetCrcMode	Sets the current CRC mode	Optional	6.2.11

Table 6-2 describes the requests that are use in the Networking Control Model Subclass, including those defined in [USBECM12].

Table 6-2: Class-Specific Request Codes for Network Control Model 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
GET_NTB_PARAMETERS	80h
GET_NET_ADDRESS	81h
SET_NET_ADDRESS	82h
GET_NTB_FORMAT	83h
SET_NTB_FORMAT	84h
GET_NTB_INPUT_SIZE	85h
SET_ NTB_ INPUT_SIZE	86h
GET_MAX_DATAGRAM_SIZE	87h
SET_MAX_DATAGRAM_SIZE	88h
GET_CRC_MODE	89h
SET_CRC_MODE	8Ah
RESERVED (future use)	8Bh-8Fh

6.2.1 GetNtbParameters

This request retrieves the parameters that describe NTBs for each direction. In response to this request, the function shall return these elements as listed in Table 6-3.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	GET_NTB_PARAM-ETERS	zero	NCM Com-munications Interface	Number of bytes to read	NTB Parameter Struc-ture (Table 6-3)

The returned NTB Parameter Structure is defined in Table 6-3.

Table 6-3: NTB Parameter Structure

Offset	Field	Size	Value	Description
0	wLength	2	Number	Size of this structure, in bytes = 1Ch.
2	bmNtbFormatsSupported	2	Bitmap	Bit 0: 16-bit NTB supported (set to 1) Bit 1: 32-bit NTB supported Bits 2 – 15: reserved (reset to zero; must be ignored by host)
4	dwNtbInMaxSize	4	Number	IN NTB Maximum Size in bytes
8	wNdpInDivisor	2	Number	Divisor used for IN NTB Datagram payload alignment
10	wNdpInPayloadRemainder	2	Number	Remainder used to align input datagram payl-oad within the NTB: (Payload Offset) mod (*wNdpInDivisor*) = *wNdpInPayloadRemainder*
12	wNdpInAlignment	2	Number	NDP alignment modulus for NTBs on the IN pipe. Shall be a power of 2, and shall be at least 4.
14	-reserved	2	Zero	Padding, shall be transmitted as zero by func-tion, and ignored by host.
16	dwNtbOutMaxSize	4	Number	OUT NTB Maximum Size
20	wNdpOutDivisor	2	Number	OUT NTB Datagram alignment modulus
22	wNdpOutPayloadRemainder	2	Number	Remainder used to align output datagram payload offsets within the NTB: (Payload Offset) mod (*wNdpOutDivisor*) = *wNdpOutPayloadRemainder*
24	wNdpOutAlignment	2	Number	NDP alignment modulus for use in NTBs on the OUT pipe. Shall be a power of 2, and shall be at least 4.
26	wNtbOutMaxDatagrams	2	Number	Maximum number of datagrams that the host may pack into a single OUT NTB. Zero means that the device imposes no limit.

Deleted: -reserved

Deleted: Padding. Shall be transmitted as zero by function and ignored by host.

To get the full response, the host should set *wLength* to at least 1Ch. The function shall never return more than 1Ch bytes in response to this command.

All NCM functions shall support 16-bit NTBs. Therefore, bit 0 of *bmNtbFormatsSupported* shall always be set to 1. NCM functions may support 32-bit NTBs. If 32-bit NTBs are supported, then GetNtbFormat and SetNtbFormat must be supported.

6.2.2 GetNetAddress

This request returns the function's current EUI-48 station address.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_NET_ADDRESS	Zero	NCM Communi-cations Interface	Number of bytes to read	The EUI-48 current address, in net-work byte order

To get the entire network address, the host should set *wLength* to at least 6. The function shall never re-turn more than 6 bytes in response to this command.

6.2.3 SetNetAddress

This request sets the function's current EUI-48 station address. It does not change the function's perma-nent EUI-48 station address, which is given by field *iMACAddress* in the Ethernet Functional Descriptor (see [USBECM12], section 5.4).

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_NET_ADDRESS	Zero	NCM Communi-cations Interface	6	The EUI-48 address, in network byte order

The host shall set *wLength* to 6. The function shall return an error response (a STALL PID) if wLength is set to any other value.

The function resets its EUI-48 station address to the permanent address, as governed by events outside the scope of this command. See section 7.1 for details.

The host shall only send this command while the NCM Data Interface is in alternate setting 0.

6.2.4 GetNtbFormat

This request returns the NTB data format currently being used by the function.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_NTB_FORMAT	Zero	NCM Communi-cations Interface	Number of bytes to read	The NTB format code (2 bytes, little-endian), as defined under *wValue* in *SetNtbFormat* (6.2.5).

To get the full response, the host should set *wLength* to at least 2. The function shall never return more than 2 bytes in response to this command.

This command must be supported by the function if it declares support for an NTB size other than 16bit in bmNtbFormatsSupported.

If the function does not support NTB sizes other than 16bit, then the host must not issue this command to the function.

6.2.5 SetNtbFormat

This request selects the format of NTB to be used for NTBs transmitted from the function to the host. The host must choose one of the available choices from the *bmNtbFormatsSupported* bitmap element from the *GetNtbParameters* command response (Table 6-3).

The command format uses the same format, with a single choice selected.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_NTB_FORMAT	NTB Format Selection: 0000h: NTB-16 0001h: NTB-32 All other values are reserved.	NCM Communications Interface	0	None

The host shall only send this command while the NCM Data Interface is in alternate setting 0.

The function's NTB format setting may be changed by events beyond the scope of this command; see section 7.1 for details.

If the value passed in wValue is not supported, the function shall return an error response (a STALL PID) and shall not change the format it is using to send and receive NTBs.

This command must be supported by the function if it declares support for an NTB size other than 16bit in bmNtbFormatsSupported.

If the function does not support NTB sizes other than 16bit, then the host must not issue this command to the function.

6.2.6 GetNtbInputSize

This request returns NTB input size currently being used by the function.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_NTB_INPUT_SIZE	Zero	NCM Communications Interface	Number of bytes to read	The NTB input size structure, as defined in *SetNtbInputSize* (6.2.7).

To get the NTB input size, the host should set *wLength* to at least 4. To get the full NTB input size structure, the host should set *wLength* to at least 8. If bit D5 is set in field *bmNetworkCapabilities* of the function's NCM Functional Descriptor, the function shall never return more than 8 bytes in response to this command. If bit D5 is reset, the function shall never return more than 4 bytes in response to this command. The fields in the input size structure are returned in little-endian order

Deleted: full

Deleted: response

Deleted: The

Deleted: is

6.2.7 SetNtbInputSize

This request selects the maximum size of NTB that the device is permitted to send to the host, and optionally the maximum number of datagrams that the device is permitted to encode into a single NTB.

Deleted: .

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_NTB_INPUT_SIZE	Zero	NCM Commu-nications Interface	4 or 8	If *wLength* is 8, then this is the NTB Input Size Structure. If *wLength* is 4, then this is the *dwNtbInMaxSize* field of the NTB Input Size Structure.

Deleted: Maximum NTB size, in bytes, in little-endian order

Table 6-4. NTB Input Size Structure

Offset	Field	Size	Value	Description
0	dwNtbInMaxSize	4	Number	IN NTB Maximum size in bytes. The host shall select a size that is at least 2048, and no larger than the maximum size permitted by the function, according to the value given in the NTB Parameter Structure
4	wNtbInMaxDatagrams	2	Number	Maximum number of datagrams within the IN NTB. Zero means no limit.
6	reserved	2	Number	Shall be transmitted as zero and ignored upon receipt.

If bit D5 is set in the *bmNetworkCapabilities* field of function's NCM Functional Descriptor, the host may set *wLength* either to 4 or to 8. If *wLength* is 4, the function shall assume that *wNtbInMaxDatagrams* is to be set to zero. If *wLength* is 8, then the function shall use the provided value as the limit. The function shall return an error response (a STALL PID) if *wLength* is set to any other value.

If bit D5 is reset in the *bmNetworkCapabilities* field of the function's NCM Functional Descriptor, the host shall set *wLength* to 4. The function shall return an error response (a STALL PID) if *wLength* is set to any other value.

Deleted: The

If the value passed in the data phase is not valid, or if *wLength* is not valid, the function shall return an error response (a STALL PID) and shall not change the value it uses for preparing NTBs.

6.2.8 GetMaxDatagramSize

This request returns the currently effective maximum datagram size that the function has in effect.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_MAX_DATAGRAM_-SIZE	Zero	NCM Communi-cations Interface	Number of bytes to read	The current maximum datagram size, in little endian order (2 bytes).

To get the full response, the host should set *wLength* to at least 2. The function shall never return more than 2 bytes in response to this command.

6.2.9 SetMaxDatagramSize

This request selects the maximum datagram size that either host or function will send in an NTB.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_MAX_DATAGRAM_-SIZE	Zero	NCM Commu-nications	2	Maximum datagram size, in bytes, in little-endian order

			Interface		

The host shall select a size that is at least 1514, and no larger than the maximum size permitted by the function, according to the value given by *wMaxSegmentSize* in the Ethernet Networking Functional Descriptor.

The host shall set *wLength* to 2. The function shall return an error response (a STALL PID) if *wLength* is set to any other value. If the value passed in the data phase is not valid, the function shall return an error response (a STALL PID) and shall not change the value it uses as current maximum datagram size.

The function's maximum datagram size is set to a default value by events outside the scope of this command; see section 7.2 for details.

Deleted: 7.1

6.2.10 GetCrcMode

This request returns the currently selected CRC mode for NTBs formatted by the function.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_CRC_MODE	Zero	NCM Communications Interface	Number of bytes to read	The current CRC mode, in little endian order (2 bytes).

To get the full response, the host should set *wLength* to at least 2. The function shall never return more than 2 bytes in response to this command.

Two values are possible. The function shall return 0000h if CRCs are not being appended to datagrams. The function shall return 0001h if CRCs are being appended to datagrams.

6.2.11 SetCrcMode

This request controls whether the function will append CRCs to datagrams when formatting NTBs to be sent to the host.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CRC_MODE	CRC mode: 0000h: CRCs shall not be appended 0001h: CRCs shall be appended All other values are reserved.	NCM Communications Interface	0	None

If the value passed in *wValue* is not valid, the function shall return an error response (a STALL PID) and shall not change the CRC mode.

The function's CRC mode is set to a default value by events outside the scope of this command; see section 7.1 for details.

6.3 Network Control Model Notifications

[USBCDC12] defines the common Communication Class notifications that the function uses to notify the host of events related to that function. These notifications are sent over the interrupt IN pipe that is in the Communications Interface.

The class-specific notifications valid for an NCM Communication Interface are listed in Table 6-5.

Deleted: Table 6-4

Table 6-6 lists the corresponding notification codes. For convenience, we repeat the codes defined in [USBCDC12].

Deleted: Table 6-5

Certain operational and sequencing requirements, which are more specific than the requirements in [USBCDC12], are imposed for notifications in section 7.1 and 7.1.

Table 6-5: Networking Control Model Notifications

Deleted: 6-4

Notification	Description	Req'd/Opt	reference
NetworkConnection	Reports whether or not the physical layer (modem, Ethernet PHY, etc.) link is up.	Required	[USBCDC12]
ResponseAvailable	Notification to host to issue a *GetEncapsulatedRes-ponse* request.	Optional (Mandatory if *SetEncapsulated-Command* and *Ge-tEncapsulatedRes-ponse* are sup-ported)	[USBCDC12]
ConnectionSpeedChange	Reports a change in upstream or downstream speed of the networking connection.	Required	[USBCDC12]

Table 6-6: Class-Specific Notification Codes for Networking Control Model subclass

Formatted: Caption

Request	Value
NETWORK_CONNECTION	00h
RESPONSE_AVAILABLE	01h
CONNECTION_SPEED_CHANGE	2Ah

7 Operational Constraints

7.1 Notification Sequencing

NCM functions are required to send ConnectionSpeedChange and NetworkConnection notifications in a specific order. To simplify the coding of host device drivers, functions that are going to send a *Network-Connection* notification with wValue == 0001h must first send a *ConnectionSpeedChange* notification that indicates the connection speed that will be in effect when the new connection takes effect.

This sequencing is justified as follows. If the *ConnectionSpeedChange* follows a *NetworkConnection* notification, then the host driver cannot signal network connection with the correct speed until the *Connection-SpeedChange* is received. This delay may introduce latency between bus events and system events, or may cause host system overhead due to a spurious change in speed. If the function signals the connection speed first, then the host driver will know the signaling speed at the time the network connection becomes valid.

Developers should be aware that this sequence of notifications (speed before connection) is different than the sequence specified in [USBECM12].

7.2 Using Alternate Settings to Reset an NCM Function

To place the network aspects of a function in a known state, the host shall:

- select alternate setting 0 of the NCM Data Interface (this is the setting with no endpoints). This can be done explicitly using *SetInterface*, or implicitly using *SetConfiguration*. See [USB30] for details.
- select the NCM operational parameters by sending commands to the NCM Communication Interface, then
- select the second alternate interface setting of the NCM Data Interface (this is the setting with a bulk IN endpoint and a bulk OUT endpoint).

Deleted: , and with *bInterfaceProtocol* set to 01h, as given in Table 4-3

Whenever alternate setting 0 is selected by the host, the function shall:

- flush function buffers
- reset the packet filter to its default state
- clear all multicast address filters
- clear all power filters set using *SetEthernetPowerManagementPatternFilter*
- reset statistics counters to zero
- restore its Ethernet address to its default state
- reset its IN NTB size to the value given by field *dwNtbInMaxSize* from the NTB Parameter Structure

- reset the NTB format to NTB-16
- reset the current Maximum Datagram Size to a function-specific default. If *SetMaxDatagramSize* is not supported, then the maximum datagram size shall be the same as the value in *wMaxSegmentSize* of the Ethernet Networking Functional Descriptor. If *SetMaxDatagramSize* is supported by the function, then the host must either query the function to determine the current effective maximum datagram size, or must explicitly set the maximum datagram size. If the host wishes to set the Maximum Datagram Size, it may do so prior to selecting the second alternate interface setting of the data interface. Doing so will ensure that the change takes effect prior to send or receiving data.
- reset CRC mode so that the function will not append CRCs to datagrams sent on the IN pipe
- reset NTB sequence numbers to zero

Deleted: Segment

When the host selects the second alternate interface setting of the NCM Data Interface, the function shall perform the following actions in the following order.

- If connected to the network, the function shall send a *ConnectionSpeedChange* notification to the host indicating the current connection speed.
- Whether connected or not, the function shall then send a *NetworkConnection* notification to the host, with *wValue* indicating the current state of network connectivity

7.3 Remote Wakeup and Network Traffic

USB Devices containing NCM functions may support remote wakeup. There are two general situations in which remote wakeup may be used:

1. to awaken a link that has been selectively suspended ("link suspend");
2. to awaken the USB host from a system suspend state ("system suspend").

A function cannot distinguish between cases 1 and 2 at the bus level. In case 1, the function should signal remote wakeup whenever traffic is received from the network or when an indication should be passed to the host. In case 2, the link should be awakened only when it's time to wake up the host system. NCM functions distinguish between these two cases based on whether the host has used *SetEthernetPowerManagementPatternFilter* to set up an active power management filter. If GetEthernetPowerManagementPatternFilter would return 0001h at the time that the device or function enters suspend, then the function shall follow the rules for system suspend given in 7.3.2. Otherwise, the function shall follow the rules for link suspend given in 7.3.1.

7.3.1 Remote Wakeup Rules for Link Suspend

In this case, if remote wakeup is enabled and if the device is suspended, the NCM function shall request a remote wakeup whenever:

1. Enough time has elapsed since suspend to allow remote wakeup to be signaled according to [USB30], AND
2. The function has not yet signaled remote wakeup or received remote wakeup from the upstream hub, AND

3. A non-zero alternate setting was selected for the Data Interface prior to suspend, AND

4. EITHER network traffic is available for the host over the bulk IN pipe of the Data Interface, OR notifications are available for the host over the interrupt IN pipe of the Communications Interface.

The USB device may define additional remote wakeup conditions. However, these conditions are sufficient to allow the host driver to suspend an NCM function transparently to save system and device power when performance is not critical.

While the link is suspended, functions shall make best efforts to retain network traffic and notifications that cause the wakeup condition, and any traffic or notifications received between the time that remote wakeup is signaled and when the link wakes up.

7.3.2 Remote Wakeup Rules for System Suspend

In this case, if remote wakeup is enabled and the device is suspended, the NCM function shall request a remote wakeup whenever:

1. Enough time has elapsed since suspend to allow remote wakeup to be signaled according to [USB30], AND

2. The function has not yet signaled remote wakeup or received remote wakeup from the upstream hub, AND

3. A non-zero alternate setting was selected for the Data Interface prior to suspend, AND

4. A packet matching the Ethernet power management filter pattern is received from the network.

While the link is suspended, functions shall observe changes in network connectivity and connection speed, but these changes shall not cause a remote wakeup to be signaled. Upon resume, if the network connectivity or connection speeds have changed compared to the state when the link was suspended, the function shall send *ConnectionSpeedChange* and *NetworkConnect* notifications to inform the host of the new network connection state.

Network traffic received while the function is suspended, other than packets matching the power management filter, shall be discarded by the function.

7.4 Using the Interface Association Descriptor

USB Device containing NCM functions may include an Interface Association Descriptor (IAD) to indicate that the Communication Class interface and the Data Class Interface are to be treated as a single function. (This is an alternative to using the WMC WHCM Union descriptor.)

If an IAD is used, the IAD should appear before the Communication Class Interface. The IAD's *bFunctionClass* shall be set to 02h (Communication Class), which is the same value used in *bInterfaceClass* of the Communication Class interface; *bFunctionSubClass* shall be set to 0Dh (Network Control Model), which is the same value used in *bInterfaceSubClass* of the Communication Class interface; and *bFunctionProtocol* shall be set to the same value as is used in *bInterfaceProtocol* of the Communication Class interface for the function. In accordance with the IAD ECN, *bInterfaceCount* shall be set to 2, and *bFirstInterface* shall be set to the interface number of the Communication Class Interface.

When using an IAD, the general order of descriptors for an NCM function should be:

- Interface Association Descriptor (IAD)
- Communications Class Interface Descriptor (interface n)
- Functional descriptors for the Communication Class Interface
- Endpoint descriptors for the Communication Class Interface
- Data Class Interface (interface n+1, alternate setting 0)
- Functional descriptors for Data Class Interface (interface n+1, alternate setting 0)
- Data Class Interface (interface n+1, alternate setting 1)
- Functional descriptors for Data Class Interface (interface n+1, alternate setting 1)
- Endpoint descriptors for Data Class Interface (interface n+1, alternate setting 1)