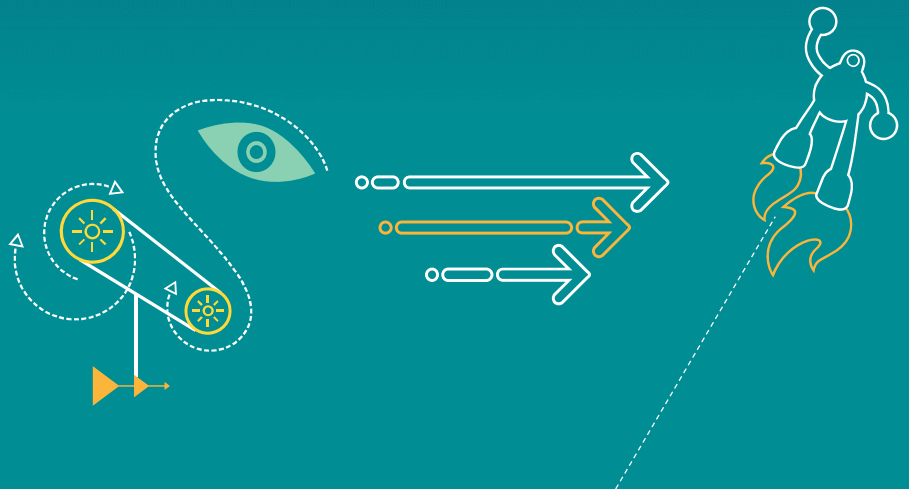
MBIM Overview

Qualcomm Technologies, Inc.



80-NF403-1 C

Confidential and Proprietary – Qualcomm Technologies, Inc.

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

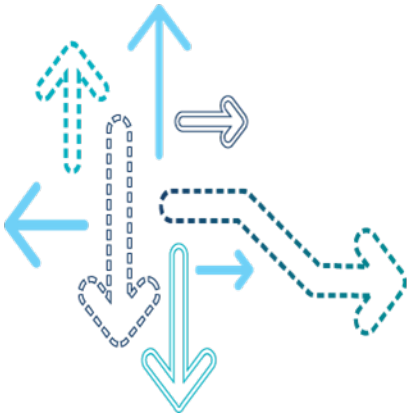
Revision	Date	Description
A	Feb 2013	Initial release
B	Feb 2013	Added Encapsulated Command (MBIM OPEN) Lecroy Log Example, Interrupt Endpoint Lecroy Example, and MBIM PL Support and CR/FR slides; updated MBIM Overview, MBIM Message Header, Data Formats, and IP Data Lecroy Log slides
C	Feb 2015	Numerous changes were made to this document; it should be read in its entirety

Contents

Mobile Broadband History

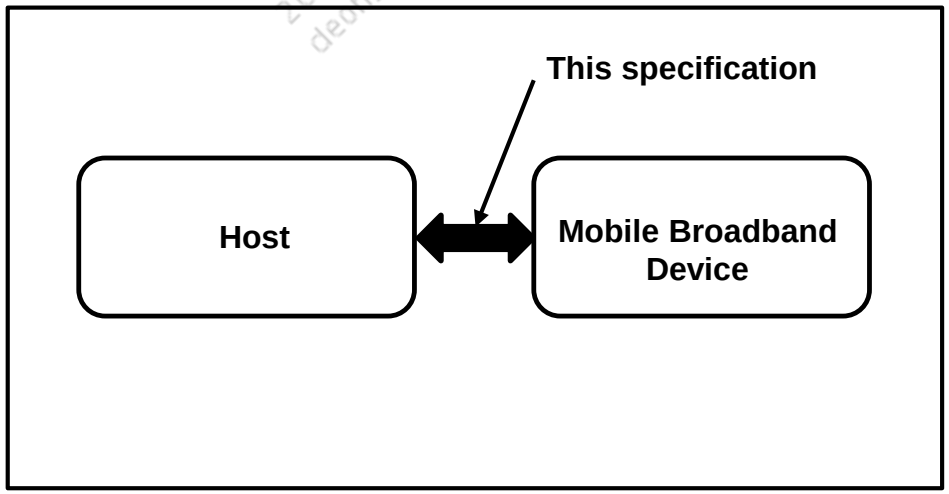
- To use mobile broadband on Windows 7 devices, there were a number of hurdles to overcome.
  - Users – Apart from the requisite mobile broadband hardware, e.g., mobile broadband dongle or embedded module and SIM, and data plan, it was also necessary to locate and install third-party device drivers. If the software was not present in CD/DVD, another internet connection was required to download it.
  - Providers – There were challenges with developing the function driver.
- On Windows 8 devices, mobile broadband provides a first-class connectivity experience, right alongside Wi-Fi.
- The Mobile Broadband Interface Model (MBIM) class driver is an inbox driver provided my Microsoft; no third-party driver is required.

Overview



What is MBIM?

- MBIM is Communications Class Subclass Specification for Mobile Broadband Interface Model.
- It is a protocol by which the USB hosts and mobile broadband devices can efficiently exchange control commands and data frames.
- MBIM extends the Network Control Model (NCM) as a protocol between the host and USB devices, with the difference that devices transfer raw IP packets instead of packets with 802.3 headers.



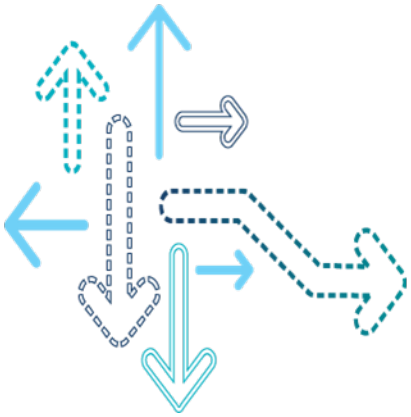
CDC Protocols before MBIM

- Ethernet Control Model (ECM) [USBECM12] – Designed for USB full-speed devices, especially to support DOCSIS 1.0 cable modems; it does not scale well in throughput or efficiency to higher USB speeds.
- Ethernet Emulation Model (EEM) [USBEEM10] – 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.
- ATM Networking Control Model [USBATM12] – Applicable to USB-connected devices like Asymmetrical Digital Subscriber Line (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.
- CDC NCM Subclass Revision [USBNCM10] – Although NCM was targeted for high-speed network attachments like HSPA and LTE, it lacks the definition of control commands and has packet (dis)assembly overhead to carry the raw IP encapsulated in an Ethernet frame.

MBIM Overview

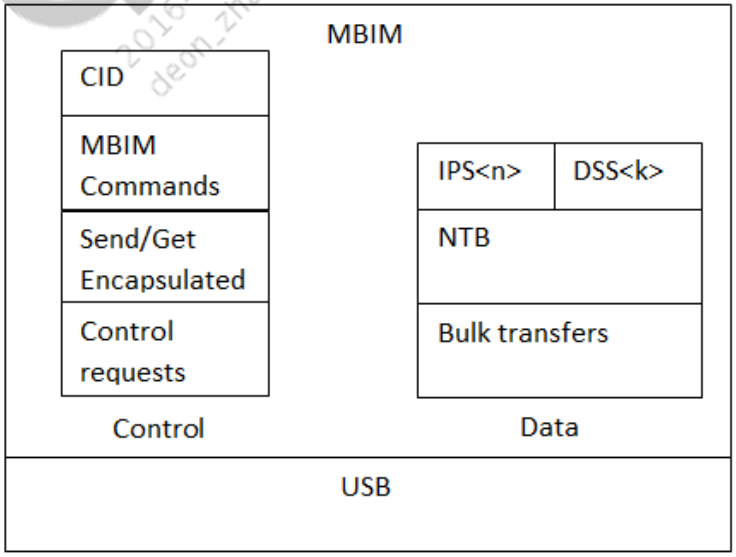
- MBIM 1.0 specification features
  - Supports multiple IP connections per a single USB interface
  - Enables flexible, efficient, and lower cost device implementations
  - Supports power-friendly implementations in diverse Operating Systems (OS)
  - Enables implementation of device-agnostic mobile broadband class drivers in diverse OSs for desktops, laptops, tablets, and mobile devices
  - Replaces cumbersome control channel mechanisms, such as AT commands
  - Minimizes overhead and improves data transfer efficiency by sending raw IP frames, eliminating the need for Ethernet headers

Protocol Basics



Protocol Basics

- MBIM contains two interfaces
  - Communication class
  - Data class
- There are two ways to group these interfaces
  - Interface Association Descriptor (IAD)
  - Union Functional Descriptor
- IAD is used to group interfaces on Qualcomm Technologies, Inc. (QTI) devices.



Interfaces

- MBIM and NCM can exist together using alternate settings. If both of them coexist, settings for the interfaces are as shown in the tables.

Class	NCM	MBIM
Communication Class	bAlternateSetting = 0, bInterfaceClass = 02h, bInterfaceSubClass = 0Dh, bInterfaceProtocol = xxH	bAlternateSetting=1 bInterfaceClass = 02h bInterfaceSubClass=0Eh bInterfaceProtocol = 00h

Class	NCM		MBIM	
Data class	Data Interface 0 No Endpoint bInterfaceClass = 0Ah, bInterfaceSubClass= 00h, bInterfaceProtocol = 02H	Data Interface 1 1 BULK IN 1 BULK OUT 1 Interrupt Endpoint bInterfaceClass = 0Ah, bInterfaceSubClass= 00h, bInterfaceProtocol = 02H	Data Interface 0 No Endpoint	Data Interface 2 1 BULK IN 1 BULK OUT 1 Interrupt Endpoint

Interfaces (cont.)

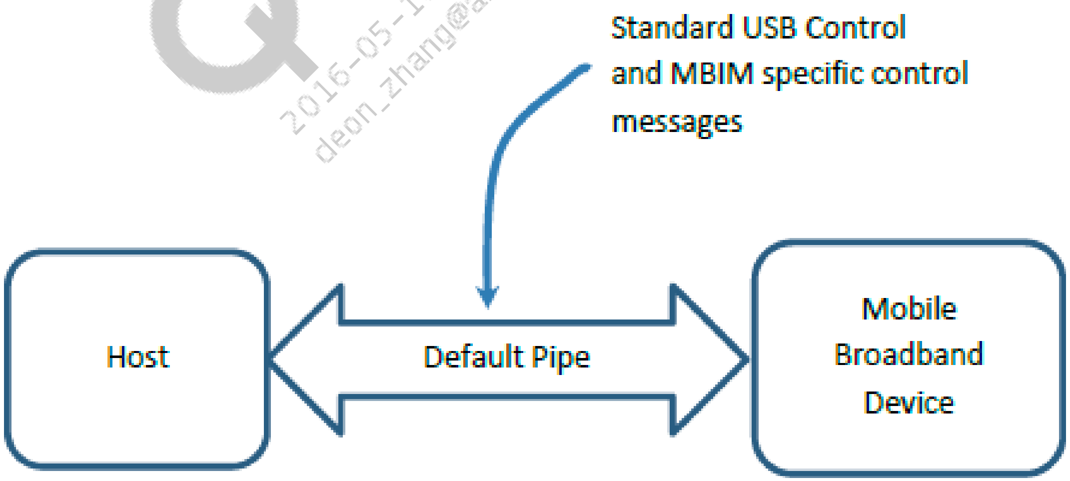
- The data class has three alternate settings. The first two alternate settings are formatted per the NCM specification. The third setting is almost identical to the second setting with the only difference being that bInterfaceProtocol is set to 02h.
- For communication interface setting 0, the only valid settings for the data interface are 0 and 1, i.e., [USBNCM10] compatible mode.
- For communication interface setting 1, the only valid settings for the data interface are 0 and 2, i.e., MBIM mode.
- Apart from these, there must be an interrupt endpoint for response notification. wMaxPacketSize for the interrupt pipe is not required to be the same in the first alternate setting as it is in the alternate setting.

Summary of Differences Between MBIM and NCM 1.0

Request	Changes from NCM 1.0	Comments
SendEncapsulatedCommand	Yes	Required for MBIM functions
GetEncapsulatedResponse	Yes	Required for MBIM functions
GetNtbParameters	No	
GetNtbFormat	No	
SetNtbFormat	No	
GetNtbInputSize	No	
SetNtbInputSize	No	
GetMaxDatagramSize	No	
SetMaxDatagramSize	No	

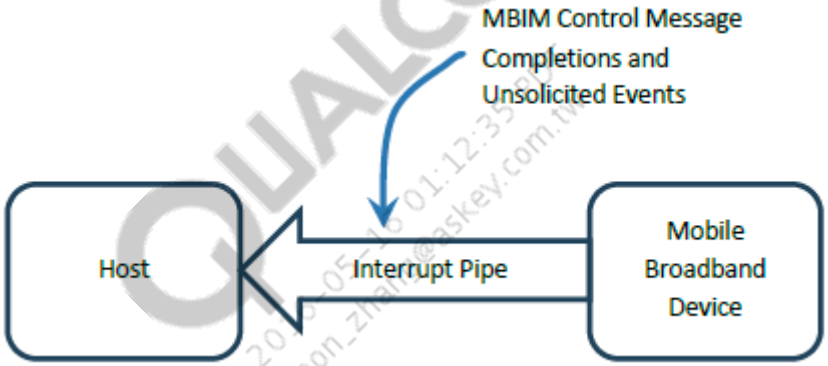
Communication Pipes

- In addition to the default pipe, each MBIM function supports one interrupt, one bulk-in, and one bulk-out pipe
- Default pipe – Default-pipe messages are generally used to control a USB device. These messages include standard requests, such as GET_DESCRIPTOR and SET_CONFIGURATION. Commands that are sent on the default pipe report information back to the host on the default pipe. If an error occurs, they generate a standard USB error token.



Communication Pipes (cont.)

- Interrupt pipe – Interrupt pipe messages are used to alert the host of an asynchronous event from the device. The event could either be an unsolicited event or a response to a previously issued class-specific control message.



- Bulk-in and bulk-out pipes – The host sends data to the MBIM function using the bulk-OUT pipe; the MBIM function sends data to the host using the bulk-IN pipe.

Note: It is recommended that the values of dwNtbInMaxSize and dwNtbOutMaxSize (which dictate the maximum transfer sizes for the bulk-in and bulk-out pipes, respectively) be multiples of the wMaxPacketSize of the respective bulk pipes. If the transfer in either direction is less than the configured Max NTB size and a multiple of wMaxPacketSize, the sender must terminate the transfer with a Zero Length Packet (ZLP). The sender may choose to pad the transfer out Max NTB size to avoid the ZLP at its discretion.

Class-Specific Code

Table 4-1: MBIM Communications Interface Subclass Code

Code	Subclass
0Eh	Mobile Broadband Interface Model

Table 4-2: MBIM Communications Interface Protocol Code

Code	Protocol Code
00h	Mobile Broadband Interface Model Protocol

Table 4-3: MBIM Descriptor Codes

Value	Descriptor Code
1Bh	Mobile Broadband Interface Model Functional Descriptor
1Ch	Extended Mobile Broadband Interface Model Functional Descriptor

Table 4-4: MBIM Class-Specific Request Codes

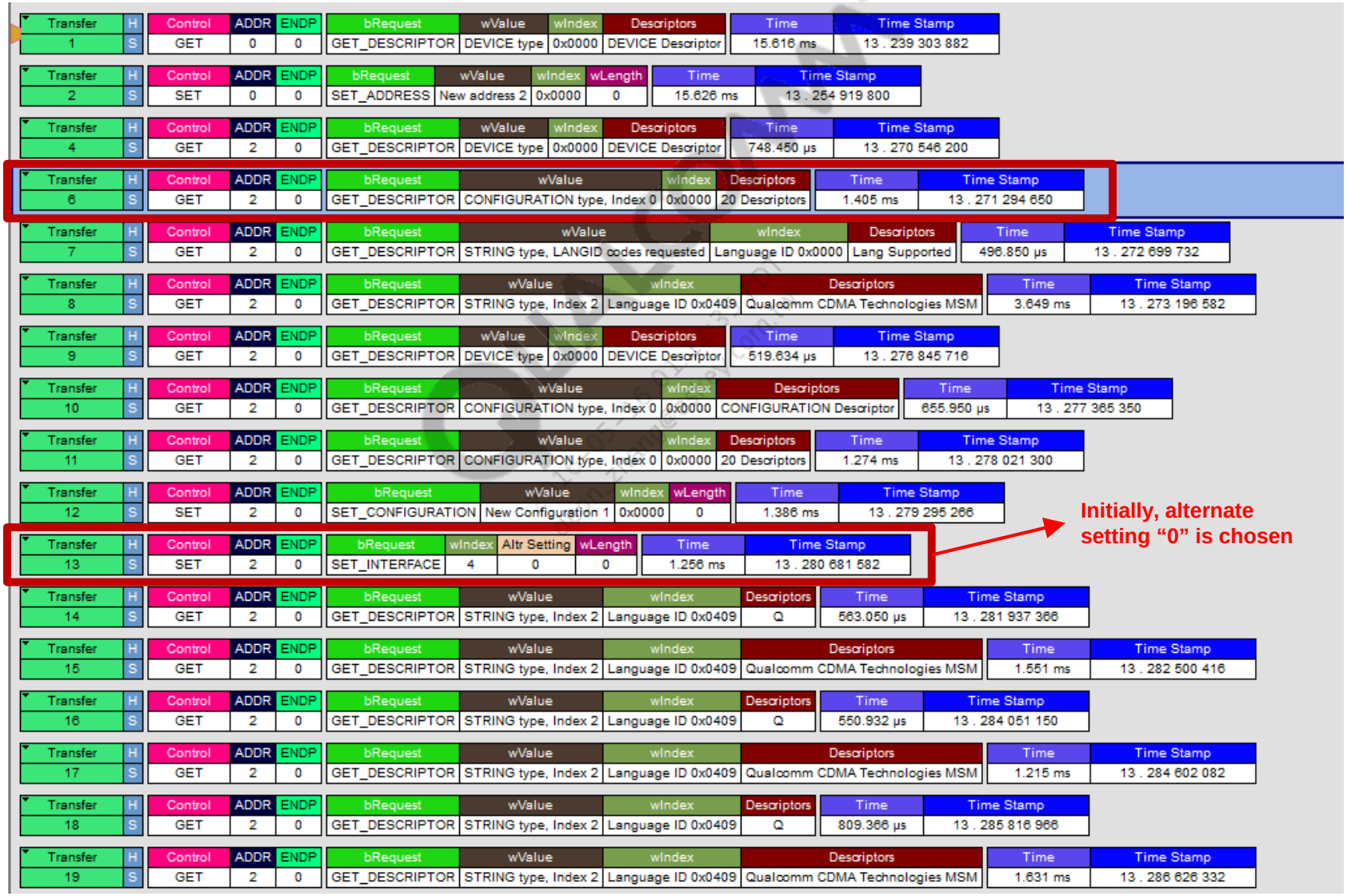
Value	Request code
05h	RESET_FUNCTION

MBIM and NCM Descriptors

- Functional descriptors MBIM and NCM functions

Descriptor	Description	Required/optional	Order	Reference
HEADER	CDC header functional descriptor	Required	First	Section 5.2.3.1 [USBCDC12]
UNION	CDC union functional descriptor	Required	Arbitrary	Section 5.2.3.2 [USBCDC12]
Ethernet	CDC Ethernet networking functional descriptor	Required	Arbitrary	Section 5.4 [USBECM12],
NCM	NCM functional descriptor	Required	Arbitrary	Section 5.2.1 [USBNCM10]
COMMAND SET	Command set functional descriptor	Required if NCM communications interface bInterfaceProtocol is 0FEh	Arbitrary	Section 5.2.2 [USBNCM10]
COMMAND SET DETAIL	Command set detail functional descriptor	Optional if command set functional descriptor is present; otherwise prohibited	After command set functional descriptor	Section 5.2.3 [USBNCM10]
MBIM	MBIM functional descriptor	Required	Arbitrary	Table 6-3: MBIM functional descriptor
MBIM Extended	MBIM extended functional descriptor	Optional	After MBIM functional descriptor	Table 6-4: MBIM extended functional descriptor

MBIM Enumeration



Transfer	H	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors	Time	Time Stamp
1	S	GET	0	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor	15.616 ms	13 . 239 303 882
2	S	SET	0	0	SET_ADDRESS	New address 2	0x0000	0	15.626 ms	13 . 254 919 800
4	S	GET	2	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor	748.450 µs	13 . 270 546 200
6	S	GET	2	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	20 Descriptors	1.405 ms	13 . 271 294 650
7	S	GET	2	0	GET_DESCRIPTOR	STRING type, LANGID codes requested	Language ID 0x0000	Lang Supported	496.850 µs	13 . 272 699 732
8	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	3.649 ms	13 . 273 196 582
9	S	GET	2	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor	519.634 µs	13 . 276 845 716
10	S	GET	2	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	CONFIGURATION Descriptor	655.950 µs	13 . 277 365 350
11	S	GET	2	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	20 Descriptors	1.274 ms	13 . 278 021 300
12	S	SET	2	0	SET_CONFIGURATION	New Configuration 1	0x0000	0	1.386 ms	13 . 279 295 266
13	S	SET	2	0	SET_INTERFACE	4	0	0	1.256 ms	13 . 280 681 582
14	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Q	563.050 µs	13 . 281 937 366
15	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	1.551 ms	13 . 282 500 416
16	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Q	550.932 µs	13 . 284 051 150
17	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	1.215 ms	13 . 284 602 082
18	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Q	809.366 µs	13 . 285 816 966
19	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	1.631 ms	13 . 286 626 332

Initially, alternate setting "0" is chosen

MBIM Enumeration (cont.)

Transfer	H	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors	Time	Time Stamp
19	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	1.631 ms	13 . 286 626 332
20	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Q	599.284 µs	13 . 288 257 816
21	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Qualcomm CDMA Technologies MSM	7.937 ms	13 . 288 857 100
22	S	GET	2	0	GET_STATUS	0x0000	USB 2.0 Standard Status	2	0x0000	624.950 µs	13 . 296 794 432
23	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 3	Language ID 0x0000	Lang Supported	691.168 µs	13 . 297 419 382
24	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 3	Language ID 0x0000	Lang Supported	498.982 µs	13 . 298 110 550
25	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0000	Lang Supported	512.734 µs	13 . 298 609 532
26	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0000	Lang Supported	661.334 µs	13 . 299 122 266
27	S	GET	2	0	Comm Request	bRequest	wValue	wIndex	wLength	NTB_Parameter	Time	Time Stamp
						GET_NTB_PARAMETERS	0x0000	0x0003	28	Parameters	571.932 µs	13 . 299 783 600
28	S	SET	2	0	Comm Request	bRequest	wIndex	wLength	Time	Time Stamp
						RESET_FUNCTION	Interface 3	0	389.134 µs	13 . 300 355 532
29	S	SET	2	0	Comm Request	bRequest	wValue	wIndex	wLength	Maximum_NTB_Size	Time	Time Stamp
						SET_NTB_INPUT_SIZE	0x0000	0x0003	4	0x400000	684.600 µs	13 . 300 744 666
30	S	SET	2	0	Comm Request	bRequest	wIndex	wLength	Time	Time Stamp
						RESET_FUNCTION	Interface 3	0	433.384 µs	13 . 301 429 266
31	S	SET	2	0	Comm Request	bRequest	wValue	wIndex	wLength	Maximum_NTB_Size	Time	Time Stamp
						SET_NTB_INPUT_SIZE	0x0000	0x0003	4	0x400000	608.482 µs	13 . 301 862 650
32	S	SET	2	0	bRequest	wIndex	Altr Setting	wLength	Time	Time Stamp
					SET_INTERFACE	4	1	0	849.684 µs	13 . 302 471 132
33	S	SET	2	0	Comm Request	bRequest	wIndex	wLength	Data	Time	Time Stamp
						SEND_ENCAPSULATED_COMMAND	Interface 3	16	15 bytes	5.034 sec	13 . 303 320 816

After the NTB parameters are set and the interface is reset, alternate setting 1 is set.

Device Descriptor Decoded

DECODING INFORMATION

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bRequest	8	8	GET_DESCRIPTOR	0x06	bRequest HexVal: 0x06
wValue	16	16	DEVICE type	0x0100	Type of Descriptor
wIndex	16	32	0x0000	0x0000	index info

DEVICE Descriptor

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	0	0x12	0x12	Descriptor size is 18 bytes
bDescriptorType	8	8	0x01	0x01	DEVICE Descriptor Type
bcdUSB	16	16	0x0200	0x0200	USB Specification version 2.00
bDeviceClass	8	32	0xEF	0xEF	The device belongs to the Miscellaneous Device Class
bDeviceSubClass	8	40	0x02	0x02	The device belongs to the Common Class Subclass
bDeviceProtocol	8	48	0x01	0x01	The device uses the Interface Association Descriptor Protocol
bMaxPacketSize0	8	56	0x40	0x40	Maximum packet size for endpoint zero is 64
idVendor	16	64	0x05C6	0x05C6	Vendor ID is 1478: Qualcomm, Inc
idProduct	16	80	0x9043	0x9043	Product ID is 36931
bcdDevice	16	96	0x0000	0x0000	The device release number is 0.00
iManufacturer	8	112	0x03	0x03	The manufacturer string descriptor index is 3
iProduct	8	120	0x02	0x02	The product string descriptor index is 2
iSerialNumber	8	128	0x00	0x00	The device doesn't have the string descriptor describing the serial number
bNumConfigurations	8	136	0x01	0x01	The device has 1 possible configurations

Configuration Descriptor Decoded

INTERFACE_ASSOCIATION Descriptor (8 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	680	0x08	0x08	Descriptor size is 8 bytes
bDescriptorType	8	688	0x0B	0x0B	INTERFACE_ASSOCIATION Descriptor Type
bFirstInterface	8	696	0x03	0x03	The first interface number associated with this function is 3
bInterfaceCount	8	704	0x02	0x02	The number of contiguous interfaces associated with this function is 2
bFunctionClass	8	712	0x02	0x02	The function belongs to the Communication Device/Interface Class
bFunctionSubClass	8	720	0x0E	0x0E	Subclass 0x0E not defined
bFunctionProtocol	8	728	0x00	0x00	The function uses the No class specific protocol required Protocol
iFunction	8	736	0x00	0x00	The Function string descriptor index is 0

INTERFACE Descriptor (9 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	744	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	752	0x04	0x04	INTERFACE Descriptor Type
bInterfaceNumber	8	760	0x03	0x03	The number of this interface is 3.
bAlternateSetting	8	768	0x00	0x00	The value used to select the alternate setting for this interface is 0
bNumEndpoints	8	776	0x01	0x01	The number of endpoints used by this interface is 1 (excluding endpoint zero)
bInterfaceClass	8	784	0x02	0x02	The interface implements the Communication Device/Interface class
bInterfaceSubClass	8	792	0x0E	0x0E	SubClass 0x0E not defined
bInterfaceProtocol	8	800	0x00	0x00	The interface uses the No class specific protocol required Protocol
iInterface	8	808	0x00	0x00	The device doesn't have a string descriptor describing this iInterface

Communication Class Specific INTERFACE Descriptor (5 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength....	8	816	0x05	0x05	Size of the descriptor, in bytes
bDescriptorType	8	824	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	832	0x00	0x00	Header Funtional Descriptor Subtype
bcdCDC	16	840	0x0110	0x0110	USB Class Definitions for Communications the Communication specification version 1.10

Configuration Descriptor Decoded (cont.)

Communication Class Specific INTERFACE Descriptor (12 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength....	8	856	0x0C	0x0C	Size of the descriptor, in bytes
bDescriptorType	8	864	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	872	0x1B	0x1B	Mobile Broadband Interface Model Functional Descriptor Subtype
bcdMBIMVersion	16	880	0x0100	0x0100	Release number of this specification in BCD
wMaxControlMessage	16	896	0x1000	0x1000	Maximum segment size in bytes a function
bNumberFilters	8	912	0x10	0x10	Contains the number of PacketFilter that are available in total for all SessionIds
bMaxFilterSize	8	920	0x80	0x80	The Maximum size of the PacketFilters in bytes
wMaxSegmentSize	16	928	0x0FE0	0x0FE0	The Maximum segment size in bytes that the function is capable of supporting
bmNetworkCapabilities	8	944	0x20	0x20	Specifies the capabilities of this function

ENDPOINT Descriptor (7 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	952	0x07	0x07	Descriptor size is 7 bytes
bDescriptorType	8	960	0x05	0x05	ENDPOINT Descriptor Type
bEndpointAddress	8	968	0x85	0x85	This is an IN endpoint with endpoint number 5
bmAttributes	8	976	0x03	0x03	Types - Transfer: INTERRUPT Low Power: No Pkt Size Adjust: No
wMaxPacketSize	16	984	0x0040	0x0040	Maximum packet size for this endpoint is 64 Bytes. If Hi-Speed, 0 additional transactions per frame
bInterval	8	1000	0x05	0x05	The polling interval value is every 5 Frames. If Hi-Speed, every 16 uFrames

Configuration Descriptor Decoded (cont.)

INTERFACE Descriptor (9 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1008	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1016	0x04	0x04	INTERFACE Descriptor Type
bInterfaceNumber	8	1024	0x04	0x04	The number of this interface is 4.
bAlternateSetting	8	1032	0x00	0x00	The value used to select the alternate setting for this interface is 0
bNumEndpoints	8	1040	0x00	0x00	The number of endpoints used by this interface is 0 (excluding endpoint zero)
bInterfaceClass	8	1048	0x0A	0x0A	The interface implements the Data Interface class
bInterfaceProtocol	8	1064	0x02	0x02	The interface uses the Reserved Protocol
bInterfaceSubClass	8	1056	0x00	0x00	The Subclass code is 0
iInterface	8	1072	0x00	0x00	The device doesn't have a string descriptor describing this iInterface

INTERFACE Descriptor (9 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1080	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1088	0x04	0x04	INTERFACE Descriptor Type
bInterfaceNumber	8	1096	0x04	0x04	The number of this interface is 4.
bAlternateSetting	8	1104	0x01	0x01	The value used to select the alternate setting for this interface is 1
bNumEndpoints	8	1112	0x02	0x02	The number of endpoints used by this interface is 2 (excluding endpoint zero)
bInterfaceClass	8	1120	0x0A	0x0A	The interface implements the Data Interface class
bInterfaceProtocol	8	1136	0x02	0x02	The interface uses the Reserved Protocol
bInterfaceSubClass	8	1128	0x00	0x00	The Subclass code is 0
iInterface	8	1144	0x00	0x00	The device doesn't have a string descriptor describing this iInterface

ENDPOINT Descriptor (7 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1152	0x07	0x07	Descriptor size is 7 bytes
bDescriptorType	8	1160	0x05	0x05	ENDPOINT Descriptor Type
bEndpointAddress	8	1168	0x86	0x86	This is an IN endpoint with endpoint number 6
bmAttributes	8	1176	0x02	0x02	Types - Transfer: BULK Pkt Size Adjust: No
wMaxPacketSize	16	1184	0x0200	0x0200	Maximum packet size for this endpoint is 512 Bytes. If Hi-Speed, 0 additional transactions per frame
bInterval	8	1200	0x20	0x20	The polling interval value is every 32 Frames. Undefined for Hi-Speed

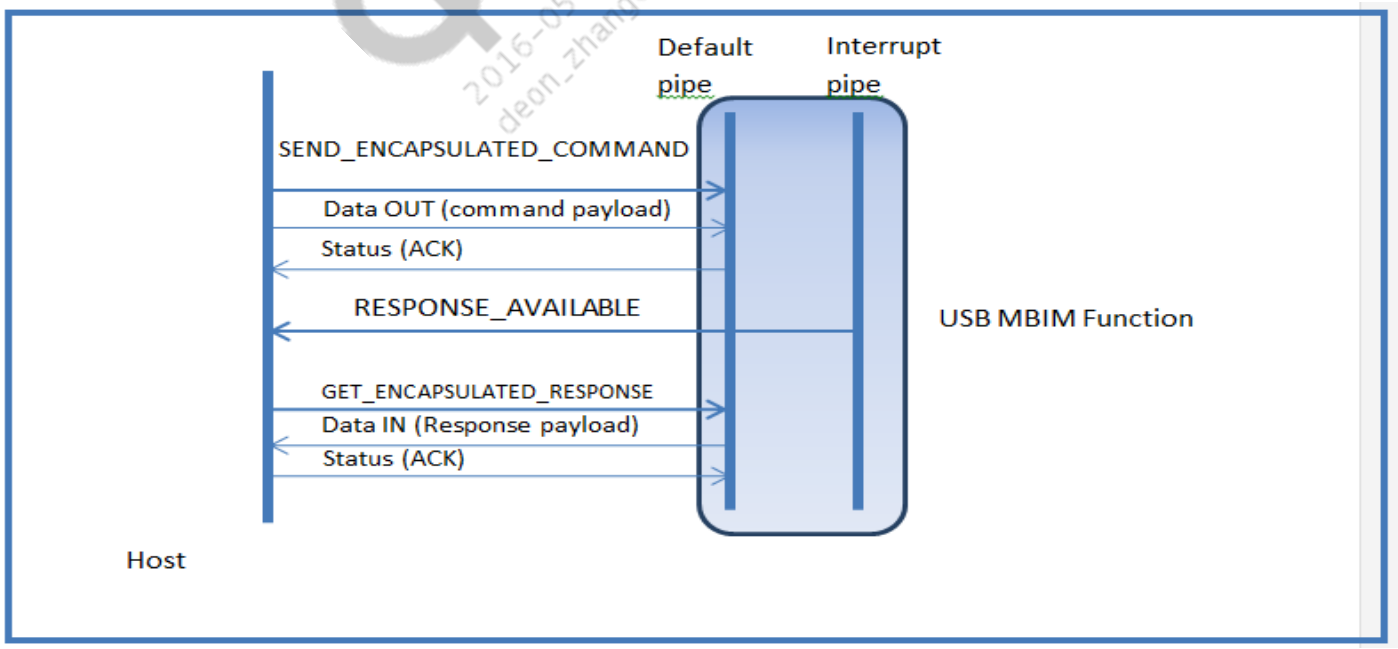
MBIM Extended Functional Descriptor

- This descriptor provides extended information about the implementation of the MBIM function. The descriptor is optional; but if it exists, the descriptor must appear after the MBIM functional descriptor.

Offset	Field	Size	Value	Description
0	bFunctionLength	1	8	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE (0x24)
2	bDescriptorSubtype	1	Constant	MBIM extended functional descriptor code (see Table 4-3)
3	bcdMBIMExtendedVersion	2	Number 0x0100	Release number of MBIM extensions in BCD, with implied decimal point between bits 7 and 8 0x0100 == 1.00 == 1.0 A little-endian constant, so the bytes are 0x00, 0x01
5	bMaxOutstandingCommand Messages	1	Number	Max number of outstanding command messages the device can handle simultaneously; is greater than 0
6	wMTU	2	Number	Operator-preferred MTU for home network; wMTU applies to IP data streams. If no specific requirements exist, the recommended value must be 1500

Operational Model

- The MBIM control messages from the host to the device are asynchronous. The host sends a control message to the function on the device's default pipe. The function asynchronously completes the messages and signals availability of the result by sending a notification to the host over the interrupt endpoint. The host then fetches the response using the default pipe. This model allows the host to issue multiple control messages to the device without waiting for the previous message to complete.



Control Commands

- The MBIM uses the control commands shown in the table. SendEncapsulatedCommand encapsulates one of the commands shown on next slide.

Table 8-1: Networking control Model Requests

Request	Description	Required/Optional
SendEncapsulatedCommand	Issues a command in the format of the supported control protocol	Required
GetEncapsulatedResponse	Requests a response in the format of the supported control protocol	Required
GetNtbParameters	Requests the function to report parameters that characterize the network control block	Required
GetNtbFormat	Gets the current NTB format	Optional
SetNtbFormat	Selects 16- or 32-bit network transfer blocks	Optional
GetNtbInputSize	Gets the current value of the maximum NTB input size	Required
SetNtbInputSize	Selects the maximum size of NTBs to be transmitted by the function over the bulk IN pipe	Required
GetMaxDatagramSize	Requests the current maximum datagram size	Optional
SetMaxDatagramSize	Sets the maximum datagram size to a value other than the default	Optional
ResetFunction	Resets the function so it returns to the same state is had after a set config	Required

MBIM Control Messages

Table 9-3: Control messages sent from the host to the function

Message Identifier	Value	Description
MBIM_OPEN_MSG	1	Initializes the function
MBIM_CLOSE_MSG	2	Closes the function
MBIM_COMMAND_MSG	3	Sends a 'COMMAND' CID
MBIM_HOST_ERROR_MSG	4	Indicates an error in the MBIM communication

Table 9-9: Control Messages sent from function to host

Message Identifier	Value	Description
MBIM_OPEN_DONE	80000001h	Device response to initialization request (MBIM_OPEN_MSG)
MBIM_CLOSE_DONE	80000002h	Device response to close request (MBIM_CLOSE_MSG)
MBIM_COMMAND_DONE	80000003h	Device response to command CID request (MBIM_COMMAND_MSG)
MBIM_FUNCTION_ERROR_MSG	80000004h	Indicates an error in the MBIM communication
MBIM_INDICATE_STATUS_MSG	80000007h	Indicates unsolicited network or devices status changes

MBIM RESET

- RESET_FUNCTION resets the function to its initial state. All IP data stream connections are disconnected; all device service streams are closed. If the function is open using MBIM_OPEN, it is returned to a Closed state.
- The function abandons all outstanding transactions that are awaiting completion; no notifications are sent.
- The data toggles of bulk and interrupt pipes are not affected.
- The alternate settings of interfaces are not changed.
- Remote wakeup enable for the function is not affected.
- The intended use of this request is to allow a host to return the function to a Known state. It differs from USB reset, in that only the targeted function is affected by this request.
- Other functions in a composite device are not affected by this request. The USB transport-level state is not affected by this request.

MBIM Message Header

Table 9-1: MBIM_MESSAGE_HEADER

Offset	Size	Field	Type	Description
0	4	MessageType	UNIT32	Specifies the MBIM message type. See Table 9-3: Control messages sent from the host to the function and Table 9-9: Control Messages sent from function to host. The Most Significant Bit (MSB) in the MessageType indicates direction: ▪ MSB set to 0: from host to function ▪ MSB set to 1: from function to host
4	4	MessageLength	UNIT32	Specifies the total length of this MBIM message in bytes
8	4	TransactionId	UNIT32	Specifies the MBIM message ID value. This value is used to match host-sent messages with function responses. This value must be unique among all outstanding transactions. For notifications, the TransactionId must be sent to 0 by the function.

Note: The remaining bytes are command-specific.

MBIM Fragment Header

- If the size of a command or a response is larger than MaxControlTransfer, the message may be split across multiple messages; this header indicates how many fragments there are in total.

Offset	Size	Field	Type	Description
0	4	TotalFragments	UINT32	This field indicates how many fragments there are in total; for fragmentation considerations, see Section 9.5 (Fragmentation of messages) in *MBIM Specification 1.0*.
4	4	CurrentFragment	UINT32	This field indicates the message fragment; values are 0 to TotalFragments - 1.

MBIM Fragmented Messages

- MBIM_COMMAND_MSG, MBIM_COMMAND_DONE, and MBIM_INDICATE_STATUS_MSG may be split across multiple encapsulated commands and responses due to limitation in the maximum size of a USB transfer across the control channel.
- Fragmented messages can be identified by the TotalFragment field having a value greater than 1. The first fragment contains all of the message headers; the following fragments only contain the headers up to the current fragment field followed by the continuation of the InformationBuffer field.

See the below example of a fragmented MBIM_COMMAND_DONE message:

Assume the max USB control transfer is 4096.
The message payload is 14000 bytes.
The message will be split across 4 messages.
Only the first fragment will contain DeviceServiceId, CID, Status, and InformationBufferLength.
The remaining fragments will hold InformationBuffer data following the CurrentFragment Field.
MessageLength is the actual length of each fragment.
InformationBufferLength is the length the whole InformationBuffer.

MessageType
MessageLength(4096)
TransactionId(99)
TotalFragments(4)
CurrentFragment(0)
DeviceServiceId
CID
Status
InformationBufferLength (14000)
InformationBuffer

MessageIdType
MessageLength(4096)
TransactionId(99)
TotalFragments(4)
CurrentFragment(1)
InformationBuffer

MessageType
MessageLength(4096)
TransactionId(99)
TotalFragments(4)
CurrentFragment(2)
InformationBuffer

MessageType
MessageLength(1820)
TransactionId(99)
TotalFragments(4)
CurrentFragment(3)
InformationBuffer

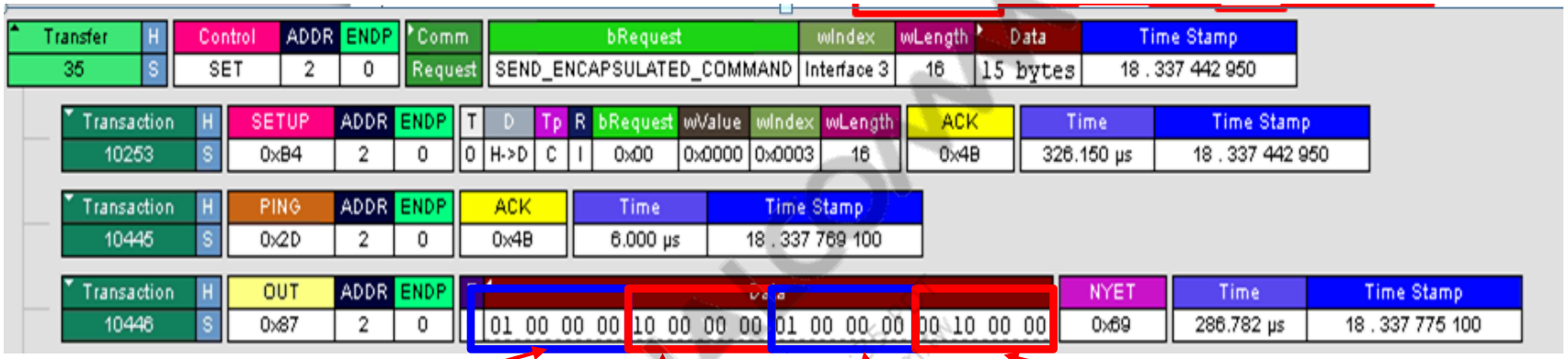
MBIM Fragmented Commands

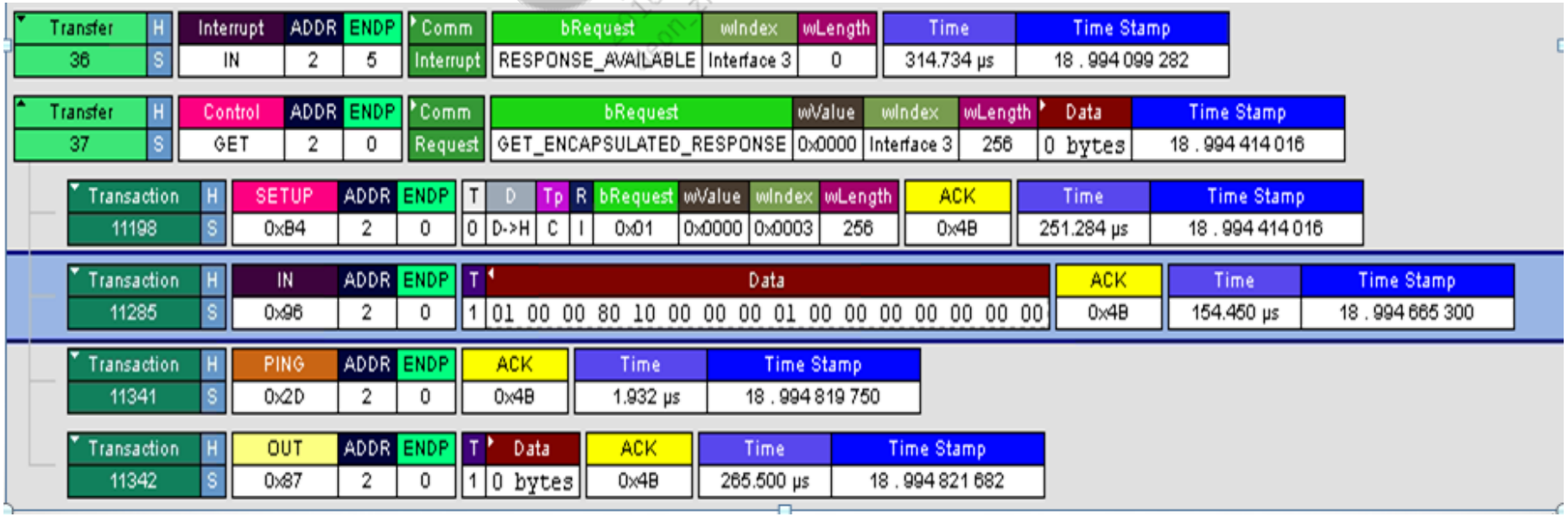
- In a fragment, MessageLength is the length of the current fragment, whereas InformationBufferLength is the total length of the InformationBuffer prior to fragmentation. Fragments are transmitted in back-to-back transfers without intermixing fragments from other messages.
- A host or function that receives fragmented messages sends an MBIM_X_ERROR_MSG with error code MBIM_ERROR_TIMEOUT_ FRAGMENT; if the time between the fragments is too long (see Section 9.3.4.1 in MBM Spec 1.0).
- If a host or function receives a fragment that is out-of-order or is improperly formatted or in any other way is faulty, the receiver sends an MBIM_X_ERROR_MSG with the corresponding error code to the sender (see Table 9-8: MBIM_PROTOCOL_ERROR_CODES in MBM Spec 1.0).
- If a host or function receives a faulty fragment, e.g., incorrect length, out of sequenced fragment, etc., all fragments with the same TransactionId are discarded. If the function receives a first fragment of a new command with a new TransactionId, the function discards the previous command and starts handling the new command. If the function receives a fragment that is not the first fragment of a new command with a new TransactionId, both commands are discarded. One MBIM_FUNCTION_ERROR_MSG message per TransactionId is sent.

MBIM Fragmented Commands (cont.)

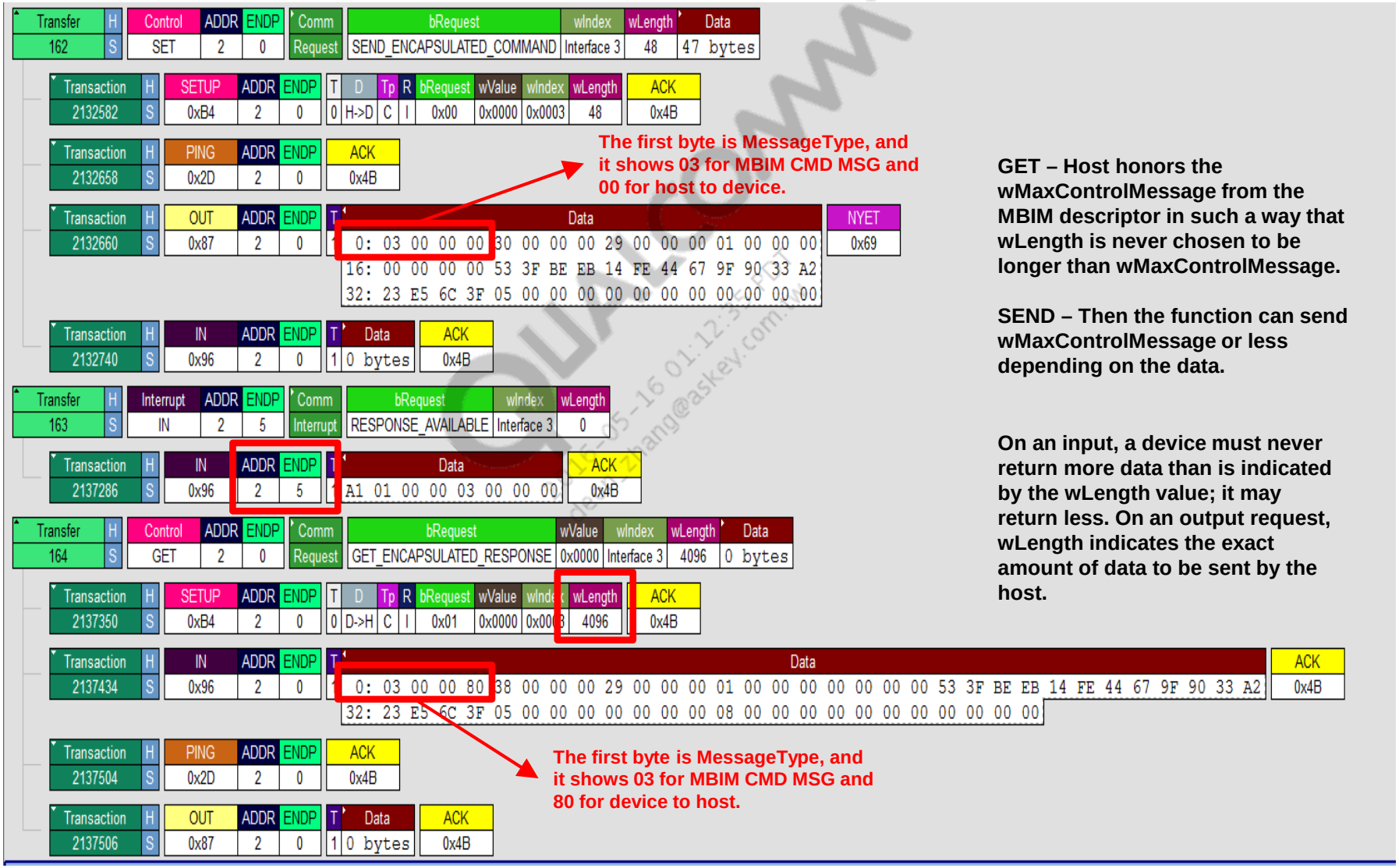
- The function must use a separate GET_ENCAPSULATED_RESPONSE transfer for each control message it has to send to the host, i.e., the function must not concatenate multiple messages into a single GET_ENCAPSULATED_RESPONSE transfer.
- The function must send a RESPONSE_AVAILABLE notification for each available ENCAPSULATED_RESPONSE to be read from the default pipe. For example, if the function has two ENCAPSULATED_RESPONSES available and the first ENCAPSULATED_RESPONSE consists of three fragments, the second one consists of only one fragment, the function should send four RESPONSE_AVAILABLE notifications over the interrupt IN pipe.
- Per [USB30] Section 8.12.2.2 in *MBIM Specification 1.0*, each RESPONSE_AVAILABLE notification must be Zero Length Packet (ZLP) terminated if the length is an exact multiple of the wMaxPacketSize of the endpoint descriptor (see [USB30], Table 9: CDC MBIM Subclass 18) for interrupt endpoint. The ENCAPSULATED_RESPONSE must also be ZLP terminated if the size returned is a multiple of the bMaxPacketSize0 and is not equal to wLength in the GET_ENCAPSULATED_RESPONSE request (see [USB30] Table 9-8).

Encapsulated Command (MBIM OPEN) Lecroy Log Example





Encapsulated Command Lecroy Log



GET – Host honors the wMaxControlMessage from the MBIM descriptor in such a way that wLength is never chosen to be longer than wMaxControlMessage.

SEND – Then the function can send wMaxControlMessage or less depending on the data.

On an input, a device must never return more data than is indicated by the wLength value; it may return less. On an output request, wLength indicates the exact amount of data to be sent by the host.

wMaxControlMessage vs. wMaxSegmentSize

- wMaxControlMessage
  - This is the maximum segment size in bytes a function can handle from a SEND_ENCAPSULATED_COMMAND or can return from a GET_ENCAPSULATED_RESPONSE.
  - This number must not be smaller than 64. However, to avoid frequent fragmentation, e.g., with full SMS or USSD messages, it is recommended that this value be no smaller than 512.
- wMaxSegmentSize
  - This is the maximum segment size in bytes that the function is capable of supporting. This number must be larger than the MTU set for IP traffic by the network.
  - This number must not be smaller than 2048.
  - **Note:** This is not used for IP MTU. For configuring IP MTU, use either MBIM extended functional descriptor or IP MTU handling MBIM_CID_IP_CONFIGURATION.

Data Format

- MBIM follows the same data format as NCM, but it supports both raw IP and device service stream instead of Ethernet frames.
- The USB transfer is formatted as a NCM Transfer Block (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.
  - The NTH effectively points to the head of a list of NCM Datagram Pointers (NDP) structures. In turn, each NDP points to one or more Ipframe/device streams encapsulated within the NTB; e.g.:
    - IPv4 datagrams start with the IPv4 header and continue with the appropriate payload.
    - The format of the device service stream payload depends on the device service (as identified by the corresponding UUID) that is used when opening the data stream; this is *not* supported.
  - Finally, the NTB contains the data itself.

Data Format (cont.)

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

NTH signature – 4E 43 4D 48

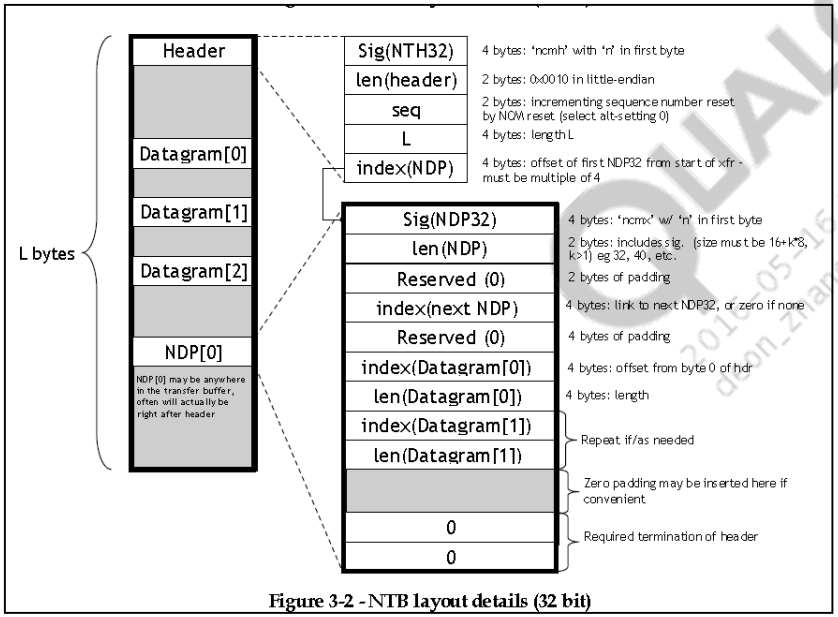


Figure 3-2 - NTB layout details (32 bit)

NDP signature – 49 50 53 00

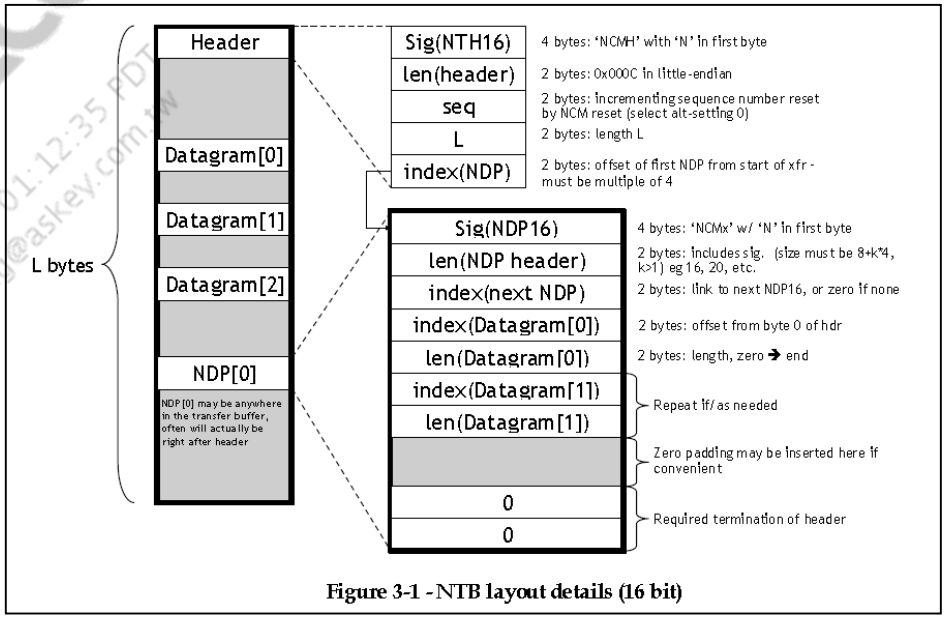


Figure 3-1 - NTB layout details (16 bit)

Data Format

- Device service
  - A logical group of functionality supported by the device constitutes a “device service.” Each device service is uniquely identified by a 128-bit Universally Unique Identifier (UUID). All CIDs exchanged between the host and the device carry the UUID to identify the device service associated with the transfer.
- Custom device services
  - Vendors are free to define additional device services by generating a UUID of their own. These services are not IP protocol-based.
  - This is not supported; IP packets are supported.
- **Note**: To identify the frames as containing payload of either raw IP or device service streams, the valid signature values for NDP structures (Section 3.3 of [USBNCM10]) are extended as follows.
  - To distinguish among the data streams, the last character of the dwSignature in the NDP header is coded with the index SessionId or DssSessionId specified by the host in the MBIM_CID_CONNECT or MBIM_CID_DSS_OPEN command, respectively. The index value is assigned separately for IP and device service streams.

Data Format

Offset	Field	Size	Value	Description
0	dwSignature	4	Number (0x304D434E, 0x314D434E)	Signature of this NDP16; transmitted in little-endian format, i.e., as 0x4E, 0x43, 0x4D, 0x30 or 0x4E, 0x43, 0x4D, 0x31, or as the character sequences NCM0 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	wNextFpIndex	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 format, of the first datagram described by this NDP16; the index is from byte zero of the NTB. This value must be greater than or equal to the value store in wHeaderLength of the NTH16, because it must point past the NTH16.
10	wDatagramLength[0]	2	Number	Byte length, in little-endian format, of the first datagram described by this NDP16; for Ethernet frames, this value must be greater than or equal to 14.
12	wDatagramIndex[1]	2	Number	Byte index, in little-endian format, of the second datagram described by this NDP16. If zero, this marks the end of the sequence of datagrams in this NDP16.
14	wDatagramLength[1]	2	Number	Byte length, in little-endian format, of the second datagram described by this NDP16. If zero, this marks the end of the sequence of datagrams in this NDP16.

- MBIM allows multiple NDPS within one NTB – MBIM allows the usage of this feature; hence, the value wNextNdpIndex in the table is no longer reserved. The new definition is as follows:

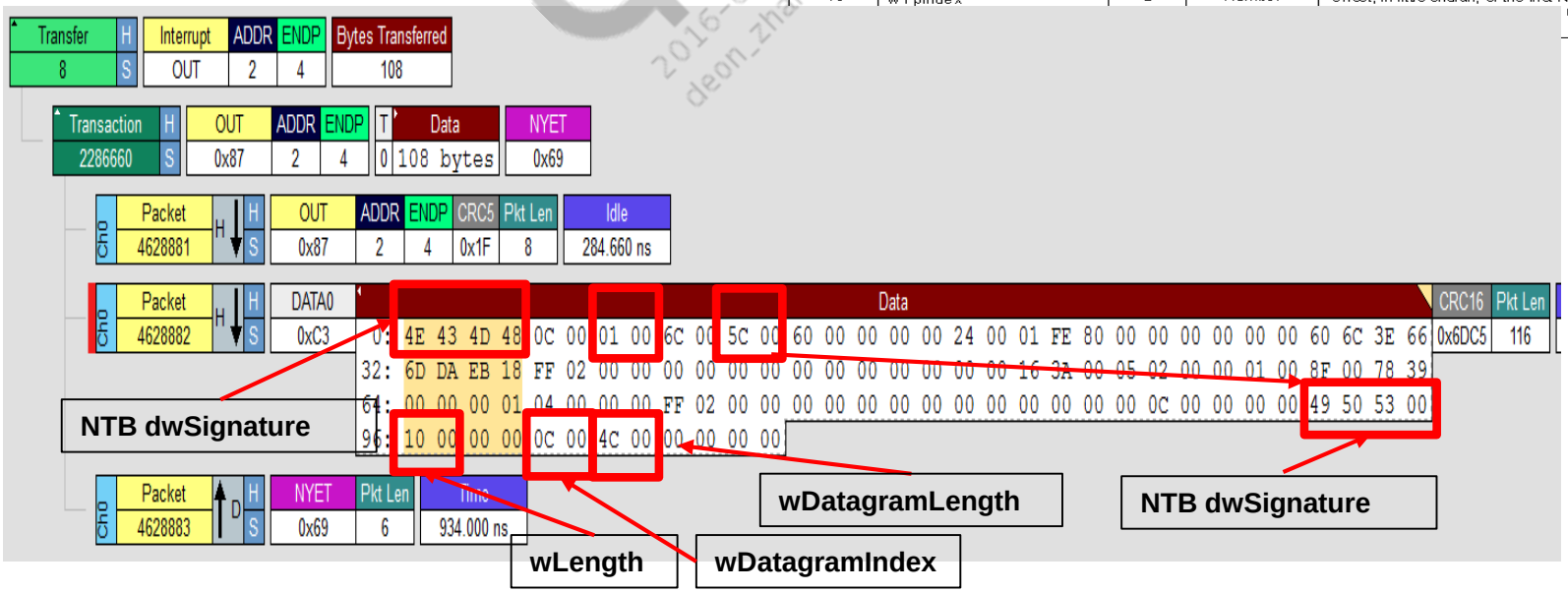
6	wNextNdpIndex	2	Number	Byte index, in little endian format, of the next NDP; the index is from byte zero of the NTB.

- The minimum values for wDatagramLength, corresponding to the field wDatagramLength[0]
  - For IPv4: ≥ 20
  - For IPv6: ≥ 40
  - For DSS: ≥ 0

IP Data Lecroy Log

- Example BULK OUT Transfer

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	wFpIndex	2	Number	Offset, in little endian, of the first NDP16 from byte zero multiple of 4, and

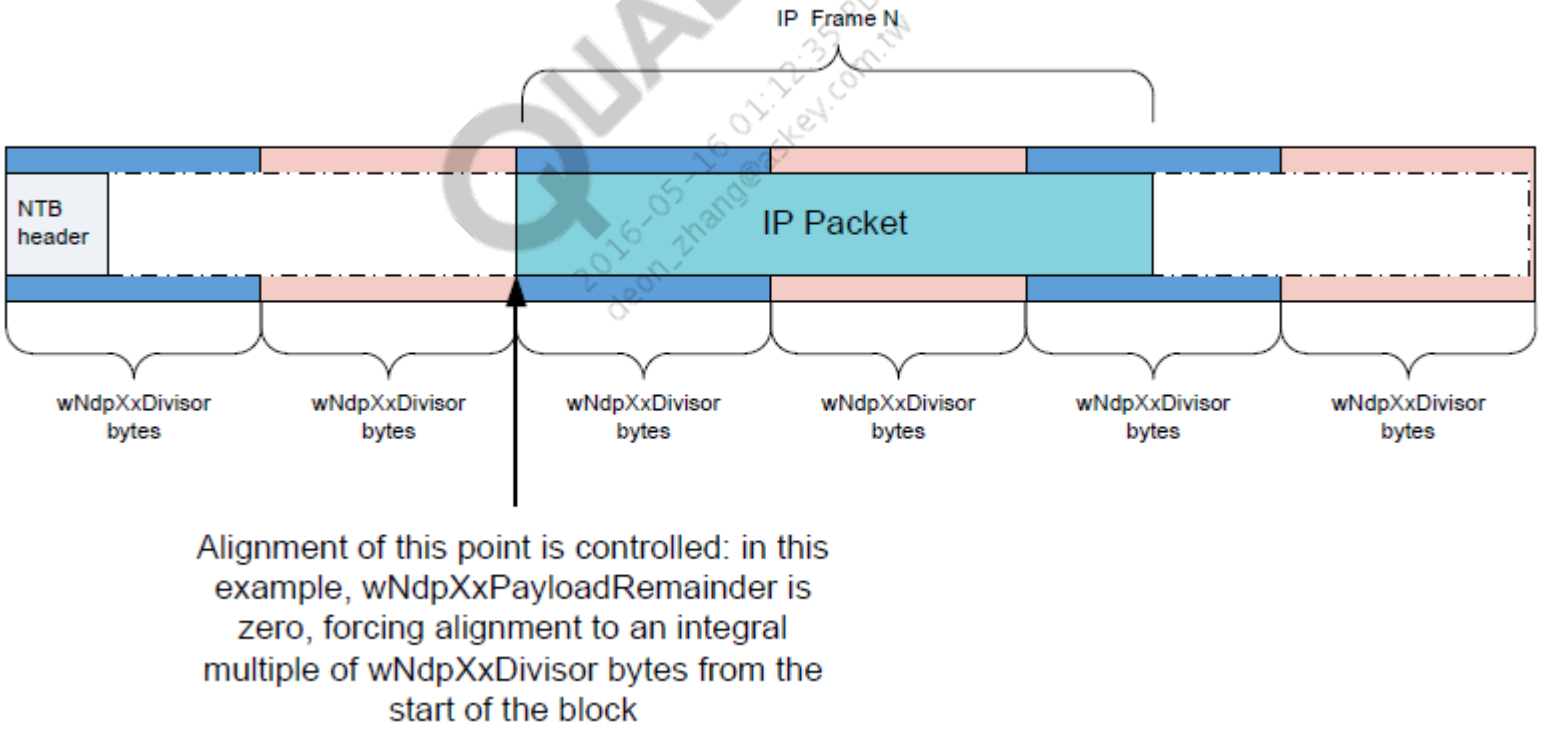


MBIM Frame Alignment

- Many network stacks in embedded devices benefit from careful alignment of the payload to system-defined memory boundaries. MBIM 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. 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.
- Alignment requirements are met by controlling the location of the payload. 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)
  - Offset % wNdpOutDivisor == wNdpOutPayloadRemainder (for OUT datagrams)

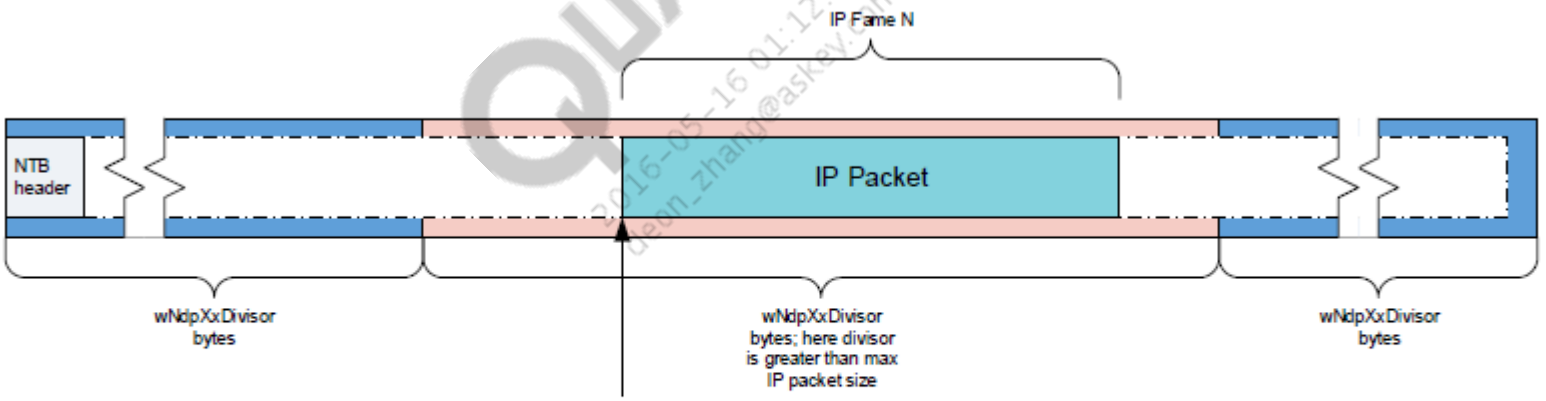
Case 1 – Alignment to a Cache Line

- 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 IP packet size. In this case, wNdpXxDivisor is set to the size of a cache line (in bytes), and wNdpXxPayloadRemainder is set to 0.



Case 2 – Alignment for Fixed-Size Internal Buffers

- 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 larger than the maximum IP packet size. In this case, wNdpXxDivisor is set to the size of the buffer, and wNdpXxPayloadRemainder is set to the intended offset of the IP packet in the fixed-size buffer.



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

Procedure to Switch Between Interfaces

- The effect of changing the communication interface alternate setting while the data interface is set to a nonzero alternate setting is not specified.
- The procedure to be followed is:
  - SET_INTERFACE (Data Interface is 0).
  - SET_INTERFACE (Communication Interface, intended alternate setting).
  - Send the required class commands for the targeted alternate setting.
  - SET_INTERFACE (Data Interface 1 if Communication Interface is 0, and Data Interface 2 if Communication Interface is 1).

Interrupt Endpoint Notifications

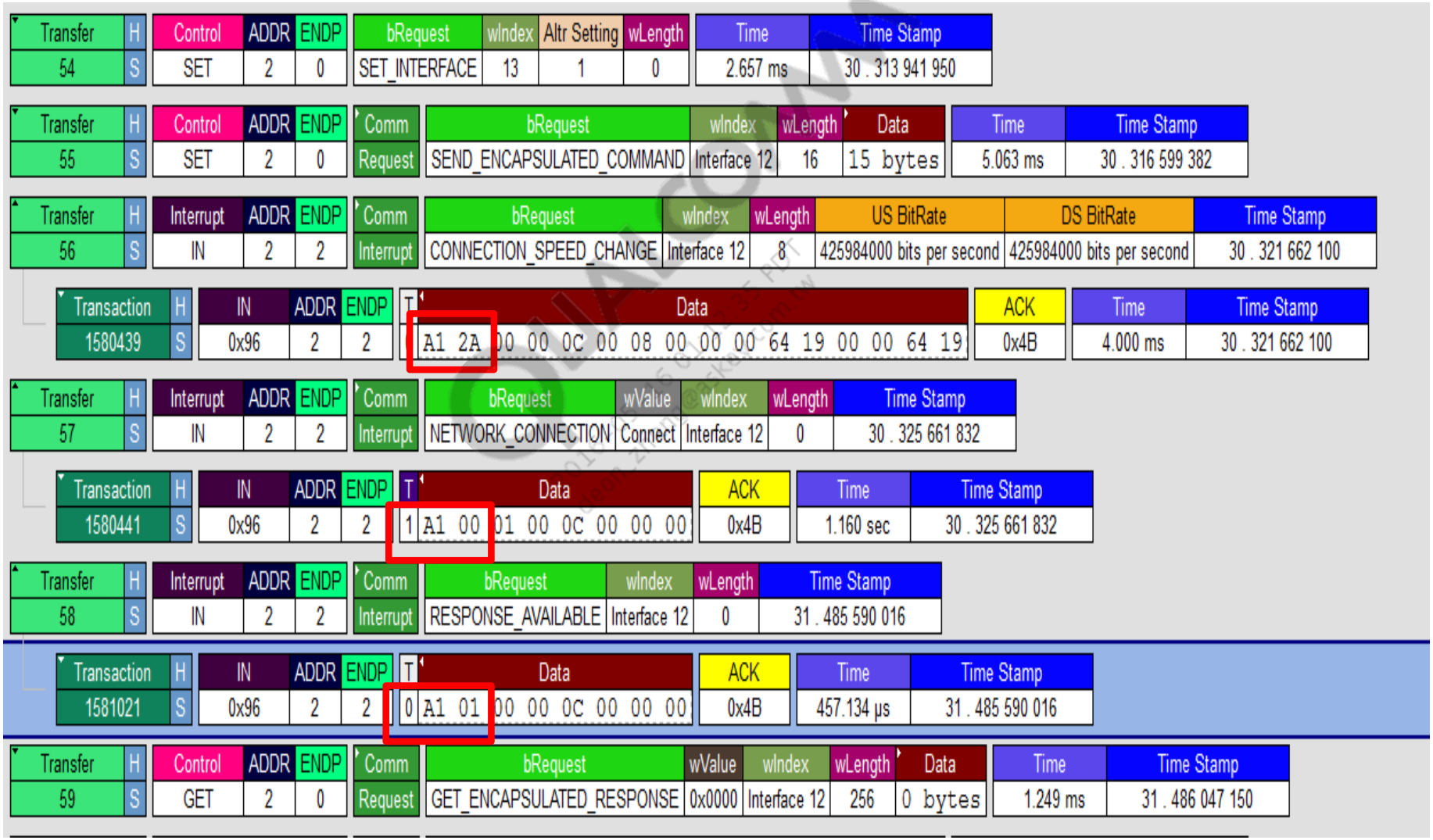
- To simplify the coding of host device drivers, functions that send a Network Connection notification with wValue == 0001h must first send a ConnectionSpeedChange notification that indicates the connection speed that is in effect when the new connection takes effect.
- This sequencing is justified as follows: if the ConnectionSpeedChange follows a NetworkConnection notification, the host driver cannot signal a 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, the host driver knows the signaling speed at the time the network connection becomes valid.

Notification	Description
NetworkConnection	Reports whether or not the physical layer (modem, Ethernet PHY, etc.) link is up.
ResponseAvailable	Notification to host to issue a *GetEncapsulatedResponse* request.
ConnectionSpeedChange	Reports a change in upstream or downstream speed of the networking connection.

Table 6-5: Class-Specific Notification Codes for Networking Control Model subclass

Request	Value
NETWORK_CONNECTION	00h
RESPONSE_AVAILABLE	01h
CONNECTION_SPEED_CHANGE	2Ah

Interrupt Endpoint Lecroy Example



Transfer	H	Control	ADDR	ENDP	bRequest	wIndex	Altr Setting	wLength	Time	Time Stamp
54	S	SET	2	0	SET_INTERFACE	13	1	0	2.657 ms	30 . 313 941 950

Transfer	H	Control	ADDR	ENDP	Comm	bRequest	wIndex	wLength	Data	Time	Time Stamp
55	S	SET	2	0	Request	SEND_ENCAPSULATED_COMMAND	Interface 12	16	15 bytes	5.063 ms	30 . 316 599 382

Transfer	H	Interrupt	ADDR	ENDP	Comm	bRequest	wIndex	wLength	US BitRate	DS BitRate	Time Stamp
56	S	IN	2	2	Interrupt	CONNECTION_SPEED_CHANGE	Interface 12	8	425984000 bits per second	425984000 bits per second	30 . 321 662 100

Transaction	H	IN	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
1580439	S	0x96	2	2		A1 2A 00 00 0C 00 08 00 00 00 64 19 00 00 64 19	0x4B	4.000 ms	30 . 321 662 100

Transfer	H	Interrupt	ADDR	ENDP	Comm	bRequest	wValue	wIndex	wLength	Time Stamp
57	S	IN	2	2	Interrupt	NETWORK_CONNECTION	Connect	Interface 12	0	30 . 325 661 832

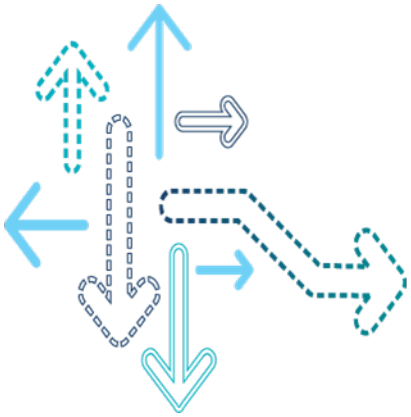
Transaction	H	IN	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
1580441	S	0x96	2	2	1	A1 00 01 00 0C 00 00 00	0x4B	1.160 sec	30 . 325 661 832

Transfer	H	Interrupt	ADDR	ENDP	Comm	bRequest	wIndex	wLength	Time Stamp
58	S	IN	2	2	Interrupt	RESPONSE_AVAILABLE	Interface 12	0	31 . 485 590 016

Transaction	H	IN	ADDR	ENDP	T	Data	ACK	Time	Time Stamp
1581021	S	0x96	2	2	0	A1 01 00 00 0C 00 00 00	0x4B	457.134 µs	31 . 485 590 016

Transfer	H	Control	ADDR	ENDP	Comm	bRequest	wValue	wIndex	wLength	Data	Time	Time Stamp
59	S	GET	2	0	Request	GET_ENCAPSULATED_RESPONSE	0x0000	Interface 12	256	0 bytes	1.249 ms	31 . 486 047 150

Overall Design



MBIM

- QTI supports MBIM through changes to existing software layers and the addition of a new layer, QBI, to process the MBIM control channel protocol.
  - Control channel – USB/MBIM Driver <==> QBI <==> QMI/Modem
  - Data channel – USB <==> A2 [MBIM-NTB] <==> RmNet/Modem

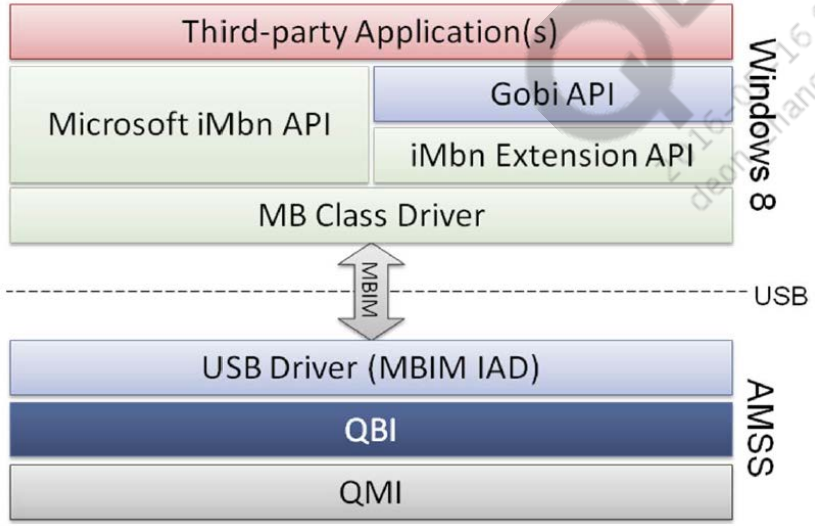
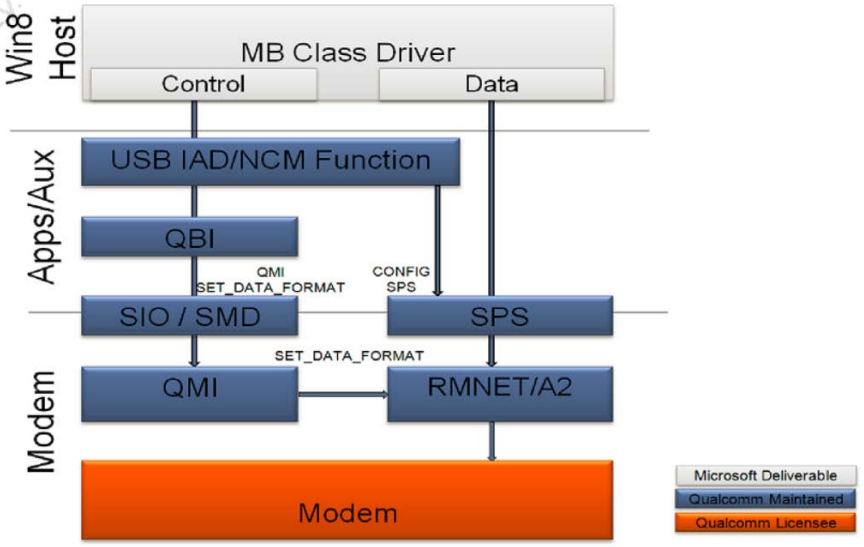


Figure 2-1 Software components involved with MBIM control channel



QBI Architecture

- QBI is a new software layer containing logic specific to processing commands (CIDs) defined in MBIM that are sent over the control channel as encapsulated messages.
- QBI runs on the same processor as the USB driver and interfaces with the modem through QMI, and translates between MBIM CIDs used to communicate with the host and QMI messages used to control the modem.
- The common messaging module is the first framework module to come into contact with an incoming CID request from the host. It parses and verifies the common parts of incoming commands, marshaling it into a more accessible internal format and passing the request on to the common device service layer for dispatch. In the outgoing direction, this layer converts between the internal format to the CID wire format.
- CID transactions are an important part of QBI. Upon receipt of a new CID request, the common messaging layer invokes the transaction allocation routine before continuing with dispatching the request.
- The common device service layer is the primary interface between the device service implementations and the rest of the QBI framework. Device services register with the common layer once at startup.

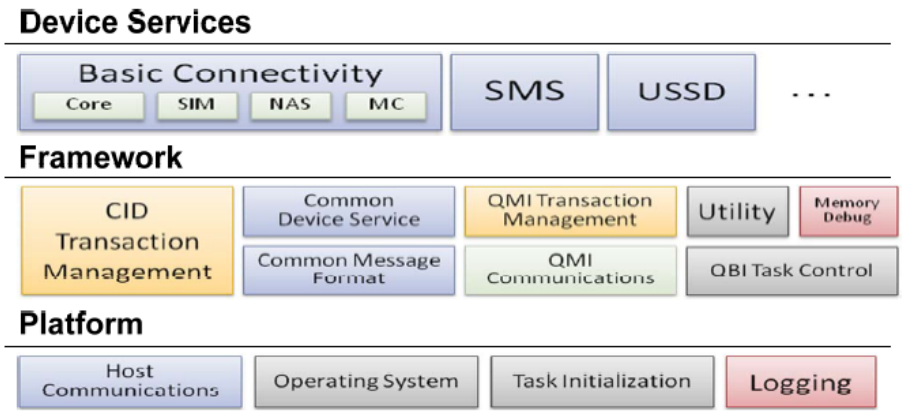
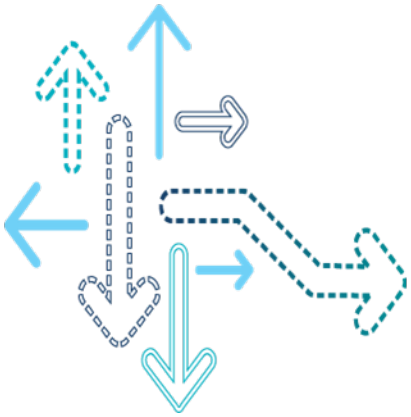
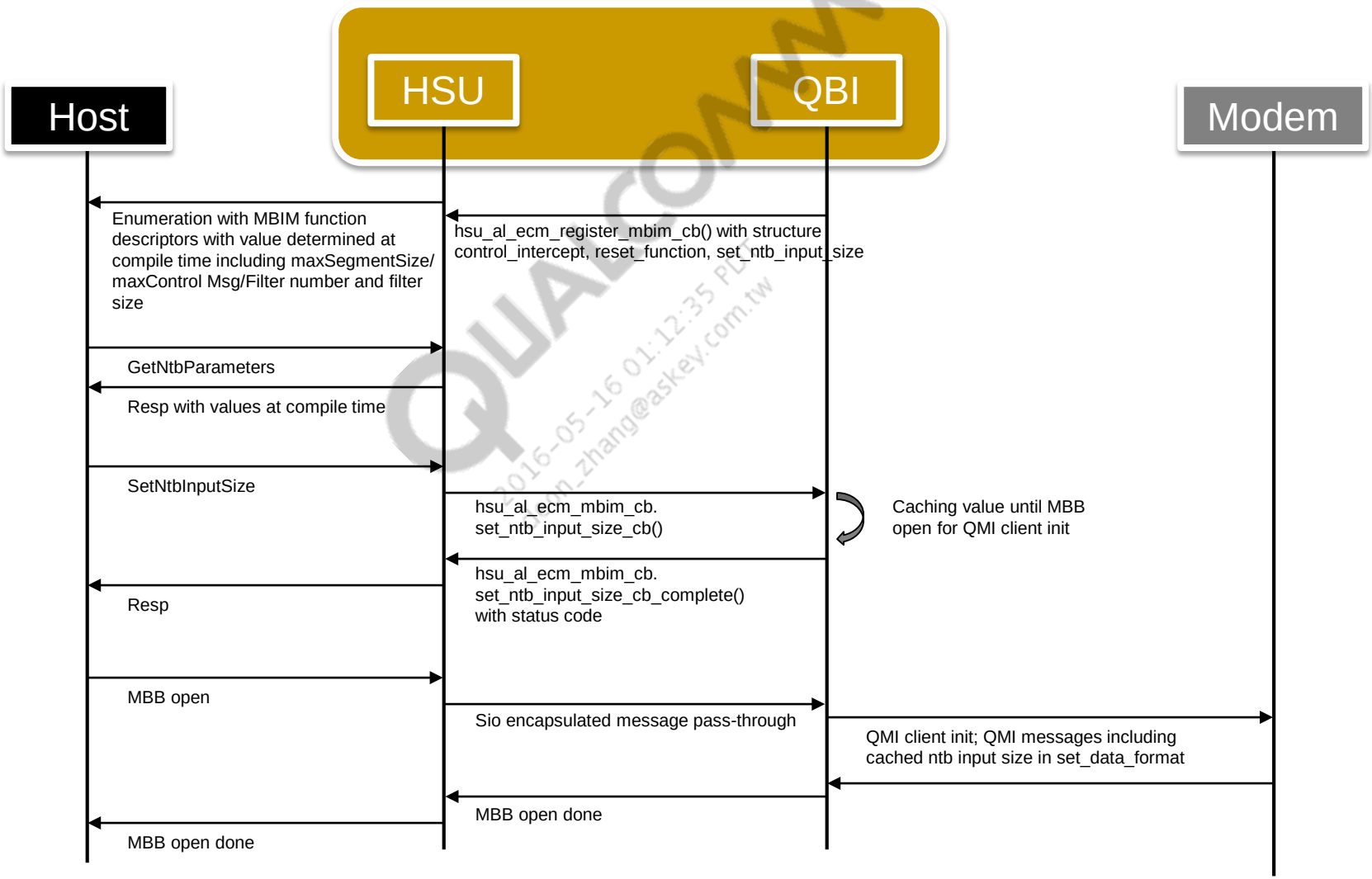


Figure 3-1 QBI module overview

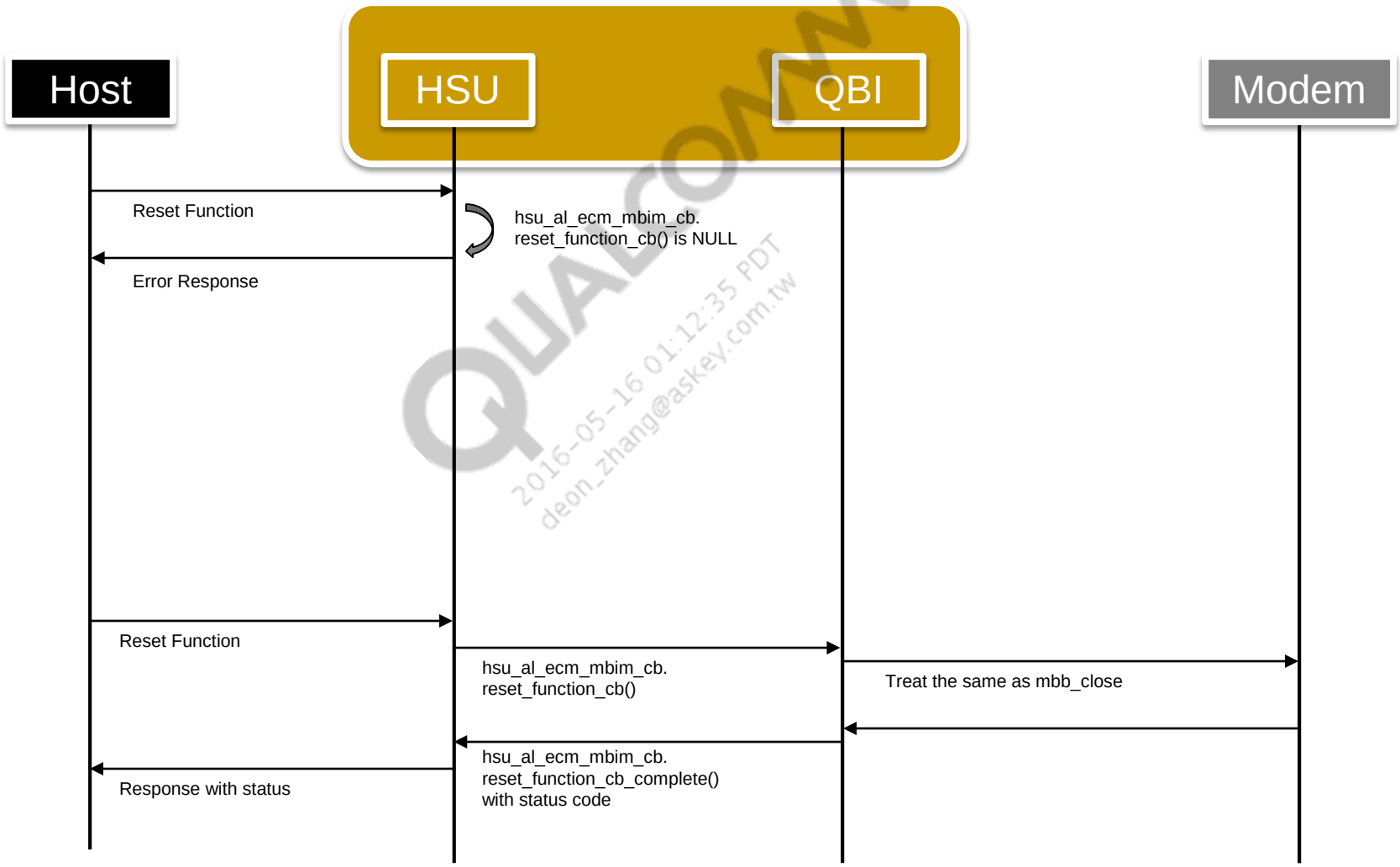
Call Flows



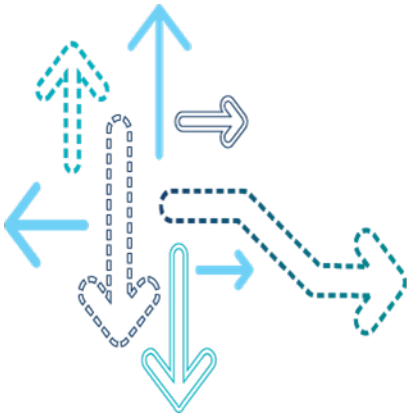
NTB Parameter Call flow



Reset Function Call flow



Device Setup

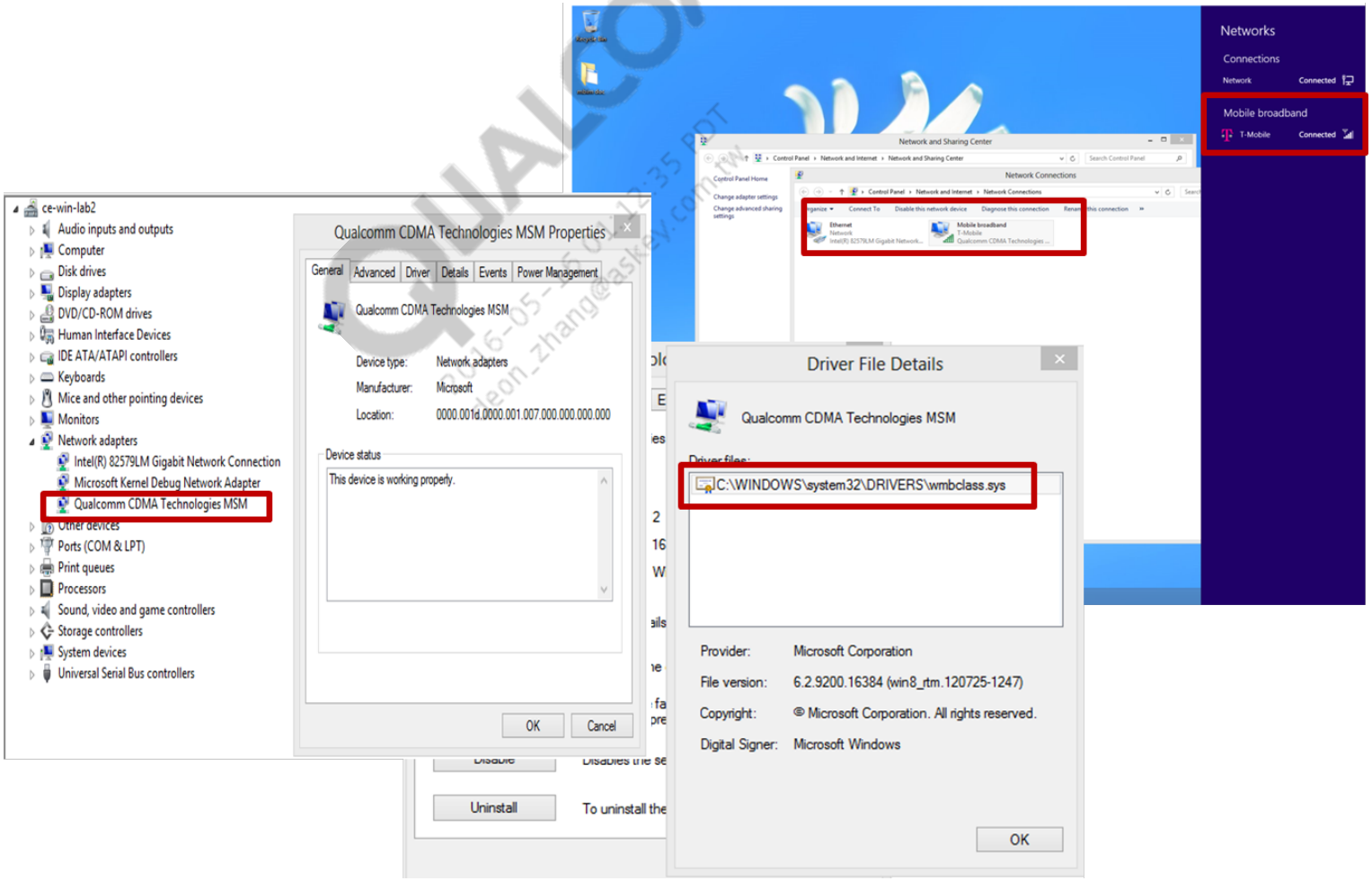


Implementation

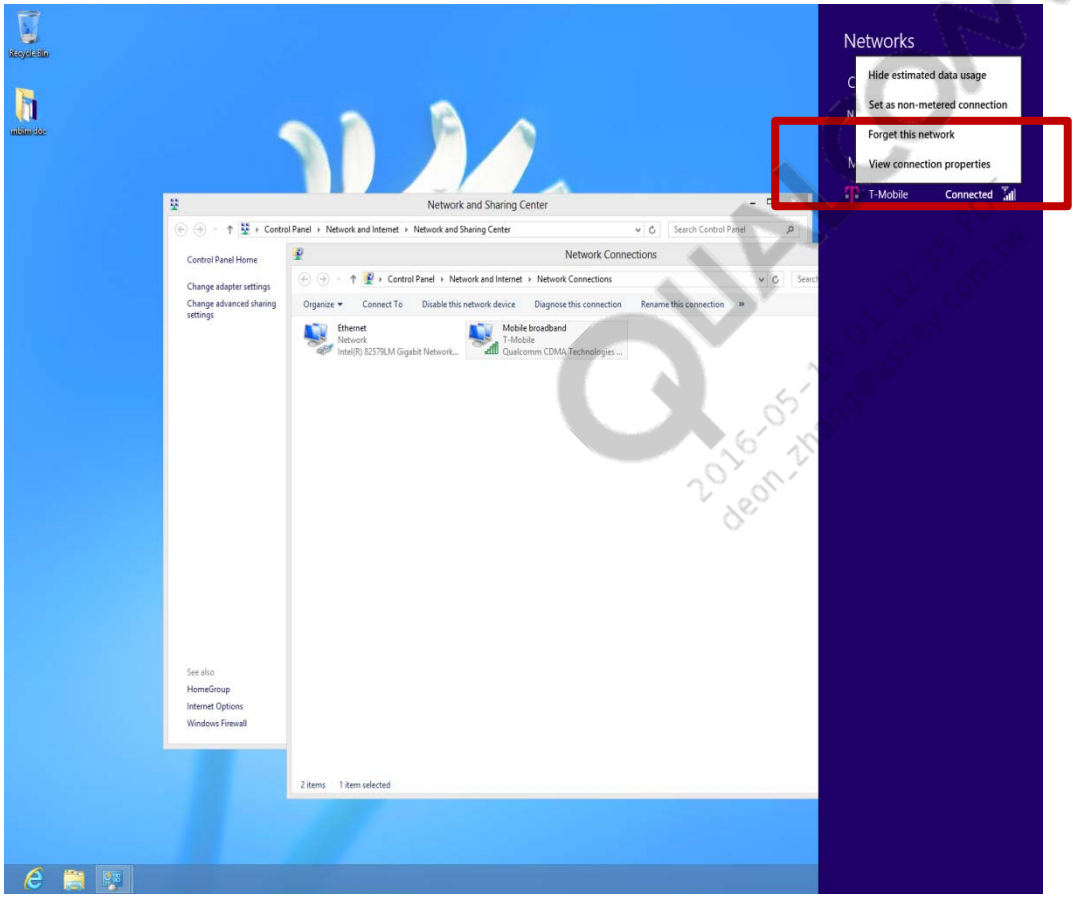
- USB MBIM driver is implemented in:
  - kernel/drivers/usb/gadget/f_mbim.c
  - kernel/drivers/usb/gadget/u_smd_ctrl.c
  - kernel/drivers/usb/gadget/u_bam_data.c
- To enable MBIM in the build, enable FEATURE_HS_USB_MBIM in custxxxxxx.h
- MBIM compositions – Several MBIM compositions are introduced
  - 0x9043 – Diag + NMEA + MDM + MBIM (AMSS)
  - 0x9058
    - Configuration 1 – MBIM (for Windows 8 host)
    - Configuration 2 – ECM (for Mac OS X and Linux hosts)
  - 0x905A
    - Configuration 1 – Diag + ADB + MBIM (for Windows 8 host)
    - Configuration 2 – ECM (for Mac OS X and Linux hosts)
  - 0x905B – MBIM
  - 0x9063 – [MDM.LE Multiple Configurations for Zero Install]
    - Configuration 1 – RNDIS (for Windows 7 and earlier versions of Windows host)
    - Configuration 2 – ECM (for Mac OS x and Linux hosts)
    - Configuration 3 – MBIM (for Windows 8 host; Microsoft OS descriptors are used to help Windows 8 pick this configuration)
  - 0x9064 – [MDM.LE Multiple Configurations for Zero Install]
    - Configuration 1 – Diag + ADB + modem + RmNet (for Win7 and earlier versions of Windows host)
    - Configuration 2 – ECM (for Mac OS x and Linux hosts)
    - Configuration 3 – MBIM (for Windows 8 host; Microsoft OS descriptors are used to help Windows 8 pick this configuration)

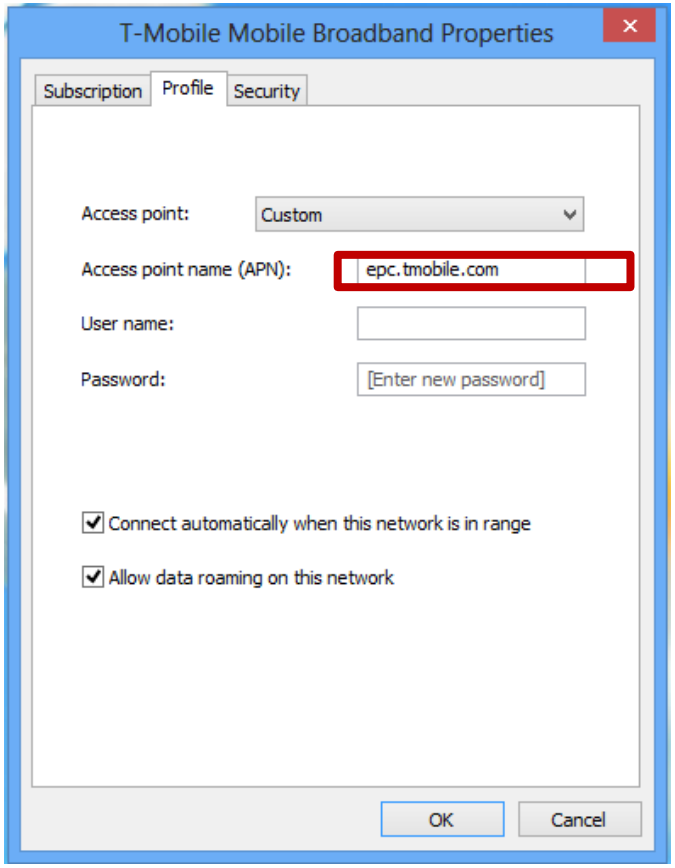
Device Setup

- After you change the NV item, reboot the device. The device appears and uses the Windows 8 MBIM driver.

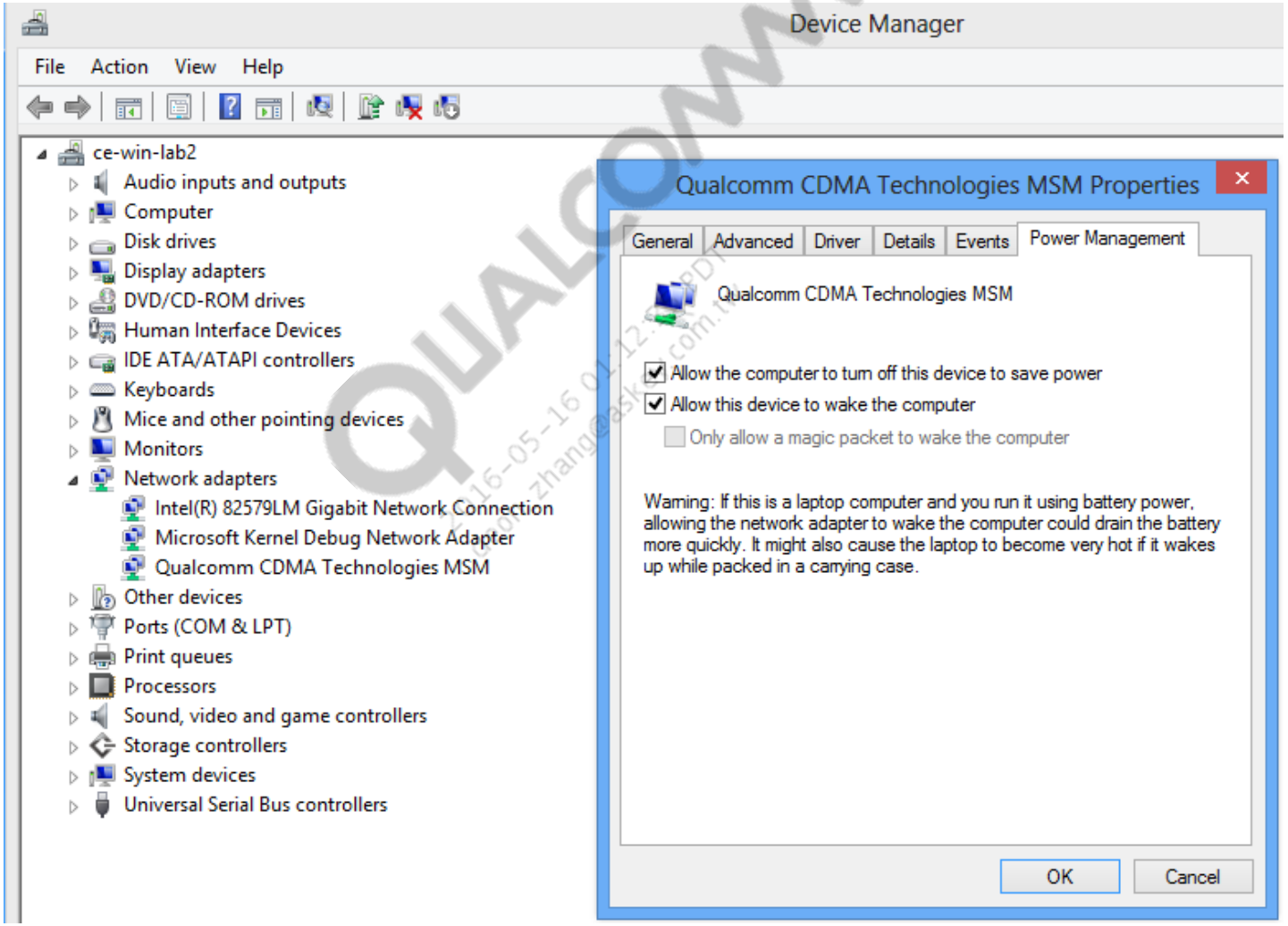


Device Setup (cont.)

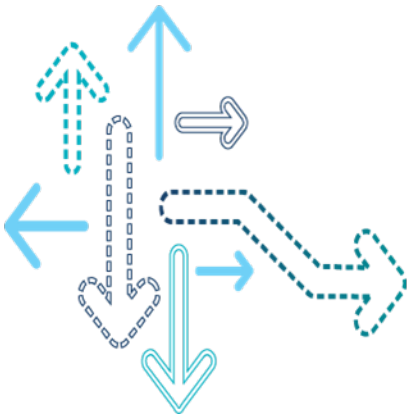




Device Setup (cont.)



Windows Hardware Certification Kit (WHCK) Testing

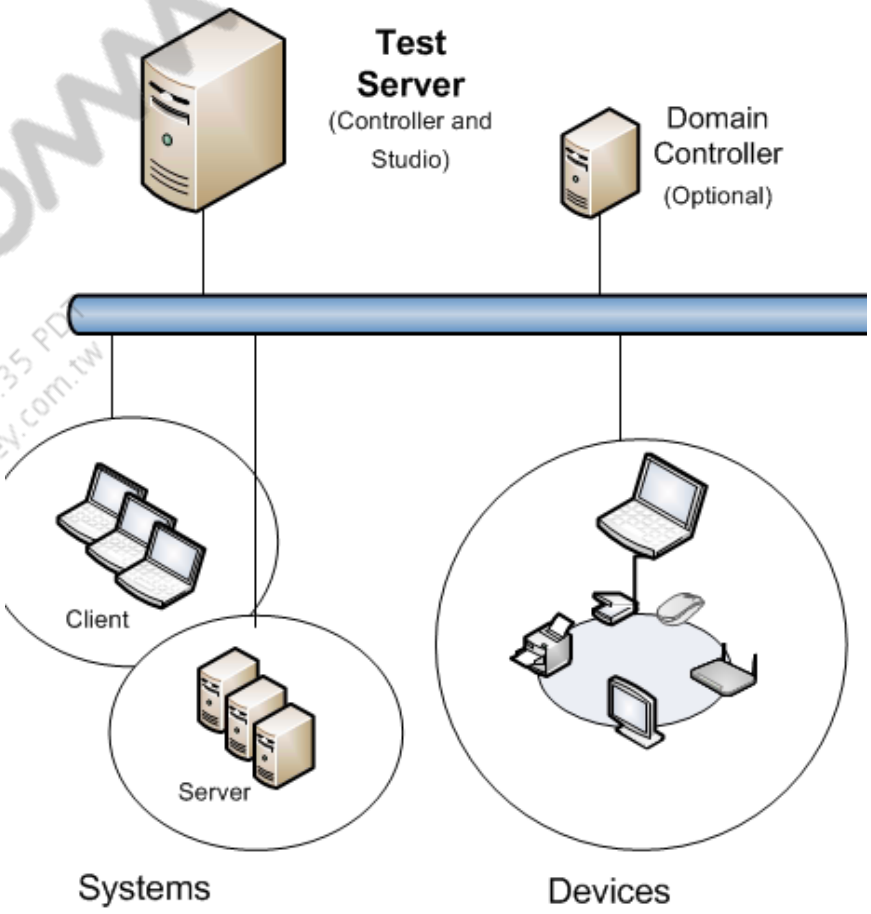


MBIM Test Procedure

- To test MBIM functionality, perform WHCK testing. Initially, during development of MBIM, numerous tests were performed on functionalities.
- It is only necessary to perform WHCK testing; the WHCK test environment is provided by Microsoft.

WHCK Test Setup

- WHCK consists of three components:
  - Controller
  - Studio
  - Client
- Install a controller on a Windows server.
- Install a controller and studio on the same computer or on different computers.
- Clients are the PC to which you attach the Device Under Test (DUT). After you install the client on the host PC, the studio is able to detect the host PC under the machine pool's root.

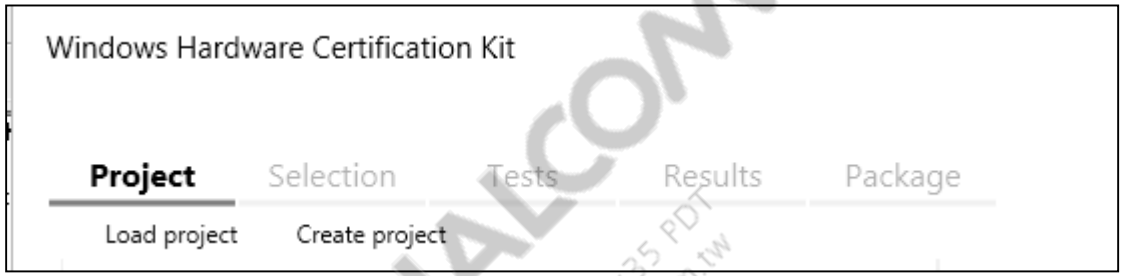


WHCK Setup Procedure

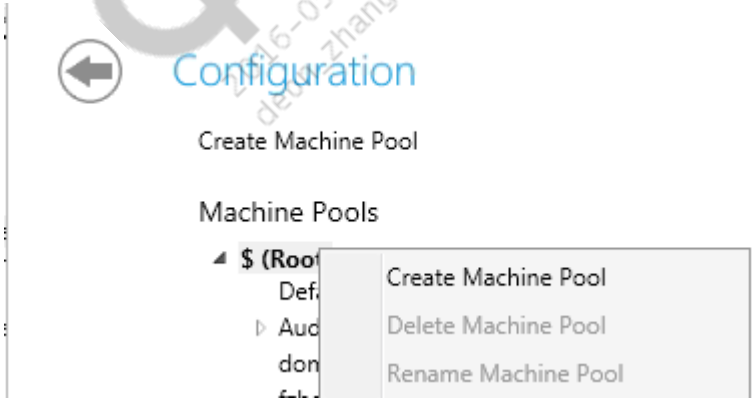
1. Install the controller and studio software.
2. Install the client software on the test computer.
3. Create a project on studio.
4. Create a machine pool.
5. Select the target to certify.
6. Select tests to run.
7. View the test results.

HCK Studio Procedure

1. Create a project.



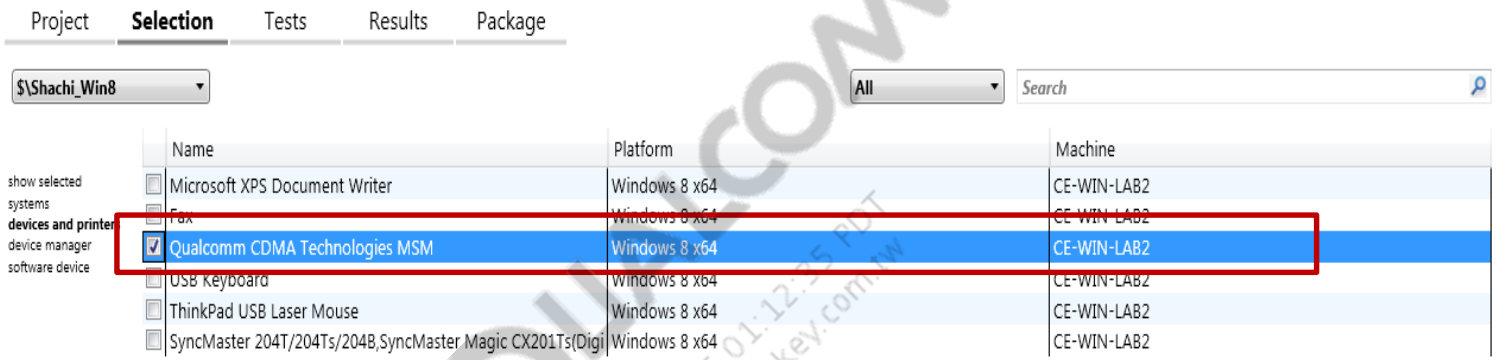
2. Create the machine pool.



3. Once the pool is created, shift your host PC into your pool.

HCK Studio

4. Select the project and then the machine pool that has QTI's host PC.



5. After the machine is selected, various devices are listed, as shown above.
6. Click **Tests** to view all the available tests for the MBIM interface.
7. Select the test to perform and run the test.
8. Click **Results** to view the test results.

MBIM WHCK Tests

		Win8.MBN.GSM.MBBDriverStress	A	01h 30m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestAuthChallenge	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestClassDriver	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestConnect	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestDeviceServices	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestDisableEnable	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestHomeProvider	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestLoopBack	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestMultiCarrier	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestPacketService	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestPin	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestPowerStates	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestPreferredProvider	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestRadioState	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestReadyInfo	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestRegisterState	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestRemoval	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSelectiveSuspend	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSetup	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSignalState	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSimBad	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSimNotInserted	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSimRoaming	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestSms	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestUssd	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestVisibleProvider	A	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2
		Win8.MBN.GSM.TestWake	M	10m	Qualcomm CDMA Technologies M	CE-WIN-LAB2

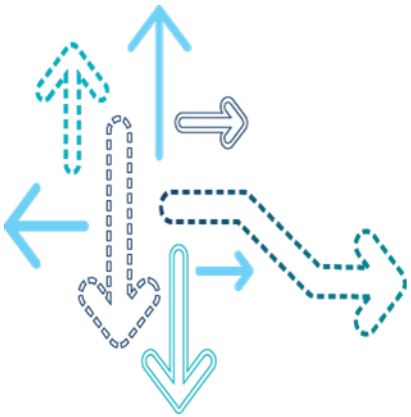
WHCK Test Settings

- To perform some of tests, the following settings may be required if these errors are received:
  1. Failure – Platform does not support always-on, always-connected
     Resolution – This test requires a platform (laptop or tablet) that supports Connected Standby (CS), also known as Always On Always Connected (AOAC). CS is a new feature for Windows 8, so a commercial x86 laptop designed for Windows 7 does not support the feature. Therefore, this test must be run with a platform that was explicitly designed for Windows 8.
  2. Failure – Attempt to query SMS configuration failed/attempt to query SMS status failed
     Resolution – Check the SMS configuration NV 0x830
  3. Failure – Attempt to find provisioned context of WwanContextTypeInternet failed
     Resolution – Change the contexttype as internet
  4. Failure – Data loss (5.841678%) is more than 5%
     Resolution – Use an I7 or faster PC to run this test
  5. Failure – Unsolicited NDIS_STATUS_WWAN_SMS_STATUS indication was not received
     Resolution – This test requires a second device to manually send an SMS to the device under test; see also TestSms failures

WHCK Test Settings (cont.)

- To perform some of tests, the following settings may be required if these errors are received: (cont.)
  6. TestSms
     Note that NV 69707 should be set to 1. This NV item enables saving SMS messages to the SIM once NV is full.
  7. Failure – Attempt to send message to self, failed

MBIM Product Line (PL) Support and CR/FR



PLs Supported

- QTI does not support MBIM on the MDM9x15 TN build; MBIM is only supported on PL 2.7 on the MDM9x15.
- It is also supported on the MDM9x00 build.
- Fused architecture builds do not support MBIM.
- MBIM support is available in CRM APSS builds starting with M9615AAAARNLZA10026300.
- MBIM is fully supported in CRM meta build M9625AAATWNLBD102028.1 and later builds. Some earlier builds might work too, but are not recommended because of known issues with the data path. MDM9x25 TN does *not* support MBIM.
- MBIM is supported in CRM meta build M9635AAATWNLBD100525.1 but with known issues related to the MBIM open and data path. MDM9x35 TN does *not* support MBIM.
- MDM9x15 TN 4.0 and TN 5.0 support MBIM.

MBIM CR for New Features

- Upcoming Windows 8.1 releases require this new extended functional header immediately after the MBIM functional header.
  - Described in Section 6.5 of *MBIM Extended Functional Descriptor of USB MBIM 1.0 Errata-1*
    - [CR428834]
    - [CR432633]
- Currently, QTI is using IAD to group interfaces; however, per the spec, a UNION descriptor should be used.
- *MBIM Specification 0.3* only specified IAD and did not specify UNION descriptors.
- *MBIM Specification 0.81* added UNION but specified that devices must implement IAD or UNION.
- In the final version of *MBIM Extended Functional Descriptor of USB MBIM 1.0 Errata-1*, Section 6.1 specifies the following:
  - There are two ways to group interfaces, the WHCM Union Functional Descriptor (see [USBWMC11]) and the IAD. Devices may also provide an IAD. If an IAD is provided, the information in the IAD for MBIM functions shall be consistent with the information in the CDC Union descriptor and Communication Class interface descriptor.
  - Therefore, there are CRs to add this feature.
    - [CR441355] – [MDM9x15 TN MBIM] Union Functional Descriptor
    - [CR441117] – [MDM9x15 LE MBIM] Union Functional Descriptor

References

Documents	
Qualcomm Technologies, Inc.	
Qualcomm Mobile Broadband Interface (QBI) Design Overview	80-N8983-1
Standards	
MBIM Extended Functional Descriptor of USB MBIM 1.0 Errata-1	Open Source
MBIM Specification 0.3	Open Source
MBIM Specification 0.81	Open Source
NCM Specification 1.0	Open Source
Universal Serial Bus Communication Class MBIM Compliance Testing Revision 1.0	Open Source
MBIM Specification 1.0	Open Source

References (cont.)

Acronyms	
Term	**Definition**
ADSL	Asymmetrical Digital Subscriber Line
AOAC	Always On Always Connected
DUT	Device Under Test
ECM	Ethernet Control Model
EEM	Ethernet Emulation Model
IAD	Interface Association Descriptor
MBIM	Mobile Broadband Interface Model
MTU	Message Transfer Unit
NCM	Network Control Model
NDP	NCM Datagram Pointer
NTP	NCM Transfer
NTB	NCM Transfer Block
NTH	NCM Transfer Header
PL	Product Line
UUID	Universally Unique Identifier
WHCK	Windows Hardware Certification Kit
ZLP	Zero Length Packet

Questions?

https://support.cdmatech.com