Universal Serial Bus Communication Class MBIM Compliance Testing

Revision 1.0

February 7, 2013

Revision History

Rev	Date	Filename	Comments
1.0	2013-02-07	MBIM-Compliance-1.0.docx	First Published version.

Please send comments via electronic mail to ncm-chair@usb.org

Contributors

Patrik Olesen	Ericsson
Michal Jablonski	Intel
Ravi Darsi	Intel
Ulrich Leucht-Roth	Intel
Ygal Blum	Jungo
Yoav Nissim	Jungo
Chris Yokum	MCCI Corporation
Greg Scaffidi	MCCI Corporation
Gunjan Saxena	MCCI Corporation
Prabu	MCCI Corporation
Subbarayalu	MCCI Corporation
Sunil kumar	MCCI Corporation
Thirumalai Rajan	MCCI Corporation
Christopher Gual	Microsoft
Gabriel Montenegro	Microsoft
Nazan Kurt	Microsoft
Srinivasan Malayala	Microsoft
Eugene Grosbein	Nokia
Richard Pettri	Nokia
Roman Pak	Nokia
Vladimir Semenyuk	Smith Micro Software
Alexey Orishko	ST-E
Morten Christiansen	ST-E

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide assertions and tests for validating devices which indicate [USBMBIM10] support in one or more functions of the device. This specification does not add requirements to the [USBMBIM10], [USBCDC12], [USB20] or [USB30] specifications. Basic interoperability and compliance can be tested using only the information in those specifications. This document is prepared using [USBMBIM10] as base document.

1.2 Scope

This specification provides assertions and test designs for devices with functions implementing [USBMBIM10]. These assertions and test designs allow limited validation of proper implementation of [USBMBIM10] by the device.

If there are conflicts between this specification document and [USBMBIM10] then [USBMBIM10] shall be taken to be the controlling document.

1.3 Related Documents

[USBNCM10] Universal Serial Bus Communications Class Subclass Specifications for Network Control Model Devices, Revision 1.0 as modified by the errata. http://www.usb.org.

[USBMBIM10] Universal Serial Bus Communications Class subclass Specification for Mobile Broadband Interface Model, Revision 1.0 – Errata 1. http://www.usb.org.

[USB20] Universal Serial Bus Specification, Revision 2.0. http://www.usb.org.

[USB30] Universal Serial Bus 3.0 Specification, Revision 1.0, November 12, 2008. http://www.usb.org. Unless otherwise specified, any reference to [USB30] includes [USB20] by reference, especially when referring to full- and high-speed devices.

[USBCDC12] Universal Serial Bus Class Definitions for Communications Devices, Revision 1.2. http://www.usb.org.

[USBECM12] Universal Serial Bus Class Definitions for Ethernet Control Model Devices, Revision 1.2. http://www.usb.org.

1.4 Abbreviations

Abbreviation	Explanation
CID_XX	Control command validation test
CM_XX	Control message validation test
COA	Checklist Only Assertion: an assertion marked as COA has no corresponding test sequence; it SHALL be validated using the COA checklist
CREQ_XX	Control request validation test
DES_XX	Descriptors validation test
DTS_XX	Data transfer validation test
ERR_XX	Error handling validation test
M	Mandatory: an assertion marked as Mandatory MUST be implemented by the Compliance Tool
NCM/MBIM	Function that supports NCM 1.0 and MBIM interfaces

2 Management Overview

MBIM functions require compliance to several standards and specifications.

IP layer: Depending on the type of USB being used, one or more of the following may apply.

- [USB20] for full- and high-speed operation
- [USB30] for super-speed operation

MBIM layer: All of the following specifications apply:

- [USBCDC12] as the general framework for all CDC subclasses
- [USBECM12] for certain descriptor, management element,
- [USBMBIM10] for notification formats.

This document focuses on testing the implementation of the MBIM 1.0 layer.

All tests below refer only to a single MBIM function even if multiple MBIM functions are present. In case of multiple MBIM functions the tests should be applied in sequence.

Please note that all values should be in little-endian, unless otherwise specified.

3 Test Assertions

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 3.2.1#1	Functions that implement both NCM 1.0 and MBIM shall provide two alternate settings for the Communication Interface.	DES_01	M
[MBIM 1.0] - 3.2.1#2	For alternate setting 0 of the Communication Interface of an NCM/MBIM function: interface, functional and endpoint descriptors shall be constructed according to the rules given in [USBNCM10].	DES_01	M
[MBIM 1.0] - 3.2.1#3	For alternate setting 1 of the Communication Interface of an NCM/MBIM function: interface, functional and endpoint descriptors shall be constructed according to the rules given in [MBIM1.0] section 6.	DES_01	M
[MBIM 1.0] - 3.2.1#4	When alternate setting 0 of the Communication Interface of an NCM/MBIM function is selected, the function shall operate according to the NCM rules given in [USBNCM10]. In particular, NTBs shall transport Ethernet frames, not IP datagrams.		COA
[MBIM 1.0] - 3.2.1#5	When alternate setting 1 of the Communication Interface of an NCM/MBIM function is selected, the function shall operate according to the MBIM rules given in [USBMBIM10]. In particular, NTBs shall transport IP datagrams, not Ethernet frames	DTS_01	M
[MBIM 1.0] - 3.2.2.1#1	If an Interface Association Descriptor is used to form an NCM/MBIM function, its interface class, subclass, and protocol codes shall match those given in alternate setting 0 of the Communication Interface.	DES_01	M
[MBIM 1.0] - 3.2.2.2#1	For an NCM/MBIM function the Communication Interface descriptor for alternate setting 0 must have bInterfaceSubClass == 0Dh and bInterfaceProtocol == XXh.	DES_01	M
[MBIM 1.0] - 3.2.2.3#1	For an NCM/MBIM function, alternate setting 0 of the Communication Interface shall be followed by alternate setting 1.	DES_01	M
[MBIM 1.0] - 3.2.2.3#2	For an NCM/MBIM function the Communication Interface descriptor for alternate setting 1 must have bInterfaceSubClass == 0Eh, and bInterfaceProtocol == 00h.	DES_01	M
[MBIM 1.0] - 3.2.2.4#1	Functions that implement both NCM 1.0 and MBIM (an “NCM/MBIM function”) shall provide three alternate settings for the Data Interface.	DES_01	M
[MBIM 1.0] - 3.2.2.4#2	For an NCM/MBIM function the Data Interface descriptors for alternate settings 0 and 1 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 01h.	DES_01	M
[MBIM 1.0] - 3.2.2.4#3	For an NCM/MBIM function the Data Interface descriptor for alternate setting 2 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 02h.	DES_01	M
[MBIM 1.0] - 3.2.2.4#4	For an NCM/MBIM function there must be no endpoints for alternate setting 0 of the Data Interface. For each of the other two alternate settings (1 and 2) there must be exactly two endpoints: one Bulk IN and one Bulk OUT.	DES_01	M
[MBIM 1.0] - 5.2.3#1	If the transfer is less than the configured Max NTB size and is multiple of the wMaxPacketSize the function must terminate the transfer with a ZLP.		COA

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 6.1#1	If an Interface Association Descriptor (IAD) is provided for the MBIM function, the IAD and the mandatory CDC Union Functional Descriptor specified for the MBIM function shall group together the same interfaces.	DES_02	M
[MBIM 1.0] - 6.1#2	If an Interface Association Descriptor (IAD) is provided for the MBIM only function, its interface class, subclass, and protocol codes shall match those given in the Communication Interface descriptor.	DES_02	M
[MBIM 1.0] - 6.3#1	The descriptor for alternate setting 0 of the Communication Interface of an MBIM only function shall have bInterfaceClass == 02h, bInterfaceSubClass == 0Eh, and bInterfaceProtocol == 00h.	DES_02	M
[MBIM 1.0] - 6.3#2	MBIM Communication Interface description shall include the following functional descriptors: • CDC Header Functional Descriptor • CDC Union Functional Descriptor • MBIM Functional Descriptor Refer to Table 6.2 of [USBMBIM10].	DES_01, DES_02	M
[MBIM 1.0] - 6.3#3	CDC Header Functional Descriptor shall appear before CDC Union Functional Descriptor and before MBIM Functional Descriptor.	DES_01, DES_02	M
[MBIM 1.0] - 6.3#4	CDC Union Functional Descriptor for an MBIM function shall group together the MBIM Communication Interface and the MBIM Data Interface.	DES_01, DES_02	M
[MBIM 1.0] - 6.3#5	The class-specific descriptors must be followed by an Interrupt IN endpoint descriptor.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#1	Field wMaxControlMessage of MBIM Functional Descriptor must not be smaller than 64.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#2	Field bNumberFilters of MBIM Functional Descriptor must not be smaller than 16.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#3	Field bMaxFilterSize of MBIM Functional Descriptor must not exceed 192.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#4	Field wMaxSegmentSize of MBIM Functional Descriptor must not be smaller than 2048.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#5	Field bFunctionLength of MBIM Functional Descriptor must be 12 representing the size of the descriptor.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#6	Field bcdMBIMVersion of MBIM Functional Descriptor must be 0x0100 in little endian format.	DES_01, DES_02	M
[MBIM 1.0] - 6.4#7	Field bmNetworkCapabilities of MBIM Functional Descriptor should have the following bits set to zero: D0, D1, D2, D4, D6 and D7.	DES_01, DES_02	M
[MBIM 1.0] - 6.5#1	If MBIM Extended Functional Descriptor is provided, it must appear after MBIM Functional Descriptor.	DES_01, DES_02	M
[MBIM 1.0] - 6.5#2	Field bFunctionLength of MBIM Extended Functional Descriptor must be 8 representing the size of the descriptor.	DES_01, DES_02	M
[MBIM 1.0] - 6.5#3	Field bcdMBIMEFDVersion of MBIM Extended Functional Descriptor must be 0x0100 in little endian format.	DES_01, DES_02	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 6.5#4	Field bMaxOutstandingCommandMessages of MBIM Extended Functional Descriptor shall be greater than 0.	DES_01, DES_02	M
[MBIM 1.0] - 6.6#1	The Data Interface for an MBIM only function shall provide two alternate settings.	DES_02	M
[MBIM 1.0] - 6.6#2	The first alternate setting for the Data Interface of an MBIM only function (the default interface setting, alternate setting 0) shall include no endpoints.	DES_02	M
[MBIM 1.0] - 6.6#3	The second alternate setting for the Data Interface of an MBIM only function (alternate setting 1) is used for normal operation, and shall include one Bulk IN endpoint and one Bulk OUT endpoint.	DES_02	M
[MBIM 1.0] - 6.6#4	For an MBIM only function the Data Interface descriptors for alternate settings 0 and 1 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 02h. Refer to Table 6.4 of [MBIM1.0].	DES_02	M
[NCM 1.0] - 3.2.1#1	The first four bytes in NTH16 shall be 0x484D434E in little-endian format (“NCMH”).	DTS_02	M
[NCM 1.0] - 3.2.1#2	wHeaderLength value in NTH16 shall be 0x000C.	DTS_03	M
[NCM 1.0] - 3.2.1#3	wSequence in NTH16 shall be set to zero by the function in the first NTB transferred after every “function reset” event.	DTS_04	M
[NCM 1.0] - 3.2.1#4	wSequence value in NTH16 shall be incremented for every NTB subsequent transfer.	DTS_05	M
[NCM 1.0] - 3.2.1#5	NTB size (IN) shall not exceed dwNtbInMaxSize.	DTS_06	M
[NCM 1.0] - 3.2.1#6	wNdpIndex value in NTH16 must be a multiple of 4, and must be >= 0x000C, in little endian.	DTS_07	M
[NCM 1.0] - 3.2.1#7	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.		COA
[NCM 1.0] - 3.2.2#1	The first four bytes in NTH32 shall be 0x686D636E in little-endian format (“ncmh”).	DTS_08	M
[NCM 1.0] - 3.2.2#2	wHeaderLength value in NTH32 shall be 0x0010.	DTS_09	M
[NCM 1.0] - 3.2.2#3	wSequence in NTH32 shall be set to zero by the function in the first NTB transferred after every “function reset” event.	DTS_10	M
[NCM 1.0] - 3.2.2#4	wSequence value in NTH32 shall be incremented for every NTB subsequent transfer.	DTS_11	M
[NCM 1.0] - 3.2.2#5	NTB size (IN) shall not exceed dwNtbInMaxSize.	DTS_12	M
[NCM 1.0] - 3.2.2#6	dwNdpIndex value in NTH32 must be a multiple of 4, and must be >= 0x0010.	DTS_13	M
[NCM 1.0] - 3.2.2#7	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.		COA

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 7#1	To distinguish among the data streams, the last character of the dwSignature in the NDP16 header shall be coded with the index SessionId specified by the host in the MBIM_CID_CONNECT. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “IPS”<SessionId>.	DTS_14	M
[MBIM 1.0] - 7#2	To distinguish among the data streams, the last character of the dwSignature in the NDP16 header shall be coded with the DssSessionId specified by the host in the MBIM_CID_DSS_CONNECT command. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “DSS”<DssSessionId>.		COA
[MBIM 1.0] - 7#3	To distinguish among the data streams, the last character of the dwSignature in the NDP32 header shall be coded with the SessionId specified by the host in the MBIM_CID_ CONNECT. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “ips”<SessionId>.	DTS_20	M
[MBIM 1.0] - 7#4	To distinguish among the data streams, the last character of the dwSignature in the NDP32 header shall be coded with the DssSessionId specified by the host in the MBIM_CID_DSS_CONNECT command. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “dss”<DssSessionId>.		COA
[NCM 1.0] - 3.3.1#1	wLength value in NDP16 must be a multiple of 4, and must be at least 16d (0x0010).	DTS_15	M
[NCM 1.0] - 3.3.1#2	wDatagramIndex[0] value in NDP16 must be >= 0x000C (because it must point past the NTH16).	DTS_16	M
[NCM 1.0] - 3.3.1#3	wDatagramLength[0] value in NDP16 must be >= 20d if datagram payload is IPv4 and >= 40d if datagram payload is IPv6.	DTS_17	M
[NCM 1.0] - 3.3.1#4	wDatagramIndex[(wLength-8)/4 - 1] value in NDP16 must be zero.	DTS_18	M
[NCM 1.0] - 3.3.1#5	wDatagramLength[(wLength-8)/4 - 1] value in NDP16 must be zero.	DTS_19	M
[NCM 1.0] - 3.3.2#1	wLength value in NDP32 must be a multiple of 8, and must be at least 32d (0x0020).	DTS_21	M
[NCM 1.0] - 3.3.2#2	dwDatagramIndex[0] value in NDP32 must be >= 0x0010 (because it must point past the NTH32).	DTS_22	M
[NCM 1.0] - 3.3.2#3	dwDatagramLength[0] value in NDP32 must be >= 20d if datagram payload is IPv4 and >= 40d if datagram payload is IPv6.	DTS_23	M
[NCM 1.0] - 3.3.2#4	dwDatagramIndex[(wLength-8)/8 - 1] value of NDP32 must be zero.	DTS_24	M
[NCM 1.0] - 3.3.2#5	dwDatagramLength[(wLength-8)/8 - 1] value of NDP32 must be zero.	DTS_25	M
[NCM 1.0] - 3.3.4	The agent formatting a given NTB aligns the payload of each datagram by inserting padding, such that the offset of each datagram payload satisfies the constraint: Offset % wNdpInDivisor == wNdpInPayloadRemainder (for IN datagrams).	DTS_26	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[NCM 1.0] - 3.4	Functions shall not send NTBs larger than the host has requested.		COA
[NCM 1.0] - 3.7#1	The first Null Datagram pointer entry in the NTB shall be interpreted as meaning that all following NCM Datagram Pointer Entries in the NDP are to be ignored.	DTS_27	M
[NCM 1.0] - 3.7#2	Transmitters are allowed to send a properly-formatted NTB containing an NDP whose datagram pointer entries are all zero. Receivers shall ignore such NTBs		COA
[MBIM 1.0] - 8.1#1	The following requests must be supported by MBIM function: • SendEncapsulatedCommand() • GetEncapsulatedResponse() • GetNtbParameters() • SetNtbInputSize() • GetNtbInputSize() • ResetFunction()	CREQ_01	M
[MBIM 1.0] - 8.1.2#1	When the MBIM function is ready to send a control message to the host, the function must return a RESPONSE_AVAILABLE notification on the Communication Class interface's Interrupt IN endpoint.		COA
[MBIM 1.0] - 8.1.2#2	The function must use a separate GET_ENCAPSULATED_RESPONSE transfer for each control message it has to send to the host.	CM_05	M
[MBIM 1.0] - 8.1.2#3	The function must send a RESPONSE_AVAILABLE notification for each available fragment of ENCAPSULATED_RESPONSE to be read from the default pipe.	CM_15	M
[MBIM 1.0] - 8.1.2#4	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.		COA
[MBIM 1.0] - 8.1.5	In case of RESET_FUNCTION, the function shall abandon all outstanding transactions that are awaiting completion. No notifications shall be sent.		COA
[MBIM 1.0] - 9.1#1	For notifications, the TransactionId must be set to 0 by the function.	CM_09	M
[MBIM 1.0] - 9.1#2	MessageLength in MBIM_MESSAGE_HEADER must be >= 0x0C	CM_02	M
[MBIM 1.0] - 9.2	Function should fragment responses based on MaxControlTransfer value from MBIM_OPEN_MSG.	CM_15	M
[MBIM 1.0] - 9.3.1#1	In case MBIM_OPEN_MSG message is sent to a function that is already opened, the function shall interpret this as that the host and the function are out of synchronization. The function shall then perform the actions dictated by the MBIM_CLOSE_MSG before it performs the actions dictated by this command. The function shall not send the MBIM_CLOSE_DONE when the transition to the Closed state has been completed. Only the MBIM_OPEN_DONE message is sent upon successful completion of this message.	CM_03	M
[MBIM 1.0] - 9.3.2#1	Between the host's sending MBIM_CLOSE_MSG message and the function's completing the request (acknowledged with MBIM_CLOSE_DONE), the function shall ignore any MBIM control messages it receives on the control plane or the data on the bulk pipes.	CM_11	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 9.3.2#2	The function shall not send any MBIM control messages on the control plane or data on the bulk pipes after completing MBIM_CLOSE_MSG message (acknowledging it with the MBIM_CLOSE_DONE message) with one exception and that is MBIM_ERROR_NOT_OPENED.	CM_12	M
[MBIM 1.0] - 9.3.2#3	On MBIM_CLOSE_MSG, any active context between the function and the host shall be terminated.	CM_13	M
[MBIM 1.0] - 9.3.4#1	An MBIM_FUNCTION_ERROR_MSG shall not be sent in response to an MBIM_HOST_ERROR_MSG.		COA
[MBIM 1.0] - 9.3.4#2	An MBIM_FUNCTION_ERROR_MSG shall not make use of a DataBuffer, so it cannot send any data payload.	CM_14	M
[MBIM 1.0] - 9.3.4#3	MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE shall be sent by the function if it detects a fragmented message out of sequence.	ERR_02	M
[MBIM 1.0] - 9.3.4#4	MBIM_ERROR_LENGTH_MISMATCH shall be sent by the function if the InformationBufferLength with required padding does not match the total of MessageLength minus headers.	ERR_06	M
[MBIM 1.0] - 9.3.4#5	MBIM_ERROR_DUPLICATED_TID shall be sent by the function if two MBIM commands are detected with the same TID.	ERR_09	M
[MBIM 1.0] - 9.3.4#6	The function shall respond with MBIM_ERROR_NOT_OPENED error code if it receives any MBIM commands prior to an open command or after a close command.	ERR_12	M
[MBIM 1.0] - 9.3.4#7	MBIM_ERROR_UNKNOWN shall be sent by the function when an unknown error is detected on the MBIM layer.		COA
[MBIM 1.0] - 9.3.4#8	MBIM_ERROR_MAX_TRANSFER shall be sent if the function does not support the maximum control transfer the host supports as specified in the MBIM_OPEN_MSG command.	ERR_14	M
[MBIM 1.0] - 9.3.4.1#1	A function that receives fragmented messages shall send an MBIM_ERROR_TIMEOUT_FRAGMENT if the time between the fragments exceeds 1250 ms.	ERR_15	M
[MBIM 1.0] - 9.3.4.1#2	A function that receives fragmented messages shall not send an MBIM_ERROR_TIMEOUT_FRAGMENT if the time between the fragments is less than 750 ms.	ERR_16	M
[MBIM 1.0] - 9.3.4.1#3	For MBIM_ERROR_TIMEOUT_FRAGMENT, the TransactionId of the responding message must match the TransactionId in the fragmented sequence that has the timing issue.	ERR_17	M
[MBIM 1.0] - 9.3.4.1#4	In case of a timeout error, the function shall discard all the packets with the same TransactionId as the fragmented message that has the timing issue.	ERR_18	M
[MBIM 1.0] - 9.3.4.2#1	The function shall stop transmitting the remaining packets with that TransactionId as soon as it receives the error message MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE.		COA
[MBIM 1.0] - 9.3.4.2#2	For MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE, the TransactionId of the responding message must match the TransactionId in the faulty fragmented sequence.	ERR_03	M
[MBIM 1.0] - 9.3.4.2#3	In case of an out of a sequence error, the function shall discard all the packets with the same TransactionId as the faulty message sequence.	ERR_04	M
[MBIM 1.0] - 9.3.4.2#4	If the function gets one more message that is out of order for the same TransactionId, it shall send a new error message with the same TransactionId once more.	ERR_05	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 9.3.4.3#1	For MBIM_ERROR_LENGTH_MISMATCH the TransactionId of the responding message must match the TransactionId of the faulty message.	ERR_07	M
[MBIM 1.0] - 9.3.4.3#2	In case of an MBIM_ERROR_LENGTH_MISMATCH all packets with the same TransactionId shall be discarded by the function.	ERR_08	M
[MBIM 1.0] - 9.3.4.4#1	For MBIM_ERROR_DUPLICATED_TID, the TransactionId of the responding message shall match the TransactionId of the duplicate message.	ERR_10	M
[MBIM 1.0] - 9.3.4.4#2	In case of an MBIM_ERROR_DUPLICATED_TID error, the function shall discard the newly arrived message.	ERR_11	M
[MBIM 1.0] - 9.3.4.5#1	If the host sends data traffic to the function while the function is in a "closed" state, the function shall respond with a MBIM_FUNCTION_ERROR_MSG status code MBIM_ERROR_NOT_OPENED.	ERR_13	M
[MBIM 1.0] - 9.3.4.5#2	If host sends MBIM_CLOSE_MSG while the function is still powering up, the function shall respond with an MBIM_FUNCTION_ERROR_MSG with MBIM_ERROR_NOT_OPENED status code.		COA
[MBIM 1.0] - 9.3.4.5#3	The function must not send any MBIM_COMMAND_DONE message after it has received a MBIM_ERROR_CANCEL message.		COA
[MBIM 1.0] - 9.3.4.6#1	For MBIM_ERROR_CANCEL the TransactionId of the responding message must match the TransactionId in the previous message in the sequence (if available).		COA
[MBIM 1.0] - 9.3.4.6#2	In case of a cancel error, the function shall discard all the packets with the same TransactionId as indicated in the MBIM_ERROR_CANCEL message.	ERR_19	M
[MBIM 1.0] - 9.4.1#1	The function shall respond to the MBIM_OPEN_MSG message with an MBIM_OPEN_DONE message in which the TransactionId must match the TransactionId in the MBIM_OPEN_MSG.	CM_01	M
[MBIM 1.0] - 9.4.1#2	The Status field of MBIM_OPEN_DONE shall be set to MBIM_STATUS_SUCCESS if the function initialized successfully.	CM_01	M
[MBIM 1.0] - 9.4.1#3	The Status field of MBIM_OPEN_DONE shall be set to error code indicating failure if the function not initialized successfully.		COA
[MBIM 1.0] - 9.4.2#1	The function shall respond to the MBIM_CLOSE_MSG message with an MBIM_CLOSE_DONE message in which the TransactionId must match the TransactionId in the MBIM_CLOSE_MSG.	CM_10	M
[MBIM 1.0] - 9.4.2#2	The Status field of MBIM_CLOSE_DONE shall always be set to MBIM_STATUS_SUCCESS.	CM_10	M
[MBIM 1.0] - 9.4.3	The function shall respond to the MBIM_COMMAND_MSG message with an MBIM_COMMAND_DONE message in which the TransactionId must match the TransactionId in the MBIM_COMMAND_MSG.	CM_04	
[MBIM 1.0] - 9.4.5#1	If the CID is successful, the function shall set the Status field to MBIM_STATUS_SUCCESS in the MBIM_COMMAND_DONE.	CM_06	M
[MBIM 1.0] - 9.4.5#2	If the function does not implement the CID, then the function shall fail the request with MBIM_STATUS_NO_DEVICE_SUPPORT.	CM_07	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 9.4.5#3	If the Status field returned to the host is not equal to MBIM_STATUS_SUCCESS, the function must set the Information BufferLength to 0, indicating an empty InformationBuffer except the following CIDs: MBIM_CID_REGISTER_STATE MBIM_CID_PACKET_SERVICE MBIM_CID_CONNECT MBIM_CID_SERVICE_ACTIVATION.	CM_08	M
[MBIM 1.0] - 9.5#1	Function should transmit fragmented message to host without intermixing fragments from other messages.	CM_16	M
[MBIM 1.0] - 10.3#1	As per MBIM recommendations, representation of string(s) should met the following constraints: • String offsets should be linear increasing. • String lengths must be a multiple of 2 (Unicode). • Offset of 0 means null string and must have a size of 0. • Strings must be non-overlapping. • String offset/size must be inside command/response buffer.	CM_17	M
[MBIM 1.0] - 10.3#2	The function shall reject incoming messages that don't follow the rules for variable-length encoding by setting MBIM_STATUS_INVALID_PARAMETERS as the status code in the MBIM_COMMAND_DONE message.	ERR_01	M
[MBIM 1.0] - 10.5.1.3#1	Functions that support CDMA must specify MBIMCtrlCapsCdmaMobileIP, or MBIMCtrlCapsCdmaSimpleIP, or both flags to inform the host about the type of IP that the function supports.	CID_01	M
[MBIM 1.0] - 10.5.1.3#2	Functions for sigle-mode CDMA-based devices must not specify MBIMCtrlCapsRegManual flag.	CID_02	M
[MBIM 1.0] - 10.5.1.3#3	As the connection credentials (AccessString, UserName, and Password) for simple IP are pre-configured, function firmware that supports both simple IP and mobile IP must report both capabilities, regardless of the runtime		COA
[MBIM 1.0] - 10.5.1.5#1	For GSM-based and multi-mode functions, the string DeviceId of MBIM_DEVICE_CAPS_INFO must conform to the International Mobile Equipment Identity (IMEI) format (up to 15 digits).	CID_03	M
[MBIM 1.0] - 10.5.1.5#2	For single-mode CDMA-based functions, the string DeviceId of MBIM_DEVICE_CAPS_INFO must conform to either the Electronic Serial Number (ESN, 8 or 11 digits) or the Mobile Equipment Identifier (MEID, 14 or 18 digits) formats.	CID_04	M
[MBIM 1.0] - 10.5.1.5#3	If DataClass bitmask in MBIM_DEVICE_CAPS_INFO structure does not contain 80000000h, then CustomDataClassOffset field is reserved and shall be encoded as zero by the function.	CID_05	M
[MBIM 1.0] - 10.5.1.5#4	If DataClass bitmask in MBIM_DEVICE_CAPS_INFO structure contains 80000000h, then CustomDataClassOffset and CustomDataClassSize shall not be zero.	CID_06	M
[MBIM 1.0] - 10.5.1.5#5	DEVICE_CAPS_INFO's MaxSessions field value should be <= 256d.	CID_07	M
[MBIM 1.0] - 10.5.2.1#1	After the SIM is unlocked, the function must send a MBIM_CID_SUBSCRIBER_READY_STATUS event notification with ReadyState set to the SIM card's new state.		COA
[MBIM 1.0] - 10.5.2.1#2	Functions must report all device ready-state changes as an unsolicited event.		COA

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 10.5.2.1#3	After the MBIM_OPEN_DONE message has been sent, the function shall always notify the host whenever the SIM ReadyState changes, using MBIM_INDICATE_STATUS_MSG with UUID_BASIC_CONNECT and MBIM_CID_SUBSCRIBER_READY_STATUS. This will not happen when notifications of this CID have been disabled with 10.5.30 MBIM_CID_DEVICE_SERVICE_SUBSCRIBE_LIST.		COA
[MBIM 1.0] - 10.5.2.1#4	If the SIM card has been initialized and the SIM requires PIN1 or PUK1 to be entered, the ReadyState is MBIMSubscriberReadyStateLocked.		COA
[MBIM 1.0] - 10.5.2.3#1	The function must provide a valid SubscriberId when the device ready-state is in MBIMSubscriberReadyStateInitialized.		COA
[MBIM 1.0] - 10.5.2.3#2	Functions must provide a valid SimIccId when the function's ready-state changes to MBIMSubscriberReadyStateInitialized as well as when the function is locked, waiting for entry of PIN1 and PUK1 keys.		COA
[MBIM 1.0] - 10.5.2.3#3	Functions must specify SimIccIdOffset value for all devices where MBIMCellularClass equals MBIMCellularClassGsm.		COA
[MBIM 1.0] - 10.5.2.3#4	Functions of CDMA-based devices must specify SimIccIdOffset value for devices where SimClass equals MBIMSimClassSimRemovable.		COA
[MBIM 1.0] - 10.5.2.5#1	Functions shall not return telephone numbers until the device ready-state changes to MBIMSubscriberReadyStateInitialized.		COA
[MBIM 1.0] - 10.5.2.5#2	If the ready state is not initialized, the function shall set ElementCount to zero, and shall not return any TNs on MBIM_SUBSCRIBER_READY_INFO.		COA
[MBIM 1.0] - 10.5.3.6#1	If the device does not specify MBIMCtrlCapsHwRadioSwitch the function must return MBIMRadioOn in HwRadioState field.	CID_08	M
[MBIM 1.0] - 10.5.4.1#2	CDMA function must report the power-on device lock as PIN1		COA
[MBIM 1.0] - 10.5.4.1#3	For all supported PIN types, functions must support the MBIMPinOperationEnter operation.		COA
[MBIM 1.0] - 10.5.4.1#4	If PIN1 is supported, functions must support the MBIMPinOperationEnable, MBIMPinOperationDisable, and MBIMPinOperationChange operations.		COA
[MBIM 1.0] - 10.5.4.1#5	If there are multiple PINs in enable state then functions must report PIN1 first.		COA
[MBIM 1.0] - 10.5.4.6#1	Functions that do not support reporting RemainingAttempts should set this member to 0xffffffff.		COA
[MBIM 1.0] - 10.5.5.1#1	A PIN reported as PIN1 must comply with PIN1 guidelines: for CDMA-based functions this is a PIN that provides power-on verification or identification functionality, and for GSM-based functions this is a Subscriber Identity Module (SIM) PIN.		COA
[MBIM 1.0] - 10.5.5.1#2	Functions must be able to return MBIM_CID_PIN_LIST information when the device ready-state changes to MBIMSubscriberReadyStateInitialized or when the device ready-state is MBIMSubscriberReadyStateDeviceLocked (PIN locked).		COA
[MBIM 1.0] - 10.5.5.1#3	The command MBIM_CID_PIN_LIST must report all the PINs supported by the function.		COA

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 10.5.6.4#1	Functions that do not support MBIM_CTRL_CAPS_MODEL_MULTI_CARRIER should set CellularClass field to zero.		COA
[MBIM 1.0] - 10.5.6.4#2	Functions that do not support MBIM_CTRL_CAPS_MODEL_MULTI_CARRIER should set ErrorRate field to zero.		COA
[MBIM 1.0] - 10.5.9.1#1	Functions that support manual registration must set the ControlCaps member in MBIM_DEVICE_CAPS_INFO structure to MBIM_CTRL_CAPS_REG_MANUAL.		COA
[MBIM 1.0] - 10.5.9.8	Functions that support CDMA must return MBIM_STATUS_NO_DEVICE_SUPPORT error code upon receiving a request for manual registration with a CDMA provider.		COA
[MBIM 1.0] - 10.5.12.1#1	Once a function has indicated an IP data stream session as unavailable to the host, the function must not automatically make the IP data stream session available again to the host.		COA
[MBIM 1.0] - 10.5.12.1#2	On MBIM_CID_CONNECT set request, the Host may specify an IP type to activate. If a value other than MBIMContextIPTypeDefault is specified, the function must only activate that context.	CID_09	M
[MBIM 1.0] - 10.5.12.1#3	Functions must only send MBIM_COMMAND_DONE for MBIM_CID_CONNECT's Set request after they have successfully activated or deactivated an IP data stream session, or detected an error.	CID_10	M
[MBIM 1.0] - 10.5.12.1#4	Function must use the value in MBIM_SET_CONNECT' SessionId member when completing set requests.	CID_11	M
[MBIM 1.0] - 10.5.12.1#5	When processing a MBIM_CID_CONNECT Set request and no IP data stream exists on the radio interface, the function shall try to establish it to the requested APN on the radio interface and make it available to the host.		COA
[MBIM 1.0] - 10.5.12.1#6	If the function receives a request to de-activate a context that is not currently activated, it shall respond with MBIM_STATUS_CONTEXT_NOT_ACTIVATED.	CID_12	M
[MBIM 1.0] - 10.5.12.1#7	If the function receives a Set request for MBIM_CID_CONNECT for a given SessionId while processing another Set request for MBIM_CID_CONNECT for that SessionId, the function shall fail the second Set request returning MBIM_STATUS_BUSY and continue processing the original Set request.		COA
[MBIM 1.0] - 10.5.20.7	If the function receives MBIM_CID_IP_CONFIGURATION query with SessionId specifying a context that is not currently activated, it shall respond with MBIM_STATUS_CONTEXT_NOT_ACTIVATED.	CID_13	M
[MBIM 1.0] - 10.5.29.1#1	Each device service supported by the device must have a separate MBIM_DEVICE_SERVICE_ELEMENT entry.		COA
[MBIM 1.0] - 10.5.29.1#2	There must be CidCount number of entries in the list of CIDs located in MBIM_DEVICE_SERVICE_ELEMENT structure.	CID_14	M

Assertion	Description	Test Identifier	Tags M (Mandatory) COA (Checklist Only Assertion)
[MBIM 1.0] - 11.2	The mandatory to implement functionality comprises the following CIDs from the Basic Connectivity Service: • MBIM_CID_DEVICE_CAPS • MBIM_CID_SUBSCRIBER_READY_INFO • MBIM_CID_RADIO_STATE • MBIM_CID_PIN • MBIM_CID_HOME_PROVIDER • MBIM_CID_REGISTER_STATE • MBIM_CID_SIGNAL_STATE • MBIM_CID_CONNECT • MBIM_CID_IP_CONFIGURATION_INFO • MBIM_CID_DEVICE_SERVICES • MBIM_CID_PACKET_SERVICE	CID_15	M

4 Check Only Assertions (Checklist)

Assertion	Description	YES / NO / NA
[MBIM 1.0] - 3.2.1#4	When alternate setting 0 of the Communication Interface of an NCM/MBIM function is selected, the function shall operate according to the NCM rules given in [USBNCM10]. In particular, NTBs shall transport Ethernet frames, not IP datagrams.	
[MBIM 1.0] - 5.2.3#1	If the transfer is less than the configured Max NTB size and is multiple of the wMaxPacketSize the function must terminate the transfer with a ZLP.	
[NCM 1.0] - 3.2.1#7	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.	
[NCM 1.0] - 3.2.2#7	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.	
[MBIM 1.0] - 7#2	To distinguish among the data streams, the last character of the dwSignature in the NDP16 header shall be coded with the DssSessionId specified by the host in the MBIM_CID_DSS_CONNECT command. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “DSS”<DssSessionId>.	
[MBIM 1.0] - 7#4	To distinguish among the data streams, the last character of the dwSignature in the NDP32 header shall be coded with the DssSessionId specified by the host in the MBIM_CID_DSS_CONNECT command. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “dss”<DssSessionId>.	
[NCM 1.0] - 3.4	Functions shall not send NTBs larger than the host has requested.	
[NCM 1.0] - 3.7#2	Transmitters are allowed to send a properly-formatted NTB containing an NDP whose datagram pointer entries are all zero. Receivers shall ignore such NTBs	
[MBIM 1.0] - 8.1.2#1	When the MBIM function is ready to send a control message to the host, the function must return a RESPONSE_AVAILABLE notification on the Communication Class interface’s Interrupt IN endpoint.	
[MBIM 1.0] - 8.1.2#4	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.	
[MBIM 1.0] - 8.1.5	In case of RESET_FUNCTION, the function shall abandon all outstanding transactions that are awaiting completion. No notifications shall be sent.	
[MBIM 1.0] - 9.3.4#1	An MBIM_FUNCTION_ERROR_MSG shall not be sent in response to an MBIM_HOST_ERROR_MSG.	
[MBIM 1.0] - 9.3.4#7	MBIM_ERROR_UNKNOWN shall be sent by the function when an unknown error is detected on the MBIM layer.	
[MBIM 1.0] - 9.3.4.2#1	The function shall stop transmitting the remaining packets with that TransactionId as soon as it receives the error message MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE.	
[MBIM 1.0] - 9.3.4.5#2	If host sends MBIM_CLOSE_MSG while the function is still powering up, the function shall respond with an MBIM_FUNCTION_ERROR_MSG with MBIM_ERROR_NOT_OPENED status code.	

Assertion	Description	YES / NO / NA
[MBIM 1.0] - 9.3.4.5#3	The function must not send any MBIM_COMMAND_DONE message after it has received a MBIM_ERROR_CANCEL message.	
[MBIM 1.0] - 9.3.4.6#1	For MBIM_ERROR_CANCEL the TransactionId of the responding message must match the TransactionId in the previous message in the sequence (if available).	
[MBIM 1.0] - 9.4.1#3	The Status field of MBIM_OPEN_DONE shall be set to error code indicating failure if the function not initialized successfully.	
[MBIM 1.0] - 10.5.1.3#3	As the connection credentials (AccessString, UserName, and Password) for simple IP are pre-configured, function firmware that supports both simple IP and mobile IP must report both capabilities, regardless of the runtime	
[MBIM 1.0] - 10.5.2.1#1	After the SIM is unlocked, the function must send a MBIM_CID_SUBSCRIBER_READY_STATUS event notification with ReadyState set to the SIM card's new state.	
[MBIM 1.0] - 10.5.2.1#2	Functions must report all device ready-state changes as an unsolicited event.	
[MBIM 1.0] - 10.5.2.1#3	After the MBIM_OPEN_DONE message has been sent, the function shall always notify the host whenever the SIM ReadyState changes, using MBIM_INDICATE_STATUS_MSG with UUID_BASIC_CONNECT and MBIM_CID_SUBSCRIBER_READY_STATUS. This will not happen when notifications of this CID have been disabled with 10.5.30 MBIM_CID_DEVICE_SERVICE_SUBSCRIBE_LIST.	
[MBIM 1.0] - 10.5.2.1#4	If the SIM card has been initialized and the SIM requires PIN1 or PUK1 to be entered, the ReadyState is MBIMSubscriberReadyStateLocked.	
[MBIM 1.0] - 10.5.2.3#1	The function must provide a valid SubscriberId when the device ready-state is in MBIMSubscriberReadyStateInitialized.	
[MBIM 1.0] - 10.5.2.3#2	Functions must provide a valid SimIccId when the function's ready-state changes to MBIMSubscriberReadyStateInitialized as well as when the function is locked, waiting for entry of PIN1 and PUK1 keys.	
[MBIM 1.0] - 10.5.2.3#3	Functions must specify SimIccIdOffset value for all devices where MBIMCellularClass equals MBIMCellularClassGsm.	
[MBIM 1.0] - 10.5.2.3#4	Functions of CDMA-based devices must specify SimIccIdOffset value for devices where SimClass equals MBIMSimClassSimRemovable.	
[MBIM 1.0] - 10.5.2.5#1	Functions shall not return telephone numbers until the device ready-state changes to MBIMSubscriberReadyStateInitialized.	
[MBIM 1.0] - 10.5.2.5#2	If the ready state is not initialized, the function shall set ElementCount to zero, and shall not return any TNs on MBIM_SUBSCRIBER_READY_INFO.	
[MBIM 1.0] - 10.5.4.1#2	CDMA function must report the power-on device lock as PIN1	
[MBIM 1.0] - 10.5.4.1#3	For all supported PIN types, functions must support the MBIMPinOperationEnter operation.	
[MBIM 1.0] - 10.5.4.1#4	If PIN1 is supported, functions must support the MBIMPinOperationEnable, MBIMPinOperationDisable, and MBIMPinOperationChange operations.	
[MBIM 1.0] - 10.5.4.1#5	If there are multiple PINs in enable state then functions must report PIN1 first.	

Assertion	Description	YES / NO / NA
[MBIM 1.0] - 10.5.4.6#1	Functions that do not support reporting RemainingAttempts should set this member to 0xffffffff.	
[MBIM 1.0] - 10.5.5.1#1	A PIN reported as PIN1 must comply with PIN1 guidelines: for CDMA-based functions this is a PIN that provides power-on verification or identification functionality, and for GSM-based functions this is a Subscriber Identity Module (SIM) PIN.	
[MBIM 1.0] - 10.5.5.1#2	Functions must be able to return MBIM_CID_PIN_LIST information when the device ready-state changes to MBIMSubscriberReadyStateInitialized or when the device ready-state is MBIMSubscriberReadyStateDeviceLocked (PIN locked).	
[MBIM 1.0] - 10.5.5.1#3	The command MBIM_CID_PIN_LIST must report all the PINs supported by the function.	
[MBIM 1.0] - 10.5.6.4#1	Functions that do not support MBIM_CTRL_CAPS_MODEL_MULTI_CARRIER should set CellularClass field to zero.	
[MBIM 1.0] - 10.5.6.4#2	Functions that do not support MBIM_CTRL_CAPS_MODEL_MULTI_CARRIER should set ErrorRate field to zero.	
[MBIM 1.0] - 10.5.9.1#1	Functions that support manual registration must set the ControlCaps member in MBIM_DEVICE_CAPS_INFO structure to MBIM_CTRL_CAPS_REG_MANUAL.	
[MBIM 1.0] - 10.5.9.8	Functions that support CDMA must return MBIM_STATUS_NO_DEVICE_SUPPORT error code upon receiving a request for manual registration with a CDMA provider.	
[MBIM 1.0] - 10.5.12.1#1	Once a function has indicated an IP data stream session as unavailable to the host, the function must not automatically make the IP data stream session available again to the host.	
[MBIM 1.0] - 10.5.12.1#5	When processing a MBIM_CID_CONNECT Set request and no IP data stream exists on the radio interface, the function shall try to establish it to the requested APN on the radio interface and make it available to the host.	
[MBIM 1.0] - 10.5.12.1#7	If the function receives a Set request for MBIM_CID_CONNECT for a given SessionId while processing another Set request for MBIM_CID_CONNECT for that SessionId, the function shall fail the second Set request returning MBIM_STATUS_BUSY and continue processing the original Set request.	
[MBIM 1.0] - 10.5.29.1#1	Each device service supported by the device must have a separate MBIM_DEVICE_SERVICE_ELEMENT entry.	

5 Standard Test Sequences

This section contains test sequences that are common for different tests described in section 6 of this document. Some tests require pre-execution of these sequences as a precondition.

NOTE: If a standard test sequence is used in a test all its steps shall be passed successfully, unless otherwise specified.

5.1 "Get Descriptors" Sequence

1. Place the device in the desired starting state.
2. Send GetDescriptor() request using the following parameters:
   - wValue – high byte set to 2d (Configuration), low byte set to the desired configuration value
   - wIndex – set to 0d
   - wLength – 9d

Test fails if the device returns bLength not equal to 9d.

3. Send GetDescriptor() request using the following parameters:
   - wValue – high byte set to 2d (Configuration), low byte set to the desired configuration value
   - wIndex – set to 0d
   - wLength – wTotalLength (All of the Configuration Set)
4. Parse the returned data.

Note: For all tests where this test sequence is listed as a precondition the sequence can be performed either once or multiple times, before each individual test; in the first case the returned data is stored to be analyzed later.

5.2 "MBIM Open – NTB-16" Sequence

1. Execute "Get Descriptors" sequence for the selected configuration if it has not been already executed.
2. Determine the number of Communication Interface and the number of Data Interface for the desired MBIM (or NCM/MBIM) function. Find and parse MBIM Functional Descriptor.
3. Select alternate setting 0d of the interface determined in step 2 for the Data Interface using SetInterface() request.
4. For NCM/MBIM function select alternate setting 1d for the interface determined in step 2 for the Communication Interface using SetInterface() request. Ignore this step if the function is MBIM only.
5. Send ResetFunction() request using the following parameter:
   - wIndex – set to the number of Communication Interface determined in step 2
6. Send GetNtbParameters() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 28d
7. Verify that bit 0 of bmNtbFormatsSupported field of the NTB Parameter Structure returned as a result of GetNtbParameters() request in step 6 is set to 1.
8. If bit 1 of bmNtbFormatsSupported field is set to 1, send SetNtbFormat() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wValue – set 0d (NTB-16)
   - wLength – set to 0d
9. Send SetNtbInputSize() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 4d
   - dwNtbInMaxSize field in the Data part – set to the value from dwNtbInMaxSize field of the NTB Parameter Structure returned as a result of the GetNtbParameters() request in step 6
10. If bit D3 in bmNetworkCapabilities field of the MBIM Functional Descriptor parsed in step 2 is set, send SetMaxDatagramSize() request using the following parameters:
    - wIndex – set to the number of the Communication Interface determined in step 2
    - wLength – set to 2d
    - Data – set to 1514d
11. Select alternate setting 1d (for MBIM only function) or 2d (for NCM/MBIM function) of the interface determined in step 2 for the Data Interface using SetInterface() request.
12. Send MBIM_OPEN_MSG message using the following parameters:
    - MessageLength – set to 16d
    - TransactionId – set to 1
    - MaxControlTransfer – set to wMaxControlMessage value from MBIM Functional Descriptor parsed in step 2
13. Retrieve an MBIM_OPEN_DONE response with TransactionId used in step 12.
14. Verify that the MBIM_OPEN_DONE response has been returned with Status == MBIM_STATUS_SUCCESS.

5.3 "MBIM Open – NTB-32" Sequence

1. Execute "Get Descriptors" sequence for the selected configuration if it has not been already executed.
2. Determine the number of Communication Interface and the number of Data Interface for the desired MBIM (or NCM/MBIM) function. Find and parse MBIM Functional Descriptor.
3. Select alternate setting 0d of the interface determined in step 2 for the Data Interface using SetInterface() request.
4. For NCM/MBIM function select alternate setting 1d for the interface determined in step 2 for the Communication Interface using SetInterface() request. Ignore this step if the function is MBIM only.
5. Send ResetFunction() request using the following parameter:
   - wIndex – set to the number of Communication Interface determined in step 2
6. Send GetNtbParameters() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 28d
7. Verify that bit 1 of bmNtbFormatsSupported field of the NTB Parameter Structure returned as a result of GetNtbParameters() request in step 6 is set to 1.
8. Send SetNtbFormat() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wValue – set 1d (NTB-32)
   - wLength – set to 0d
9. Send SetNtbInputSize() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 4d
   - dwNtbInMaxSize field in the Data part – set to the value from dwNtbInMaxSize field of the NTB Parameter Structure returned as a result of the GetNtbParameters() request in step 6
10. If bit D3 in bmNetworkCapabilities field of the MBIM Functional Descriptor parsed in step 2 is set, send SetMaxDatagramSize() request using the following parameters:
    - wIndex – set to the number of the Communication Interface determined in step 2
    - wLength – set to 2d
    - Data – set to 1514d
11. Select alternate setting 1d (for MBIM only function) or 2d (for NCM/MBIM function) of the interface determined in step 2 for the Data Interface using SetInterface() request.
12. Send MBIM_OPEN_MSG message using the following parameters:
    - MessageLength – set to 16d
    - TransactionId – set to 1d
    - MaxControlTransfer – set to wMaxControlMessage value from MBIM Functional Descriptor parsed in step 2
13. Retrieve an MBIM_OPEN_DONE response with TransactionId used in step 12.
14. Verify that the MBIM_OPEN_DONE response has been returned with Status == MBIM_STATUS_SUCCESS.

5.4 "MBIM Open" Generic Sequence

1. Execute "Get Descriptors" sequence for the selected configuration if it has not been already executed.
2. Determine the number of Communication Interface and the number of Data Interface for the desired MBIM (or NCM/MBIM) function. Find and parse MBIM Functional Descriptor.
3. Select alternate setting 0d of the interface determined in step 2 for the Data Interface using SetInterface() request.
4. For NCM/MBIM function select alternate setting 1d for the interface determined in step 2 for the Communication Interface using SetInterface() request. Ignore this step if the function is MBIM only.
5. Send ResetFunction() request using the following parameter:
   - wIndex – set to the number of Communication Interface determined in step 2
6. Send GetNtbParameters() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 28d
7. If bit 1 of bmNtbFormatsSupported field of the NTB Parameter Structure returned as a result of GetNtbParameters() request in step 6 is set to 1, send SetNtbFormat() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wValue – set 1d (NTB-32)
   - wLength – set to 0d
8. Send SetNtbInputSize() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 4d
   - dwNtbInMaxSize field in the Data part – set to the value from dwNtbInMaxSize field of the NTB Parameter Structure returned as a result of the GetNtbParameters() request in step 6
9. If bit D3 in bmNetworkCapabilities field of the MBIM Functional Descriptor parsed in step 2 is set, send SetMaxDatagramSize() request using the following parameters:
   - wIndex – set to the number of the Communication Interface determined in step 2
   - wLength – set to 2d
   - Data – set to 1514d

10. Select alternate setting 1d (for MBIM only function) or 2d (for NCM/MBIM function) of the interface determined in step 2 for the Data Interface using SetInterface() request.
11. Send MBIM_OPEN_MSG message using the following parameters:
    - MessageLength – set to 16d
    - TransactionId – set 1d
    - MaxControlTransfer – set to wMaxControlMessage value from MBIM Functional Descriptor parsed in step 2
12. Retrieve an MBIM_OPEN_DONE response with TransactionId used in step 11.
13. Verify that the MBIM_OPEN_DONE response has been returned with Status == MBIM_STATUS_SUCCESS.

5.5 "MBIM Close" Sequence

1. Send MBIM_CLOSE_MSG using the following parameters:
    - MessageLength – set to 12d
    - TransactionId – set to previous TransactionId + 1
2. Retrieve an MBIM_CLOSE_DONE response with TransactionId used in step 1.
3. Verify that the MBIM_CLOSE_DONE response has been returned with Status == MBIM_STATUS_SUCCESS.

5.6 "Connect" Sequence

1. Send MBIM_COMMAND_MSG using the following parameters:
    - MessageLength – set to 124d
    - TransactionId – set to previous TransactionId + 1
    - TotalFragments – set to 1d (assuming that the MaxControlTransfer value used in MBIM_OPEN_MSG message is larger or equal to 124d; if it is smaller than 124d, refer to section 9.5 of [MBIM 1.0] for information on how to fragment the message)
    - CurrentFragment – set to 0d
    - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
    - CID – set to 12d (MBIM_CID_CONNECT)
    - CommandType – set to 1d (Set)
    - InformationBufferLength – set to 76d
    - InformationBuffer (contains MBIM_SET_CONNECT structure)
        - SessionId – set to 0d
        - ActivationCommand – set to 1d (MBIMActivationCommandActivate)
        - AccessStringOffset – set to 60d
        - AccessStringSize – set to 16d
        - UserNameOffset –set to 0d
        - UserNameSize – set to 0d
        - PasswordOffset – set to 0d
        - PasswordSize – set to 0d
        - Compression – set to 0d (MBIMCompressionNone)
        - AuthProtocol – set to 0d (MBIMAuthProtocolNone)
        - IPType – set to 1d (MBIMContextIPTypeIPv4)
        - ContextType – set to 7E5E2A7E-4E6F-7272-736B-656E7E5E2A7E (MBIMContextTypeInternet)
        - DataBuffer – set to "loopback" coded in UTF-16LE
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_SUCCESS.

5.7 "Loopback NTB-16" Sequence

1. Make sure "Connect" sequence has been executed successfully (execute if necessary).
2. Send the following NTB (16-bit NTB, IPv4 "ping" packet):

    - **0x4E 0x43 0x4D 0x48 0x0C 0x00 0x## 0x## 0x80 0x00 0x70 0x00** 0x00 0x00 0x00 0x00
    - 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
    - **0x45 0x00 0x00 0x46 0x00 0x00 0x00 0x00 0x00 0x01 0xBC 0xB4 0x7F 0x00 0x00 0x01**
    - **0x7F 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x61 0x62 0x63 0x64**
    - **0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70 0x71 0x72 0x73 0x74**
    - **0x75 0x76 0x77 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69** 0x00 0x00 0x00 0x00
    - 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
    - **0x49 0x50 0x53 0x30 0x10 0x00** 0x00 0x00 **0x20 0x00 0x3C 0x00 0x00 0x00 0x00 0x00**

    The above mentioned NTB packet contains following parameters:

    NTB specific parameters and their values:

    wNdpOutDivisor - 32

wNdpOutPayloadRemainder - 0

NTH16 dwSignature - 0x4E 0x43 0x4D 0x48 (“NCMH”)
NTH16 wHeaderLength - 0x0C 0x00 (0x000C)
NTH16 wSequence - 0x## 0x## (sequence number)
NTH16 wBlockLength - 0x80 0x00 (0x0080)
NTH16 wNdpIndex - 0x70 0x00 (0x0070)

NDP16 dwSignature - 0x49 0x50 0x53 0x30 (“IPS0”)
NDP16 wLength - 0x10 0x00 (0x0010)
NDP16 wDatagramIndex[0] - 0x20 0x00 (0x0020)
NDP16 wDatagramLength[0] - 0x3C 0x00 (0x003C)
NDP16 wDatagramIndex[1] - 0x00 0x00 (0x0000)
NDP16 wDatagramLength[1] - 0x00 0x00 (0x0000)

IPv4 “ping” datagram (60 bytes):

- 0x45 0x00 0x00 0x46 0x00 0x00 0x00 0x00 0x00 0x01 0xBC 0xB4 0x7F 0x00 0x00 0x01
- 0x7F 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x61 0x62 0x63 0x64
- 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70 0x71 0x72 0x73 0x74
- 0x75 0x76 0x77 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69

NOTE:

The actual NTB formatting can be different since it depends on wNdpOutDivisor and wNdpOutPayloadRemainder parameters, which are function specific (these parameters are specified in NTB parameter structure; see section 6.2.1 of [NCM 1.0]). For more details see section 7 of [MBIM 1.0].

3. Receive an NTB with “looped” IPv4 “ping” packet and store the received NTB for further examination.

5.8 “Loopback NTB-32” Sequence

1. Make sure "Connect" sequence has been executed successfully (execute if necessary).
2. Send the following NTB (32-bit NTB, IPv4 “ping” packet):

- 0x6E 0x63 0x6D 0x68 0x10 0x00 0x## 0x## 0x90 0x00 0x00 0x00 0x70 0x00 0x00 0x00
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
- 0x45 0x00 0x00 0x46 0x00 0x00 0x00 0x00 0x00 0x01 0xBC 0xB4 0x7F 0x00 0x00 0x01
- 0x7F 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x61 0x62 0x63 0x64
- 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70 0x71 0x72 0x73 0x74
- 0x75 0x76 0x77 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x00 0x00 0x00 0x00
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
- 0x69 0x70 0x73 0x30 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
- 0x20 0x00 0x00 0x00 0x3C 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

The above mentioned NTB packet contains following parameters:

NTB specific parameters and their values:

wNdpOutDivisor - 32
wNdpOutPayloadRemainder - 0

NTH32 dwSignature - 0x6E 0x63 0x6D 0x68 (“ncmh”)
NTH32 wHeaderLength - 0x10 0x00 (0x000C)
NTH32 wSequence - 0x## 0x## (sequence number)
NTH32 dwBlockLength - 0x90 0x00 0x00 0x00 (0x00000090)
NTH32 dwNdpIndex - 0x70 0x00 0x00 0x00 (0x00000070)

NDP32 dwSignature - 0x69 0x70 0x73 0x30 (“ips0”)
NDP32 wLength - 0x20 0x00 (0x0020)
NDP32 dwDatagramIndex[0] - 0x20 0x00 0x00 0x00 (0x00000020)
NDP32 dwDatagramLength[0] - 0x3C 0x00 0x00 0x00 (0x0000003C)
NDP32 dwDatagramIndex[1] - 0x00 0x00 0x00 0x00 (0x00000000)
NDP32 dwDatagramLength[1] - 0x00 0x00 0x00 0x00 (0x00000000)

IPv4 “ping” datagram (60 bytes):

- 0x45 0x00 0x00 0x46 0x00 0x00 0x00 0x00 0x00 0x01 0xBC 0xB4 0x7F 0x00 0x00 0x01
- 0x7F 0x00 0x00 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x61 0x62 0x63 0x64
- 0x65 0x66 0x67 0x68 0x69 0x6A 0x6B 0x6C 0x6D 0x6E 0x6F 0x70 0x71 0x72 0x73 0x74

- **0x75 0x76 0x77 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69**

NOTE:

The actual NTB formatting can be different since it depends on wNdpOutDivisor and wNdpOutPayloadRemainder parameters, which are function specific (these parameters are specified in NTB parameter structure; see section 6.2.1 of [NCM 1.0]). For more details see section 7 of [MBIM 1.0].

3. Receive an NTB with "looped" IPv4 "ping" packet and store the received NTB for further examination.

5.9 "MBIM_CID_DEVICE_CAPS" Sequence

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageLength – set to 48d
   - TransactionId – set to previous TransactionId + 1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 0d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
   - CID – set to 1d (MBIM_CID_DEVICE_CAPS)
   - CommandType – set to 0d (Query)
   - InformationBufferLength – set to 0
   - InformationBuffer – NULL
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. If the MBIM_COMMAND_DONE response is returned with Status == MBIM_STATUS_SUCCESS, store the content of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response for further analysis.

5.10 "MBIM_CID_DEVICE_SERVICES" Sequence

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageType – set to 3d (MBIM_COMMAND_MSG)
   - MessageLength – set to 48d
   - TransactionId – set to old TransactionId+1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 0d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
   - CID – set to MBIM_CID_DEVICE_SERVICES
   - CommandType – set to 0d (Query)
   - InfromationBufferLength – set to 0d
   - InformationBuffer – NULL
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. If the MBIM_COMMAND_DONE response is returned with Status == MBIM_STATUS_SUCCESS, store the content of the MBIM_DEVICE_SERVICES_INFO structure returned in the MBIM_COMMAND_DONE response for further analysis.

6 Tests

This section contains a detailed description of test procedures. A typical description includes:

- Short description of the test.
- Assertions used in the test.
- Preconditions that shall be satisfied before performing the test.
- Test steps (may include references to the standard test sequences described in section 5 of this document).

NOTES:

1. To pass a test all its steps shall be passed successfully including successful execution of standard test sequences the test references to, unless otherwise specified.

2. Some of the tests defined in this specification when performed may change the internal state of the device function. Thus execution of a test may affect results of subsequent tests. It is assumed that proper re-initialization is performed between tests when necessary.

6.1 Descriptors Validation

The tests provided in this section validate descriptors for the combination NCM/MBIM and MBIM only functions.

DES_01 Descriptors Validation for NCM/MBIM Functions

This test validates descriptors for the combination NCM/MBIM functions.

Assertion(s) used in the test:

[MBIM 1.0] - 3.2.1#1: Functions that implement both NCM 1.0 and MBIM shall provide two alternate settings for the Communication Interface.

[MBIM 1.0] - 3.2.1#2: For alternate setting 0 of the Communication Interface of an NCM/MBIM function: interface, functional and endpoint descriptors shall be constructed according to the rules given in [USBNCM10].

[MBIM 1.0] - 3.2.1#3: For alternate setting 1 of the Communication Interface of an NCM/MBIM function: interface, functional and endpoint descriptors shall be constructed according to the rules given in [MBIM1.0] section 6.

[MBIM 1.0] - 3.2.1#4: When alternate setting 0 of the Communication Interface of an NCM/MBIM function is selected, the function shall operate according to the NCM rules given in [USBNCM10]. In particular, NTBs shall transport Ethernet frames, not IP datagrams.

[MBIM 1.0] - 3.2.1#5: When alternate setting 1 of the Communication Interface of an NCM/MBIM function is selected, the function shall operate according to the MBIM rules given in [USBMBIM10]. In particular, NTBs shall transport IP datagrams, not Ethernet frames

[MBIM 1.0] - 3.2.2.1#1: If an Interface Association Descriptor is used to form an NCM/MBIM function, its interface class, subclass, and protocol codes shall match those given in alternate setting 0 of the Communication Interface.

[MBIM 1.0] - 3.2.2.2#1: For an NCM/MBIM function the Communication Interface descriptor for alternate setting 0 must have bInterfaceSubClass == 0Dh and bInterfaceProtocol == XXh.

[MBIM 1.0] - 3.2.2.3#1: For an NCM/MBIM function, alternate setting 0 of the Communication Interface shall be followed by alternate setting 1.

[MBIM 1.0] - 3.2.2.3#2: For an NCM/MBIM function the Communication Interface descriptor for alternate setting 1 must have bInterfaceSubClass == 0Eh, and bInterfaceProtocol == 00h.

[MBIM 1.0] - 3.2.2.4#1: Functions that implement both NCM 1.0 and MBIM (an “NCM/MBIM function”) shall provide three alternate settings for the Data Interface.

[MBIM 1.0] - 3.2.2.4#2: For an NCM/MBIM function the Data Interface descriptors for alternate settings 0 and 1 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 01h.

[MBIM 1.0] - 3.2.2.4#3: For an NCM/MBIM function the Data Interface descriptor for alternate setting 2 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 02h.

[MBIM 1.0] - 3.2.2.4#4: For an NCM/MBIM function there must be no endpoints for alternate setting 0 of the Data Interface. For each of the other two alternate settings (1 and 2) there must be exactly two endpoints: one Bulk IN and one Bulk OUT.

[MBIM 1.0] - 6.3#2: MBIM Communication Interface description shall include the following functional descriptors:

- CDC Header Functional Descriptor
- CDC Union Functional Descriptor
- MBIM Functional Descriptor

Refer to Table 6.2 of [USBMBIM10].

[MBIM 1.0] - 6.3#3: CDC Header Functional Descriptor shall appear before CDC Union Functional Descriptor and before MBIM Functional Descriptor.

[MBIM 1.0] - 6.3#4: CDC Union Functional Descriptor for an MBIM function shall group together the MBIM Communication Interface and the MBIM Data Interface.

[MBIM 1.0] - 6.3#5: The class-specific descriptors must be followed by an Interrupt IN endpoint descriptor.

[MBIM 1.0] - 6.4#1: Field wMaxControlMessage of MBIM Functional Descriptor must not be smaller than 64.

[MBIM 1.0] - 6.4#2: Field bNumberFilters of MBIM Functional Descriptor must not be smaller than 16.

[MBIM 1.0] - 6.4#3: Field bMaxFilterSize of MBIM Functional Descriptor must not exceed 192.

[MBIM 1.0] - 6.4#4: Field wMaxSegmentSize of MBIM Functional Descriptor must not be smaller than 2048.

[MBIM 1.0] - 6.4#5: Field bFunctionLength of MBIM Functional Descriptor must be 12 representing the size of the descriptor.

[MBIM 1.0] - 6.4#6: Field bcdMBIMVersion of MBIM Functional Descriptor must be 0x0100 in little endian format.

[MBIM 1.0] - 6.4#7: Field bmNetworkCapabilities of MBIM Functional Descriptor should have the following bits set to zero: D7, D6, D4, D2, D1 and D0.

[MBIM 1.0] - 6.5#1: If MBIM Extended Functional Descriptor is provided, it must appear after MBIM Functional Descriptor.

[MBIM 1.0] - 6.5#2: Field bFunctionLength of MBIM Extended Functional Descriptor must be 8 representing the size of the descriptor.

[MBIM 1.0] - 6.5#3: Field bcdMBIMEFDVersion of MBIM Extended Functional Descriptor must be 0x0100 in little endian format.

[MBIM 1.0] - 6.5#4: Field bMaxOutstandingCommandMessages of MBIM Extended Functional Descriptor shall be greater than 0.

Precondition(s):

1. "Get Descriptors" sequence has been successfully executed for the selected configuration.

Test step(s):

1. Search for the first interface (bDescriptorType == 04h) with the following two alternate settings:
   - alternate setting 0:
     - bNumEndpoints == 1d
     - bInterfaceClass == 02h (Communications Interface)
     - bInterfaceSubClass == 0Dh (NCM)
   - alternate setting 1:
     - bNumEndpoints == 1d
     - bInterfaceClass == 02h (Communications Interface)
     - bInterfaceSubClass == 0Eh (MBIM)
     - bInterfaceProtocol == 00h

   The test stops and considered passed if the specified interface has not been found.
2. Verify that alternate setting 1 of the interface most recently found in step 1 or step 17 is specified after alternate setting 0.
3. Determine the interface bundle for alternate setting 0 of the interface most recently found in step 1 or step 17. The bundle includes the interface descriptor for alternate setting 0, any functional (bDescriptorType == 24h), endpoint (bDescriptorType == 05h) and endpoint companion (bDescriptorType == 30h) descriptors located after the specified interface descriptor, but before the next standard descriptor with bDescriptorType different from 25h, 05h and 30h.
4. Find the first functional descriptor (bDescriptorType == 24h) in the interface bundle most recently determined in step 3. Verify that
   - it's a Header Functional Descriptor (bDescriptorSubtype == 00h) with
     - bFunctionLength == 5d
     - bcdCDC >= 0120h
   - there is no other Header Functional Descriptor in the interface bundle most recently determined in step 3
5. Find a Union Function Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 06h) in the interface bundle most recently determined in step 3. Verify that
   - bFunctionLength == 5d
   - bControlInterface == bInterfaceNumber of the interface most recently found in step 1 or step 17
   - bSubordinateInterface0 == bInterfaceNumber of the interface with the following three alternate settings and endpoint configurations:
     - alternate setting 0:
       - bNumEndpoints == 0d
       - bInterfaceClass == 0Ah (Data Interface)
       - bInterfaceSubClass == 00h
       - bInterfaceProtocol == 01h
       - no endpoint descriptors shall follow the interface descriptor for this alternate setting
     - alternate setting 1:
       - bNumEndpoints == 2d
       - bInterfaceClass == 0Ah (Data Interface)
       - bInterfaceSubClass == 00h
       - bInterfaceProtocol == 01h

        - exactly 2 endpoint descriptors shall follow the interface descriptor for this alternate setting in arbitrary order:
            1. Bulk OUT endpoint:
                - bLength == 7
                - bEndpointAddress < 80h (OUT)
                - bmAttributes == 02h (Bulk)
            2. Bulk IN endpoint:
                - bLength == 7
                - bEndpointAddress >= 80h (IN)
                - bmAttributes == 02h (Bulk)
    - alternate setting 2:
        - bNumEndpoints == 2d
        - bInterfaceClass == 0Ah (Data Interface)
        - bInterfaceSubClass == 00h
        - bInterfaceProtocol == 02h
        - exactly 2 endpoint descriptors shall follow the interface descriptor for this alternate setting in arbitrary order:
            1. Bulk OUT endpoint:
                - bLength == 7d
                - bEndpointAddress < 80h (OUT)
                - bmAttributes == 02h (Bulk)
            2. Bulk IN endpoint:
                - bLength == 7
                - bEndpointAddress >= 80h (IN)
                - bmAttributes == 02h (Bulk)
  - the alternate settings of the interface specified in bSubordinateInterface0 appear after alternate setting 1 of the interface most recently found in step 1 or step 17
  - there is no other Union Functional Descriptor in the interface bundle most recently determined in step 3 that is not a duplicate of the descriptor found in this step
6. Find an Ethernet Networking Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 0Fh) in the interface bundle most recently determined in step 3. Verify that
   - bFunctionLength == 13d
   - wMaxSegmentSize >= 1514d
   - there is no other Ethernet Networking Functional Descriptor in the interface bundle most recently determined in step 3 that is not a duplicate of the descriptor found in this step
7. Find an NCM Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 1Ah) in the interface bundle most recently determined in step 3. Verify that
   - bcdNcmVersion >= 0100h
   - there is no other NCM Functional Descriptor in the interface bundle most recently determined in step 3 that is not a duplicate of the descriptor found in this step
8. If bInterfaceProtocol == FEh for the alternate setting 0 of the interface most recently found in step 1 or step 17, find a Command Set Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 16h) in the interface bundle most recently determined in step 3. Verify that
   - bFunctionLength == 22d
   - bcdVersion >= 0110h
   - there is no other Command Set Functional Descriptor in the interface bundle most recently determined in step 3 that is not a duplicate of the descriptor found in this step
9. If bInterfaceProtocol != FEh for the alternate setting 0 of the interface most recently found in step 1 or step 17, verify that there is no Command Set Detail Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 17h) in the interface bundle most recently determined in step 3.
10. Find the first endpoint descriptor (bDescriptorType == 05h) in the interface bundle most recently determined in step 3. Verify that
    - bLength == 7d
    - bEndpointAddress >= 80h (IN)
    - bmAttributes == 03h (Interrupt)
    - there is no other endpoint descriptor in the interface bundle most recently determined in step 3
    - all the functional descriptors in the interface bundle most recently determined in step 3 are located before the endpoint descriptor found in this step
11. Determine the interface bundle for alternate setting 1 of the interface most recently found in step 1 or step 17. The bundle includes the interface descriptor for alternate setting 1, any functional (bDescriptorType == 24h), endpoint (bDescriptorType == 05h) and endpoint companion (bDescriptorType == 30h) descriptors located after the specified interface descriptor, but before the next standard descriptor with bDescriptorType different from 25h, 05h and 30h.
12. Find the first functional descriptor (bDescriptorType == 24h) in the interface bundle most recently determined in step 11. Verify that
    - it's a Header Functional Descriptor (bDescriptorSubtype == 00h) with
        - bFunctionLength == 5d
        - bcdCDC >= 0120h
    - there is no other Header Functional Descriptor in the interface bundle most recently determined in step 11
13. Find a Union Function Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 06h) in the interface bundle most recently determined in step 11. Verify that
    - the descriptor duplicates the descriptor most recently found in step 5

- there is no other Union Functional Descriptor in the interface bundle most recently determined in step 11 that is not a duplicate of the descriptor found in this step

14. Find an MBIM Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 1Bh) in the interface bundle most recently determined in step 11. Verify that
    - bFunctionLength == 12d
    - bcdMBIMVersion == 0100h
    - wMaxControlMessage >= 64d
    - bNumberFilters >= 16d
    - bMaxFilterSize <= 192d
    - wMaxSegmentSize >= 2048d
    - in bmNetworkCapabilities field the following bits shall be zero: D7, D6, D4, D2, D1, D0
    - there is no other MBIM Functional Descriptor in the interface bundle most recently determined in step 11 that is not a duplicate of the descriptor found in this step
15. Search for the first instance of optional MBIM Extended Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 1Ch) in the interface bundle most recently determined in step 11. If the specified descriptor has been found, verify that
    - all the MBIM Functional Descriptors found in step 14 are located before the descriptor found in this step
    - bFunctionLength == 8d
    - bcdMBIMEFDVersion == 0100h
    - bMaxOutstandingCommandMessages > 0
    - there is no other MBIM Extended Functional Descriptor in the interface bundle most recently determined in step 11 that is not a duplicate of the descriptor found in this step
16. Find the first endpoint descriptor (bDescriptorType == 05h) in the interface bundle most recently determined in step 11. Verify that
    - bLength == 7d
    - bEndpointAddress >= 80h (IN)
    - bmAttributes == 03h (Interrupt)
    - there is no other endpoint descriptor in the interface bundle most recently determined in step 11
    - all the functional descriptors in the interface bundle most recently determined in step 11 are located before the endpoint descriptor found in this step
17. Search for all Interface Association Descriptors (bDescriptorType == 0Bh) for which bFirstInterface <= bControlInterface < bFirstInterface + bInterfaceCount or bFirstInterface <= bSubordinateInterface0 < bFirstInterface + bInterfaceCount, where bControlInterface and bSubordinateInterface0 are the fields of the Union Functional Descriptor most recently found in step 5. If at least one descriptor with the specified fields has been found, verify that
    - all found descriptors are located before the interfaces which numbers are specified in bControlInterface and bSubordinateInterface0
    - if there is more than one descriptor found, all the found descriptors duplicate each other
    - bFirstInterface == bControlInterface
    - bInterfaceCount == 2d
    - bSubordinateInterface0 == bControlInterface + 1
    - bFunctionClass == 02h (Communications Interface)
    - bFunctionSubClass == 0Dh (NCM)
    - bFunctionProtocol == bInterfaceProtocol specified in the alternate setting 0 of the interface most recently found in step 1 or step 17
18. Search for the next interface (bDescriptorType == 04h) with the alternate settings specified in step 1. If the interface has been found, proceed to step 2.

The test passes if steps 2 – 17 when performed are passed successfully. Note: all the “finds” shall necessarily succeed.

DES_02 Descriptors Validation for MBIM Only Functions

This test validates descriptors for MBIM only functions.

Assertion(s) used in the test:

[MBIM 1.0] - 6.1#1: If an Interface Association Descriptor (IAD) is provided for the MBIM function, the IAD and the mandatory CDC Union Functional Descriptor specified for the MBIM function shall group together the same interfaces.

[MBIM 1.0] - 6.1#2: If an Interface Association Descriptor (IAD) is provided for the MBIM only function, its interface class, subclass, and protocol codes shall match those given in the Communication Interface descriptor.

[MBIM 1.0] - 6.3#1: The descriptor for alternate setting 0 of the Communication Interface of an MBIM only function shall have bInterfaceClass == 02h, bInterfaceSubClass == 0Eh, and bInterfaceProtocol == 00h.

[MBIM 1.0] - 6.3#2: MBIM Communication Interface description shall include the following functional descriptors:

- CDC Header Functional Descriptor
- CDC Union Functional Descriptor
- MBIM Functional Descriptor

Refer to Table 6.2 of [USBMBIM10].

[MBIM 1.0] - 6.3#3: CDC Header Functional Descriptor shall appear before CDC Union Functional Descriptor and before MBIM Functional Descriptor.

[MBIM 1.0] - 6.3#4: CDC Union Functional Descriptor for an MBIM function shall group together the MBIM Communication Interface and the MBIM Data Interface.

[MBIM 1.0] - 6.3#5: The class-specific descriptors must be followed by an Interrupt IN endpoint descriptor.

[MBIM 1.0] - 6.4#1: Field wMaxControlMessage of MBIM Functional Descriptor must not be smaller than 64.

[MBIM 1.0] - 6.4#2: Field bNumberFilters of MBIM Functional Descriptor must not be smaller than 16.

[MBIM 1.0] - 6.4#3: Field bMaxFilterSize of MBIM Functional Descriptor must not exceed 192.

[MBIM 1.0] - 6.4#4: Field wMaxSegmentSize of MBIM Functional Descriptor must not be smaller than 2048.

[MBIM 1.0] - 6.4#5: Field bFunctionLength of MBIM Functional Descriptor must be 12 representing the size of the descriptor.

[MBIM 1.0] - 6.4#6: Field bcdMBIMVersion of MBIM Functional Descriptor must be 0x0100 in little endian format.

[MBIM 1.0] - 6.4#7: Field bmNetworkCapabilities of MBIM Functional Descriptor should have the following bits set to zero: D0, D1, D2, D4, D6 and D7.

[MBIM 1.0] - 6.5#1: If MBIM Extended Functional Descriptor is provided, it must appear after MBIM Functional Descriptor.

[MBIM 1.0] - 6.5#2: Field bFunctionLength of MBIM Extended Functional Descriptor must be 8 representing the size of the descriptor.

[MBIM 1.0] - 6.5#3: Field bcdMBIMEFDVersion of MBIM Extended Functional Descriptor must be 0x0100 in little endian format.

[MBIM 1.0] - 6.5#4: Field bMaxOutstandingCommandMessages of MBIM Extended Functional Descriptor shall be greater than 0.

[MBIM 1.0] - 6.6#1: The Data Interface for an MBIM only function shall provide two alternate settings.

[MBIM 1.0] - 6.6#2: The first alternate setting for the Data Interface of an MBIM only function (the default interface setting, alternate setting 0) shall include no endpoints.

[MBIM 1.0] - 6.6#3: The second alternate setting for the Data Interface of an MBIM only function (alternate setting 1) is used for normal operation, and shall include one Bulk IN endpoint and one Bulk OUT endpoint.

[MBIM 1.0] - 6.6#4: For an MBIM only function the Data Interface descriptors for alternate settings 0 and 1 must have bInterfaceSubClass == 00h, and bInterfaceProtocol == 02h. Refer to Table 6.4 of [USBMBIM10].

Precondition(s):

1. "Get Descriptors" sequence has been executed for the selected configuration without faults.

Test step(s):

1. Search for the first interface (bDescriptorType == 04h) with the following alternate setting:
   - alternate setting 0:
     - bNumEndpoints == 1d
     - bInterfaceClass == 02h (Communications Interface)
     - bInterfaceSubClass == 0Eh (MBIM)
     - bInterfaceProtocol == 00h

   The test stops and considered passed if the specified interface has not been found.
2. Determine the interface bundle for alternate setting 0 of the interface most recently found in step 1 or step 8. The bundle includes the interface descriptor for alternate setting 0, any functional (bDescriptorType == 24h), endpoint (bDescriptorType == 05h) and endpoint companion (bDescriptorType == 30h) descriptors located after the specified interface descriptor, but before the next standard descriptor with bDescriptorType different from 25h, 05h and 30h.
3. Find the first functional descriptor (bDescriptorType == 24h) in the interface bundle most recently determined in step 2. Verify that
   - it's a Header Functional Descriptor (bDescriptorSubtype == 00h) with
     - bFunctionLength == 5d
     - bcdCDC >= 0120h
   - there is no other Header Functional Descriptor in the interface bundle most recently determined in step 2
4. Find a Union Function Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 06h) in the interface bundle most recently determined in step 2. Verify that
   - bFunctionLength == 5d
   - bControlInterface == bInterfaceNumber of the interface most recently found in step 1 or step 8
   - bSubordinateInterface0 == bInterfaceNumber of the interface with the following two alternate settings and endpoint configurations:
     - alternate setting 0:
       - bNumEndpoints == 0d
       - bInterfaceClass == 0Ah (Data Interface)
       - bInterfaceSubClass == 00h
       - bInterfaceProtocol == 02h
       - no endpoint descriptors shall follow the interface descriptor for this alternate setting
     - alternate setting 1:
       - bNumEndpoints == 2d
       - bInterfaceClass == 0Ah (Data Interface)

        - bInterfaceSubClass == 00h
        - bInterfaceProtocol == 02h
        - exactly 2 endpoint descriptors shall follow the interface descriptor for this alternate setting in arbitrary order:
            1. Bulk OUT endpoint:
                - bLength == 7d
                - bEndpointAddress < 80h (OUT)
                - bmAttributes == 02h (Bulk)
            2. Bulk IN endpoint:
                - bLength == 7d
                - bEndpointAddress >= 80h (IN)
                - bmAttributes == 02h (Bulk)
    - there is no other Union Functional Descriptor in the interface bundle most recently determined in step 2 that is not a duplicate of the descriptor found in this step
5. Find an MBIM Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 1Bh) in the interface bundle most recently determined in step 2. Verify that
    - bFunctionLength == 12d
    - bcdMBIMVersion == 0100h
    - wMaxControlMessage >= 64d
    - bNumberFilters >= 16d
    - bMaxFilterSize <= 192d
    - wMaxSegmentSize >= 2048d
    - in bmNetworkCapabilities field the following bits shall be zero: D7, D6, D4, D2, D1, D0
    - there is no other MBIM Functional Descriptor in the interface bundle most recently determined in step 2 that is not a duplicate of the descriptor found in this step
6. Search for the first instance of optional MBIM Extended Functional Descriptor (bDescriptorType == 24h, bDescriptorSubtype == 1Ch) in the interface bundle most recently determined in step 2. If the specified descriptor has been found, verify that
    - all the MBIM Functional Descriptors found in step 5 are located before the descriptor found in this step
    - bFunctionLength == 8d
    - bcdMBIMEFDVersion == 0100h
    - bMaxOutstandingCommandMessages > 0
    - there is no other MBIM Extended Functional Descriptor in the interface bundle most recently determined in step 2 that is not a duplicate of the descriptor found in this step
7. Find the first endpoint descriptor (bDescriptorType == 05h) in the interface bundle most recently determined in step 2. Verify that
    - bLength == 7d
    - bEndpointAddress >= 80h (IN)
    - bmAttributes == 03h (Interrupt)
    - there is no other endpoint descriptor in the interface bundle most recently determined in step 2
    - all the functional descriptors in the interface bundle most recently determined in step 2 are located before the endpoint descriptor found in this step
8. Search for all Interface Association Descriptors (bDescriptorType == 0Bh) for which bFirstInterface <= bControlInterface < bFirstInterface + bInterfaceCount or bFirstInterface <= bSubordinateInterface0 < bFirstInterface + bInterfaceCount, where bControlInterface and bSubordinateInterface0 are the fields of the Union Functional Descriptor most recently found in step 4. If at least one descriptor with the specified fields has been found, verify that
    - all found descriptors are located before the interfaces which numbers are specified in bControlInterface and bSubordinateInterface0
    - if there is more than one descriptor found, all the found descriptors duplicate each other
    - bFirstInterface == bControlInterface or bFirstInterface == bSubordinateInterface0
    - bInterfaceCount == 2d
    - bSubordinateInterface0 == bControlInterface + 1 or bSubordinateInterface0 == bControlInterface – 1
    - bFunctionClass == 02h (Communications Interface)
    - bFunctionSubClass == 0Eh (MBIM)
    - bFunctionProtocol == 00h
9. Search for the next interface (bDescriptorType == 04h) with the alternate settings specified in step 1. If the interface has been found, proceed to step 2.

The test passes if steps 2 – 8 when performed are passed successfully. Note: all the “finds” shall necessarily succeed.

6.2 Data Transfer Validation

This section contains a test that validates the specifics of data transfer between the host and the function.

DTS_01 Validation for Alternate Setting 1 of the Communication Interface

This test validates data transfer operation for alternate setting 1 of the Communication Interface.

Assertion(s) used in the test:

[MBIM 1.0] - 3.2.1#5: When alternate setting 1 is selected, the function shall operate according to the MBIM rules given in [USBMBIM10]. In particular, NTBs shall transport IP datagrams, not Ethernet frames.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that "looped" packet transports an IP datagram.

6.3 Validation of 16-Bit NCM Transfer Header (NTH16)

This section contains tests that validate 16-bit NCM Transfer Header.

DTS_02 Validation of dwSignature

This test validates 16-bit NCM Transfer Header signature.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#1: The first four bytes in NTH16 shall be 0x484D434E in little-endian format ("NCMH").

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that dwSignature in the NCM Transfer Header of the received NTB (NTH16) is set to 484D434Eh ("NCMH").

DTS_03 Validation of wHeaderLength

This test validates the value in wHeaderLength field of NTH16.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#2: wHeaderLength value in NTH16 shall be 0x000C.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wHeaderLength in the NCM Transfer Header of the received NTB (NTH16) is set to 000Ch.

DTS_04 Validation of wSequence After Function Reset

This test verifies that function reset properly re-initializes the sequence number.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#3: wSequence in NTH16 shall be set to zero by the function in the first NTB transferred after every "function reset" event.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Execute "MBIM Open – NTB-16" sequence (includes "function reset").
3. Execute "Loopback NTB-16" sequence.
4. Verify that wSequence in the NCM Transfer Header of the received in step 3 NTB (NTH16) is set to 0.

DTS_05 Validation of wSequence Increment

This test verifies that the expected increment happens for wSequence.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#4: wSequence value in NTH16 shall be incremented for every NTB subsequent transfer.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Repeat "Loopback NTB-16" sequence from step 2 with wSequence in the NCM Transfer Header of the NTB being sent (NTH16) incremented by 1.
3. Verify that wSequence value in the NCM Transfer Header of the received NTB (NTH16) is incremented for each NTB received.

DTS_06 Validation of wBlockLength

This test validates the value in dwBlockLength field of NTH16.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#5: NTB size (IN) shall not exceed dwNtbInMaxSize.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wBlockLength value in the NCM Transfer Header of the received NTB (NTH16) is <= dwNtbInMaxSize value in the NTB Parameter Structure returned as a result of the GetNtbParameters() request in "MBIM Open – NTB-16" sequence.

DTS_07 Validation of wNdpIndex

This test validates the value in wNdpIndex field of NTH16.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.1#6: wNdpIndex value in NTH16 must be a multiple of 4, and must be >= 0x000C, in little endian.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wNdpIndex value in the NCM Transfer Header of the received NTB (NTH16) is >= 000Ch and is a multiple of 4.

6.4 Validation of 32-Bit NCM Transfer Header (NTH32)

This section contains tests that validate 32-bit NCM Transfer Header.

DTS_08 Validation of dwSignature

This test validates 32-bit NCM Transfer Header signature.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#1: The first four bytes in NTH32 shall be 0x686D636E in little-endian format ("ncmh").

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwSignature in the NCM Transfer Header of the received NTB (NTH32) is set to 686D636Eh ("ncmh").

DTS_09 Validation of wHeaderLength

This test validates the value in wHeaderLength field of NTH32.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#2: wHeaderLength value in NTH32 shall be 0x0010.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that wHeaderLength in the NCM Transfer Header of the received NTB (NTH32) is set to 0010h.

DTS_10 Validation of wSequence After Function Reset

This test verifies that function reset properly re-initializes the sequence number.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#3: wSequence in NTH32 shall be set to zero in the first NTB transferred after every "function reset" event.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Execute "MBIM Open – NTB-32" sequence (includes "function reset").

3. Execute "Loopback NTB-32" sequence.
4. Verify that wSequence in the NCM Transfer Header of the received in step 3 NTB (NTH32) is set to 0.

DTS_11 Validation of wSequence Increment

This test verifies that the expected increment happens for wSequence.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#4: wSequence value in NTH32 shall be incremented for every NTB subsequent transfer.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Repeat "Loopback NTB-32" sequence from step 2 with wSequence in the NCM Transfer Header of the NTB being sent (NTH32) incremented by 1.
3. Verify that wSequence value in the NCM Transfer Header of the received NTB (NTH32) is incremented for each NTB received.

DTS_12 Validation of dwBlockLength

This test validates the value in dwBlockLength field of NTH32.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#5: NTB size (IN) shall not exceed dwNtbInMaxSize.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwBlockLength value in the NCM Transfer Header of the received NTB (NTH32) is <= dwNtbInMaxSize value in the NTB Parameter Structure returned as a result of the GetNtbParameters() request in "MBIM Open – NTB-32" sequence.

DTS_13 Validation of dwNdpIndex

This test validates the value in wNdpIndex field of NTH32.

Assertion(s) used in the test:

[NCM 1.0] - 3.2.2#6: dwNdpIndex value in NTH32 must be a multiple of 4, and must be >= 0x0010.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwNdpIndex value in the NCM Transfer Header of the received NTB (NTH32) is >= 0010h and is a multiple of 4.

6.5 Validation of 16-Bit NCM Datagram Pointer (NDP16)

This section contains tests that validate 16-bit NCM Datagram Pointer.

DTS_14 Validation of dwSignature for IP Stream

This test validates 16-bit NCM Datagram Pointer signature for IP stream.

Assertion(s) used in the test:

[MBIM1.0] - 7#1: To distinguish among the data streams, the last character of the dwSignature in the NDP16 header shall be coded with the index SessionId specified by the host in the MBIM_CID_CONNECT. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: “IPS”<SessionId>.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute “Loopback NTB-16” sequence.
2. Verify that dwSignature in the NCM Datagram Pointer of the received NTB (NDP16) is set to 30535049h (“IPS0”).

DTS_15 Validation of wLength

This test validates the value in wLength field of NDP16.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.1#1: wLength value in the NDP16 must be a multiple of 4, and must be at least 16d (0x0010).

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute “Loopback NTB-16” sequence.
2. Verify that wLength value in the NCM Datagram Pointer of the received NTB (NDP16) is >= 0010h and is a multiple of 4.

DTS_16 Validation of wDatagramIndex[0]

This test validates the value in wDatagramIndex[0] field of NDP16.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.1#2: wDatagramIndex[0] value in NDP16 must be >= 0x000C (because it must point past the NTH16).

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute “Loopback NTB-16” sequence.
2. Verify that wDatagramIndex[0] value in the NCM Datagram Pointer of the received NTB (NDP16) is >= 000Ch.

DTS_17 Validation of wDatagramLength[0]

This test validates the value in wDatagramLength[0] field of NDP16.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.1#3: wDatagramLength[0] value in NDP16 must be >= 20d if datagram payload is IPv4 and >= 40d if datagram payload is IPv6.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wDatagramLength[0] value in the NCM Datagram Pointer of the received NTB (NDP16) is >= 20.

DTS_18 Validation of the Last wDatagramIndex

This test validates the value in wDatagramIndex[(wLength-8)/4 - 1] field of NDP16.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.1#4: wDatagramIndex[(wLength-8)/4 - 1] value in NDP16 must be zero.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wDatagramIndex[(wLength-8)/4 - 1] in the NCM Datagram Pointer of the received NTB (NDP16) is set to 0.

DTS_19 Validation of the Last wDatagramLength

This test validates the value in wDatagramLength[(wLength-8)/4 - 1] field of NDP16.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.1#5: wDatagramLength [(wLength-8)/4 - 1] value in NDP16 must be zero.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that wDatagramLength [(wLength-8)/4 - 1] in the NCM Datagram Pointer of the received NTB (NDP16) is set to 0.

6.6 Validation of 32-Bit NCM Datagram Pointer (NDP32)

This section contains test cases that validate 32-bit NCM Datagram Pointer.

DTS_20 Validation of dwSignature for IP Stream

This test validates 32-bit NCM Datagram Pointer signature for IP stream.

Assertion(s) used in the test:

[MBIM1.0] - 7#3: To distinguish among the data streams, the last character of the dwSignature in the NDP32 header shall be coded with the SessionId specified by the host in the MBIM_CID_CONNECT. The first three symbols are encoded as ASCII characters in little-endian form plus a last byte in HEX (binary) format: "ips"<SessionId>.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwSignature in the NCM Datagram Pointer of the received NTB (NDP32) is set to 30737069h ("ips0").

DTS_21 Validation of wLength

This test validates the value in wLength field of NDP32.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.2 #1: wLength value in NDP32 value must be a multiple of 8, and must be at least 32d (0x0020).

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that wLength value in the NCM Datagram Pointer of the received NTB (NDP32) is >= 0020h and is a multiple of 8.

DTS_22 Validation of dwDatagramIndex[0]

This test validates the value in dwDatagramIndex[0] field of NDP32.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.2#2: dwDatagramIndex[0] value in NDP32 must be >= 0x0010 (because it must point past the NTH32).

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwDatagramIndex[0] value in the NCM Datagram Pointer of the received NTB (NDP32) is >= 0010h.

DTS_23 Validation of dwDatagramLength[0]

This test validates the value in dwDatagramLength[0] field of NDP32.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.2#3: dwDatagramLength[0] value in NDP32 must be >= 20d if datagram payload is IPv4 and >= 40d if datagram payload is IPv6.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwDatagramLength[0] value in the NCM Datagram Pointer of the received NTB (NDP32) is >= 20d.

DTS_24 Validation of the Last dwDatagramIndex

This test validates the value in dwDatagramIndex[(wLength-8)/8 - 1] field of NDP32.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.2#4: dwDatagramIndex[(wLength-8)/8 - 1] value in NDP32 must be zero.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwDatagramIndex[(wLength-8)/8 - 1] in the NCM Datagram Pointer of the received NTB (NDP32) is set to 0.

DTS_25 Validation of the Last dwDatagramLength

This test validates the value in dwDatagramLength[(wLength-8)/8 - 1] field of NDP32.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.2#5: dwDatagramLength[(wLength-8)/8 - 1] value in NDP32 must be zero.

Precondition(s):

1. "MBIM Open – NTB-32" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-32" sequence.
2. Verify that dwDatagramLength[(wLength-8)/8 - 1] in the NCM Datagram Pointer of the received NTB (NDP32) is set to 0.

6.7 Validation of Datagram Payload Alignment

This section contains a test that validates datagram payload alignment.

DTS_26 Validation of Datagram Payload Alignment Based on wNdpInDivisor and wNdpInPayloadRemainder

This test verifies that the datagram payload is located at a proper offset.

Assertion(s) used in the test:

[NCM 1.0] - 3.3.4: The agent formatting a given NTB aligns the payload of each datagram by inserting padding, such that the offset of each datagram payload satisfies the constraint:
Offset % wNdpInDivisor == wNdpInPayloadRemainder (for IN datagrams).

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence.
2. Verify that the offset of the datagram payload in the received NTB satisfies the alignment requirement specified in the assertion.

6.8 Validation of Null NDP Handling Specifics

This section contains a test that validates the specifics of Null NDPs handling.

DTS_27 Validation of NDP Handling

This test verifies that function ignores all NDP entries following the first Null NDP entry.

Assertion(s) used in the test:

[NCM 1.0] - 3.7 #1: The first Null Datagram pointer entry in the NTB shall be interpreted as meaning that all following NCM Datagram Pointer Entries in the NDP are to be ignored.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.

Test step(s):

1. Execute "Loopback NTB-16" sequence putting 4 entries in the NDP instead of 2, where the second two entries duplicate the first two.
2. Verify that only one "looped" IPv4 "ping" packet has been received (i.e., the second two entries have been ignored by the function).

6.9 Control Requests Validation

This section contains a test that validates class-specific requests supported by MBIM interface class. These requests are sent over the default control pipe when the device is in USB configured state.

CREQ_01 Mandatory Control Requests Support Validation

This test verifies that MBIM function supports the mandatory control requests.

Assertion(s) used in the test:

[MBIM 1.0] - 8.1#1: The following requests must be supported by MBIM function:
- SendEncapsulatedCommand()
- GetEncapsulatedResponse()
- GetNtbParameters()
- SetNtbInputSize()
- GetNtbInputSize()
- ResetFunction()

Precondition(s):

1. "Get Descriptors" sequence has been executed for the selected configuration.

Test step(s):

1. Determine the number of Communication Interface and the number of Data Interface for the desired MBIM (or NCM/MBIM) function. Find and parse MBIM Functional Descriptor.
2. Select alternate setting 0 for the interface determined in step 2 for the Data Interface using SetInterface() request.
3. For NCM/MBIM function select alternate setting 1 of the interface determined in step 2 for the Communication Interface using SetInterface() request. Ignore this step if the function is MBIM only.
4. Send ResetFunction() request using the following parameter:
   - wIndex – set to the number of Communication Interface determined in step 2

   Verify that the request has succeeded (e.g., STALL is not returned, etc.).
5. Send GetNtbParameters() request using the following parameters:
   - wIndex – set to the number of Communication Interface determined in step 2
   - wLength – set to 28d

   Verify that the request has succeeded (e.g., STALL is not returned, etc.).
6. Send SetNtbInputSize() request using the following parameters:
   - wIndex – set to the number of Communication Interface determined in step 2
   - wLength – set to 4d
   - dwNtbInMaxSize field in the Data part – set to the value from dwNtbInMaxSize field of the NTB Parameter Structure returned as a result of the GetNtbParameters() request in step 5

   Verify that the request has succeeded (e.g., STALL is not returned, etc.).
7. Send GetNtbInputSize() request using the following parameters:

- wIndex – set to the number of Communication Interface determined in step 2
- wLength – set to 4d

Verify that the request has succeeded (e.g., STALL is not returned, etc.).

8. Send SendEncapsulatedCommand() request to send MBIM_OPEN_MSG using the following parameters:
   - MessageLength – set to 16d
   - TransactionId – set 1d
   - MaxControlTransfer – set to wMaxControlMessage from MBIM Functional Descriptor parsed in step 1

   Verify that the request has succeeded (e.g., STALL is not returned, etc.).
9. Receive RESPONSE_AVAILABLE notification on the Interrupt IN pipe (the corresponding endpoint is specified in the descriptor bundle of the currently selected alternate setting for the Communication Interface).
10. Retrieve the response by making GetEncapsulatedResponse() request with the following parameter:
    - wLength – set to wMaxControlMessage from MBIM Functional Descriptor parsed in step 1

    Verify that the request has succeeded (e.g., STALL is not returned, etc.).

6.10 Validation of MBIM_OPEN_MSG

This section contains tests that validate the specifics of MBIM_OPEN_MSG request and the function's response.

CM_01 Validation of Function's Response

This test verifies that MBIM_OPEN_DONE message is issued by the function in response to MBIM_OPEN_MSG message and checks TransactionId and Status fields.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.1#2: The function shall respond to the MBIM_OPEN_MSG message with an MBIM_OPEN_DONE message in which the TransactionId must match the TransactionId in the MBIM_OPEN_MSG.

[MBIM 1.0] - 9.4.1#2: The Status field of MBIM_OPEN_DONE shall be set to MBIM_STATUS_SUCCESS if the function initialized successfully.

Test step(s):

1. Execute "MBIM Open" sequence.

CM_02 Validation of MessageLength in MBIM_MESSAGE_HEADER

This test validates MessageLength field in MBIM_MESSAGE_HEADER.

Assertion(s) used in the test:

[MBIM 1.0] - 9.1#2: MessageLength in MBIM_MESSAGE_HEADER should be >= 0x0C

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Verify that MessageLength value in the MBIM_MESSAGE_HEADER structure returned in the MBIM_OPEN_DONE response to the MBIM_OPEN_MSG request is >= 0x0C.

CM_03 Validation of Function's Behavior in Case of an Unsynchronized Request

This test validates function's behavior in case of an unsynchronized open operation.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.1#1: In case MBIM_OPEN_MSG message is sent to a function that is already opened, the function shall interpret this as that the host and the function are out of synchronization. The function shall then perform the actions dictated by the MBIM_CLOSE_MSG before it performs the actions dictated by this command. The function shall not send the MBIM_CLOSE_DONE when the transition to the Closed state has been completed. Only the MBIM_OPEN_DONE message is sent upon successful completion of this message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM Open" sequence.
2. Verify that an MBIM_CLOSE_DONE message has not been received.

6.11 Validation of MBIM_COMMAND_MSG

This section contains tests that validate the specifics of MBIM_COMMAND_MSG request and the function's response.

CM_04 Validation of Function's Response

This test verifies that an MBIM_COMMAND_DONE message is issued by the function in response to an MBIM_COMMAND_MSG message and checks TransactionId field.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.3: The function shall respond to the MBIM_COMMAND_MSG message with an MBIM_COMMAND_DONE message in which the TransactionId must match the TransactionId in the MBIM_COMMAND_MSG.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.

CM_05 Validation of Function's Behavior in Case of Multiple Response Transactions

This test verifies that the function uses separate transactions to deliver control message responses.

Assertion(s) used in the test:

[MBIM 1.0] - 8.1.2#2: The function must use a separate GET_ENCAPSULATED_RESPONSE transfer for each control message it has to send to the host.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute the first step of "MBIM_CID_DEVICE_CAPS" sequence.
2. Execute the first step of "MBIM_CID_DEVICE_SERVICES" sequence.
3. Retrieve two MBIM_COMMAND_DONE with TransactionId, DeviceSeviceId and CID used in test sequencies in step 1 and step 2 in any order.

CM_06 Validation of Status in Case of Success

This test verifies that the function returns MBIM_STATUS_SUCCESS in Status field of MBIM_COMMAND_DONE response in case of a successfully executed command.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.5#1: If the CID is successful, the function shall set the Status field to MBIM_STATUS_SUCCESS in the MBIM_COMMAND_DONE.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.

CM_07 Validation of Status in Case of an Unsupported CID

This test verifies that the function returns MBIM_STATUS_NO_DEVICE_SUPPORT in Status field of the MBIM_COMMAND_DONE response when a command is not supported by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.5#2: If the function does not implement the CID, then the function shall fail the request with MBIM_STATUS_NO_DEVICE_SUPPORT.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageLength – set to 48d
   - TransactionId – set to old TransactionId+1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 1d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
   - CID – set to 255d (unsupported CID)
   - CommandType – set to 0d
   - InformationBufferLength – set to 0d
   - InformationBuffer – NULL
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_NO_DEVICE_SUPPORT.

CM_08 Validation of InformationBuffer in Case of a Failure

This test verifies that in case of a command failure the buffer in the MBIM_COMMAND_DONE response is empty.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.5#3: If the Status field returned to the host is not equal to MBIM_STATUS_SUCCESS, the function must set the InformationBufferLength to 0, indicating an empty InformationBuffer except the following CIDs:
MBIM_CID_REGISTER_STATE
MBIM_CID_PACKET_SERVICE
MBIM_CID_CONNECT
MBIM_CID_SERVICE_ACTIVATION.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageType – set to 3d (MBIM_COMMAND_MSG)
   - MessageLength – set to 52d
   - TransactionId – set to old TransactionId+1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 1d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)

- CID – set to MBIM_CID_RADIO_STATE
- CommandType – set to 1d (Set)
- InformationBufferLength – set to 4d
- InformationBuffer
  - RadioState – set to 2d (UnSupported Value)

2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that Status is not equal to MBIM_STATUS_SUCCESS and InformationBufferLength field in the retrieved MBIM_COMMAND_DONE response is set to 0.

6.12 Validation of MBIM_INDICATE_STATUS_MSG

This section contains a test that validates the specifics of unsolicited notifications.

CM_09 Validation of TransactionId for Notifications

This test verifies that TransactionId for notifications is zero.

Assertion(s) used in the test:

[MBIM 1.0] - 9.1#1: For notifications, the TransactionId must be set to 0 by the function.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "Connect" sequence.
2. Receive an MBIM_INDICATE_STATUS_MSG notification and verify that its TransactionId is set to 0.

6.13 Validation of MBIM_CLOSE_MSG

This section contains a test that validates the specifics of MBIM_CLOSE_MSG request and the function's response.

CM_10 Validation of Function's Response

This test verifies that an MBIM_CLOSE_DONE message is issued by the function in response to an MBIM_CLOSE_MSG message and checks TransactionId and Status fields.

Assertion(s) used in the test:

[MBIM 1.0] - 9.4.2#1: The function shall respond to the MBIM_CLOSE_MSG message with an MBIM_CLOSE_DONE message in which the TransactionId must match the TransactionId in the MBIM_CLOSE_MSG.

[MBIM 1.0] - 9.4.2#2: The Status field of MBIM_CLOSE_DONE shall always be set to MBIM_STATUS_SUCCESS.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM Close" sequence.

CM_11 Validation of Function's Behavior While Completing MBIM_CLOSE_MSG Request

This test verifies that the function ignores control messages while completing an MBIM_CLOSE_MSG request.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.2#1: Between the host's sending MBIM_CLOSE_MSG message and the function's completing the request (acknowledged with MBIM_CLOSE_DONE), the function shall ignore any MBIM control messages it receives on the control plane or the data on the bulk pipes.

Precondition(s):

1. "MBIM Open – NTB-16" sequence has been executed successfully.
2. "Connect" sequence has been executed successfully .

Test step(s):

1. Execute the first step of "MBIM Close" sequence.
2. Execute the first step of "MBIM_CID_DEVICE_CAPS" sequence.
3. Execute step 2 of "Loopback NTB-16" sequence.
4. Verify that an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 2 has not been received.
5. Verify that NTB with "looped" IPv4 "ping" packet has not been received.

CM_12 Validation of Function's Behavior after Completion of MBIM_CLOSE_MSG Request

This test verifies that the function does not send any control message after completion of MBIM_CLOSE_MSG request.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.2#2: The function shall not send any MBIM control messages on the control plane or data on the bulk pipes after completing MBIM_CLOSE_MSG message (acknowledging it with the MBIM_CLOSE_DONE message) with one exception and that is MBIM_ERROR_NOT_OPENED.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM Close" sequence.
2. Execute the first step of "Connect" sequence.
3. Verify that an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 2 has not been received.
4. Verify that no data has been sent on the bulk pipe after "MBIM Close" sequence had been executed.
5. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_NOT_OPENED.

CM_13 Validation of Active Context Termination on Function's Closing

This test verifies that no any active context exists after closing of the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.2#3: On MBIM_CLOSE_MSG, any active context between the function and the host shall be terminated.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "Connect" sequence has been executed successfully.
3. "MBIM Close" sequence has been executed successfully (after "Connect" sequence).
4. "MBIM Open" sequence has been executed successfully again.

Test step(s):

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageLength – set to 84d
   - TransactionId – set to previous TransactionId + 1

- TotalFragments – set to 1d (assuming that the MaxControlTransfer value used in MBIM_OPEN_MSG message is larger or equal to 84d; if it is smaller than 84d, refer to section 9.5 of [MBIM 1.0] for information on how to fragment the message)
- CurrentFragment – set to 0d
- DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
- CID – set to 12d (MBIM_CID_CONNECT)
- CommandType – set to 0d (Query)
- InformationBufferLength – set to 36d
- InformationBuffer (contains MBIM_ CONNECT_INFO structure)
  - SessionId – set to 0d
  - All other fields are ignored by function

2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 2.
3. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_CONTEXT_NOT_ACTIVATED.

6.14 Validation of MBIM_FUNCTION_ERROR_MSG

This section contains a test that validates the content of MBIM_FUNCTION_ERROR_MSG message.

CM_14 Validation of Not Sending Data Payload in Error Messages

This test verifies that an MBIM_FUNCTION_ERROR_MSG does contain a data payload.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4#2: An MBIM_FUNCTION_ERROR_MSG shall not make use of a DataBuffer, so it cannot send any data payload.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM Close" sequence has been executed successfully (the function is now in the closed state).

Test step(s):

1. Execute "Connect" sequence.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with MessageLength in the MBIM_MESSAGE_HEADER structure set to 16d (i.e., no data payload).

6.15 Validation of Message Fragmentation

This section contains tests that validate the specifics of message fragmentation.

CM_15 Validation of Message Fragmentation Ability

This test verifies that the function follows the rules of control message fragmentation.

Assertion(s) used in the test:

[MBIM 1.0] - 8.1.2#3: The function must send a RESPONSE_AVAILABLE notification for each available fragment of ENCAPSULATED_RESPONSE to be read from the default pipe.

[MBIM 1.0] - 9.2: Function should fragment responses based on MaxControlTransfer value from MBIM_OPEN_MSG.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence using wLength set to 64d for all GetEncapsulatedResponse() requests made in response to separate RESPONSE_AVAILABLE notifications.
2. Verify that
   - An MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_SUCCESS

- number of fragments of the returned MBIM_COMMAND_DONE response equals to the value in TotalFragments field of the fragment header structure located in each of these fragments
- MessageLength == 64d for all the fragments of the returned MBIM_COMMAND_DONE response, except for the very last one, which must be smaller or equal to 64d
- value in InformationBufferLength field of the first fragment of the returned MBIM_COMMAND_DONE response equals to the total size of information buffers in all the fragments of the returned MBIM_COMMAND_DONE response

CM_16 Validation of Fragmented Message Transmission in Case of Multiple Fragmented Messages

This test verifies that fragmented messages sent from the function are not intermixed. Note that this test is only applicable for devices that support multiple outstanding commands.

Assertion(s) used in the test:

[MBIM 1.0] - 9.5#1: Function should transmit fragmented message to host without intermixing fragments from other messages.

Precondition(s):

1. The test is only valid for functions that support multiple outstanding commands.
2. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "MBIM_CID_DEVICE_CAPS" sequence.
2. Execute step 1 of "MBIM_CID_DEVICE_SERVICES" sequence.
3. Retrieve two fragmented MBIM_COMMAND_DONE repsonses for the two query messages sent in steps 1 and 2 (expect TransactionId, DeviceSeviceId and CID equal to those from MBIM_COMMAND_MSG messages sent in steps 1 and 2).
4. Verify that the fragmented responses are sent without intermixing the fragments.

6.16 Validation of Variable Length Encoding

This section contains a test that validates function's ability to properly use variable length encoding.

CM_17 Validation of Strings Representation

This test verifies that the function follows the requirements for strings representation.

Assertion(s) used in the test:

[MBIM 1.0] - 10.3#1: As per MBIM recommendations, representation of string(s) should meet the following constraints:

- String offsets should be linear increasing.
- String lengths must be a multiple of 2 (Unicode).
- Offset of 0 means null string and must have a size of 0.
- Strings must be non-overlapping.
- String offset/size must be inside command/response buffer.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.
2. Verify the following in the DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command:
   - CustomdataClassOffset, CustomdataClassSize, CustomdataClass in DataBuffer satisfy the constraints specified in the above assertion
   - DeviceidOffset, DeviceidSize, Deviceid in DataBuffer satisfy the constraints specified in the above assertion
   - FirmwareInfoOffset, FirmwareInfoSize, FirmwareInfo in DataBuffer satisfy the constraints specified in the above assertion
   - HardwareInfoOffset, HardwareInfoSize, HardwareInfo in DataBuffer satisfy the constraints specified in the above assertion

6.17 Validation of Error Handling

This section contains tests that validate the specifics of MBIM error handling.

6.17.1 Validation of Variable-Length Encoding Error Handling

This section contains a test that validates function's ability to properly handle the errors of the variable-length encoding.

ERR_01 Validation of Function's Response to Messages with Variable-Length Encoding Errors

This test verifies that incoming messages are rejected when variable-length encoding rules are not followed.

Assertion(s) used in the test:

[MBIM 1.0] - 10.3#2: The function shall reject incoming messages that don't follow the rules for variable-length encoding by setting MBIM_STATUS_INVALID_PARAMETERS as the status code in the MBIM_COMMAND_DONE message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute the first step of "Connect" sequence with AccessStringOffset field of MBIM_SET_CONNECT structure set to 0.
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_INVALID_PARAMETERS.

6.17.2 Validation of MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE

This section contains tests that validate error messaging in case of out of sequence fragments.

ERR_02 Validation of Issuing the Error Message

This test verifies that an error message with status code MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE is issued when fragments received in a wrong order.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4#3: MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE shall be sent by the function if it detects a fragmented message out of sequence.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence deliberately sending the second fragment of the MBIM_COMMAND_MSG message before the first fragment.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE.

ERR_03 Validation of Error Message TransactionId

This test verifies that TransactionId of an error message with status code MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE is the same as TransactionId of the incorrectly fragmented message.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.2#2: For MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE, the TransactionId of the responding message must match the TransactionId in the faulty fragmented sequence.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence deliberately sending the second fragment of the MBIM_COMMAND_MSG message before the first fragment.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE and TransactionId equal to the TransactionId of the fragmented MBIM_COMMAND_MSG message from step 1.

ERR_04 Validation of Discarding Packets in Case of an Error

This test verifies that in case of an error message with status code MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE all packets of the message caused the error are discarded by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.2#3: In case of an out of a sequence error, the function shall discard all the packets with the same TransactionId as the faulty message sequence.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence deliberately sending the second fragment of the MBIM_COMMAND_MSG message before the first fragment.
2. Verify that an MBIM_COMMAND_DONE response with TransactionId of the fragmented MBIM_COMMAND_MSG message from step 1 has not been received (i.e., the MBIM_COMMAND_MSG message has been discarded by the function).

ERR_05 Validation of Issuing a New Error Message

This test verifies that another error message with status code MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE is issued when another message with out-of-order fragmentation with the same TransactionId is received.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.2#4: If the function gets one more message that is out of order for the same TransactionId, it shall send a new error message with the same TransactionId once more.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence deliberately sending the second fragment of the MBIM_COMMAND_MSG message two times in a row without sending the first fragment.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message with ErrorStatusCode == MBIM_ERROR_FRAGMENT_OUT_OF_SEQUENCE and TransactionId equal to the TransactionId of the fragmented MBIM_COMMAND_MSG message from step 1 has been received 2 times.

6.17.3 Validation of MBIM_ERROR_LENGTH_MISMATCH

This section contains tests that validate error messaging in case of a mismatch between the specified buffer length and the actual buffer length calculated based on the total message length.

ERR_06 Validation of Issuing the Error Message

This test verifies that an error message with status code MBIM_ERROR_LENGTH_MISMATCH is issued when InformationBufferLength value is inconsistend with MessageLength value.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.3#1: MBIM_ERROR_LENGTH_MISMATCH shall be sent by the function if the InformationBufferLength with required padding does not match the total of MessageLength minus headers.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute step 1 of "Connect" sequence with InformationBufferLength in the MBIM_COMMAND_MSG message set to 80d (not equal to the actual InformationBuffer length).
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatuCode == MBIM_ERROR_LENGTH_MISMATCH.

ERR_07 Validation of Error Message TransactionId

This test verifies that TransactionId of an error message with status code MBIM_ERROR_LENGTH_MISMATCH is the same as TransactionId of the faulty message.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.3#2: For MBIM_ERROR_LENGTH_MISMATCH the TransactionId of the responding message must match the TransactionId of the faulty message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute step 1 of "Connect" sequence with InformationBufferLength in the MBIM_COMMAND_MSG message set to 80d (not equal to the actual InformationBuffer length).
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_LENGTH_MISMATCH and TransactionId equal to the TransactionId of the faulty MBIM_COMMAND_MSG message from step 1.

ERR_08 Validation of Discarding Packets in Case of an Error

This test verifies that in case of an error message with status code MBIM_ERROR_LENGTH_MISMATCH all packets of the message caused the error are discarded by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.3#a3: In case of an MBIM_ERROR_LENGTH_MISMATCH all packets with the same TransactionId shall be discarded by the function.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute step 1 of "Connect" sequence with InformationBufferLength in the MBIM_COMMAND_MSG message set to 80d (not equal to the actual InformationBuffer length).
2. Verify that an MBIM_COMMAND_DONE response with TransactionId of the faulty MBIM_COMMAND_MSG message from step 1 has not been received (i.e., the MBIM_COMMAND_MSG message has been discarded by the function).

6.17.4 Validation of MBIM_ERROR_DUPLICATED_TID

This section contains tests that validate error messaging in case of a duplicate TransactionId.

ERR_09 Validation of Issuing the Error Message

This test verifies that an error message with status code MBIM_ERROR_DUPLICATED_TID is issued when the function receives a message with TransactionId already used in another message.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4#5: MBIM_ERROR_DUPLICATED_TID shall be sent by the function if two MBIM commands are detected with the same TID.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.
2. Execute the first step of "MBIM_CID_DEVICE_CAPS" sequence using TransactionId of the MBIM_COMMAND_MSG message from step 1.
3. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatuCode == MBIM_ERROR_DUPLICATED_TID.

ERR_10 Validation of Error Message TransactionId

This test verifies that TransactionId of an error message with status code MBIM_ERROR_DUPLICATED_TID is the same as TransactionId of the duplicate message.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.4#1: For MBIM_ERROR_DUPLICATED_TID, the TransactionId of the responding message shall match the TransactionId of the duplicate message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.
2. Execute the first step of "MBIM_CID_DEVICE_CAPS" sequence using TransactionId of the MBIM_COMMAND_MSG message from step 1.
3. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatuCode == MBIM_ERROR_DUPLICATED_TID and TransactionId equal to the TransactionId of the MBIM_COMMAND_MSG messages from steps 1 and 2.

ERR_11 Validation of Discarding Packets in Case of an Error

This test verifies that in case of an error message with status code MBIM_ERROR_DUPLICATED_TID all packets of the message caused the error are discarded by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.4#2: In case of an MBIM_ERROR_DUPLICATED_TID error, the function shall discard the newly arrived message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "MBIM_CID_DEVICE_CAPS" sequence.
2. Execute the first step of "MBIM_CID_DEVICE_CAPS" sequence using TransactionId of the MBIM_COMMAND_MSG from step 1.
3. Verify that an MBIM_COMMAND_DONE response with TransactionId of the MBIM_COMMAND_MSG messages from steps 1 and 2 has not been received for the second message (i.e., the second MBIM_COMMAND_MSG message has been discarded by the function).

6.17.5 Validation of MBIM_ERROR_NOT_OPENED

This section contains tests that validate error messaging in case a command is sent to a closed function.

ERR_12 Validation of Issuing the Error Message in Response to a Control Command

This test verifies that an error message with status code MBIM_ERROR_NOT_OPENED is issued in response to any command received by the function in closed state.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4#6: The function shall respond with MBIM_ERROR_NOT_OPENED error code if it receives any MBIM commands prior to an open command or after a close command.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM Close" sequence has been executed successfully (properly initialized function is now in the closed state).

Test step(s):

1. Execute "Connect" sequence.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_NOT_OPENED.

ERR_13 Validation of Issuing the Error Message in Response to Data Traffic

This test verifies that an error message with status code MBIM_ERROR_NOT_OPENED is issued in response to any data traffic received by the function in closed state.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.5#1 If the host sends data traffic to the function while the function is in a "closed" state, the function shall respond with a MBIM_FUNCTION_ERROR_MSG status code MBIM_ERROR_NOT_OPENED.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM Close" sequence has been executed successfully (properly initialized function is now in the closed state).

Test step(s):

1. Execute step 2 of the "Loopback NTB-16" sequence.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_NOT_OPENED.

6.17.6 Validation of MBIM_ERROR_ MAX_TRANSFER

This section contains a test that validates error messaging in case the function does not support the maximum control transfer size proposed by the host.

ERR_14 Validation of Issuing the Error Message

This test verifies that an error message with status code MBIM_ERROR_MAX_TRANSFER is issued when the function receives "open" request containing an unsupported value in MaxControlTransfer field.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4#8: MBIM_ERROR_MAX_TRANSFER shall be sent if the function does not support the maximum control transfer the host supports as specified in the MBIM_OPEN_MSG command.

Test step(s):

1. Execute "MBIM Open" sequence not performing the final 2 steps with MaxControlTransfer set to wMaxControlMessage + 1 (wMaxControlMessage value is taken from the MBIM Functional Descriptor).
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_MAX_TRANSFER.

6.17.7 Validation of MBIM_ERROR_TIMEOUT_FRAGMENT

This section contains tests that validate error messaging in case when a message is fragmented and the time between the fragments exceeds the predefined maximum.

ERR_15 Validation of Proper Handling of the Maximum Limit

This test verifies that an error message with status code MBIM_ERROR_TIMEOUT_FRAGMENT is issued when the delay between message fragments is too big.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.1#1: A function that receives fragmented messages shall send an MBIM_ERROR_TIMEOUT_FRAGMENT if the time between the fragments exceeds 1250 ms.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence delaying transmission of the second fragment of the MBIM_COMMAND_MSG message for more than 1250 ms.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_TIMEOUT_FRAGMENT.

ERR_16 Validation of Proper Handling of the Minimum Limit

This test verifies that an error message with status code MBIM_ERROR_TIMEOUT_FRAGMENT is not issued when the delay between message fragments is smaller than the predefined value.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.1#2: A function that receives fragmented messages shall not send an MBIM_ERROR_TIMEOUT_FRAGMENT if the time between the fragments is less than 750 ms.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence with a delay between fragments of the MBIM_COMMAND_MSG message smaller than 750 ms.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message with ErrorStatusCode == MBIM_ERROR_TIMEOUT_FRAGMENT has not been received.

ERR_17 Validation of Error Message TransactionId

This test verifies that TransactionId of an error message with status code MBIM_ERROR_TIMEOUT_FRAGMENT is the same as TransactionId of the fragmented message that has the timing issue.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.1#3: For MBIM_ERROR_TIMEOUT_FRAGMENT, the TransactionId of the responding message must match the TransactionId in the fragmented message that has the timing issue.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence delaying transmission of the second fragment of the MBIM_COMMAND_MSG message for more than 1250 ms.
2. Verify that an MBIM_FUNCTION_ERROR_MSG message has been received with ErrorStatusCode == MBIM_ERROR_TIMEOUT_FRAGMENT and TransactionId equal to the TransactionId of the fragmented MBIM_COMMAND_MSG message from step 1.

ERR_18 Validation of Discarding Packets in Case of an Error

This test verifies that in case of an error message with status code MBIM_ERROR_TIMEOUT_FRAGMENT all packets of the message caused the error are discarded by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.1#4: In case of a timeout error, the function shall discard all the packets with the same TransactionId as the fragmented message that has the timing issue.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence delaying transmission of the second fragment of the MBIM_COMMAND_MSG message for more than 1250 ms.
2. Verify that an MBIM_COMMAND_DONE response with TransactionId of the fragmented MBIM_COMMAND_MSG message from step 1 has not been received (i.e., the MBIM_COMMAND_MSG message has been discarded by the function).

6.17.8 Validation of MBIM_ERROR_CANCEL

This section contains a test that validates function's behavior upon receiving a request from the host to cancel a pending transaction.

ERR_19 Validation of Discarding Packets in Case of an Error

This test verifies that in case of a message cancellation request received from the host all packets of the message specified in the request are discarded by the function.

Assertion(s) used in the test:

[MBIM 1.0] - 9.3.4.6#2: In case of a cancel error, the function shall discard all the packets with the same TransactionId as indicated in the MBIM_ERROR_CANCEL message.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully with MaxControlTransfer set to 64d.

Test step(s):

1. Execute step 1 of "Connect" sequence sending only the first fragment of MBIM_COMMAND_MSG message.
2. Send MBIM_HOST_ERROR_MSG message using the following parameters:
   - MessageLength – set to 16d
   - TransactionId – set to TransactionId of MBIM_COMMAND_MSG message from step 1.
   - ErrorStatusCode – set to 7d (MBIM_ERROR_CANCEL).
3. Continue executing the first step of "Connect" sequence sending the second fragment of the MBIM_COMMAND_MSG message.
4. Verify that an MBIM_COMMAND_DONE response with TransactionId of the fragmented MBIM_COMMAND_MSG message has not been received (i.e., the MBIM_COMMAND_MSG message has been discarded by the function).

6.18 Validation of Mandatory Control Commands

This section contains test cases that validate the specifics of the mandatory control commands.

6.18.1 Validation of MBIM_CID_DEVICE_CAPS

This section contains tests that validate the specifics of CID MBIM_CID_DEVICE_CAPS command.

CID_01 Validation of IP Flags for Functions That Support CDMA

This test verifies that a function that supports CDMA specifies at least one of the following IP flags: MBIMCtrlCapsCdmaMobileIP, MBIMCtrlCapsCdmaSimpleIP.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.3#1: Functions that support CDMA must specify MBIMCtrlCapsCdmaMobileIP, or MBIMCtrlCapsCdmaSimpleIP, or both flags to inform the host about the type of IP that the function supports.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and MBIMCellularClassCdma bit is set in CellularClass field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command.

Test step(s):

1. Verify that ControlCaps field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command specifies at least one of the following flags: MBIMCtrlCapsCdmaMobileIP, MBIMCtrlCapsCdmaSimpleIP.

CID_02 Validation of Registration Method for Single-Mode CDMA Functions

This test verifies that a single-mode CDMA function does not support manual registration.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.3#2: Functions for single-mode CDMA-based devices must not specify MBIMCtrlCapsRegManual flag.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and MBIMCellularClassCdma bit is set and MBIMCellularClassGsm is not set in CellularClass field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command.

Test step(s):

1. Verify that MBIMCtrlCapsRegManual bit is not set in ControlCaps field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command.

CID_03 Validation of DeviceId for Functions That Support GSM

This test verifies that DeviceId in case of a function that supports GSM is in IMEI format.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.5#1: For GSM-based and multi-mode functions, the string DeviceId of MBIM_DEVICE_CAPS_INFO must conform to the International Mobile Equipment Identity (IMEI) format (up to 15 digits).

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and MBIMCellularClassGsm bit is set in CellularClass field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to MBIM_CID_DEVICE_CAPS command.

Test step(s):

1. Verify that DeviceId string located in the DataBuffer of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command is represented in IMEI format.

CID_04 Validation of DeviceId Field for Single-Mode CDMA Functions

This test verifies that DeviceId in case of a single-mode CDMA function is either in ESN format or in MEID format.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.5#2: For single-mode CDMA-based functions, the string DeviceId of MBIM_DEVICE_CAPS_INFO must conform to either the Electronic Serial Number (ESN, 8 or 11 digits) or the Mobile Equipment Identifier (MEID, 14 or 18 digits) formats.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and MBIMCellularClassCdma bit is set and MBIMCellularClassGsm is not set in CellularClass field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to MBIM_CID_DEVICE_CAPS command.

Test step(s):

1. Verify that DeviceId string located in the DataBuffer of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command is represented in either ESN or MEID formats.

CID_05 Validation of CustomDataClassOffset for Functions That Do Not Support Custom Data Classes

This test verifies that CustomDataClassOffset is not specified when the function does not support a custom data class.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.5#3: If DataClass bitmask in MBIM_DEVICE_CAPS_INFO structure does not contain 80000000h, then CustomDataClassOffset field is reserved and shall be encoded as zero by the function.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and DataClass bitmask in the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_DEVICE_CAPS command does not contain 80000000h.

Test step(s):

1. Verify that CustomDataClassOffset in the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_DEVICE_CAPS command is set to zero.

CID_06 Validation of CustomDataClass String for Functions That Support Custom Data Classes

This test verifies that CustomDataClass string is specified when the function supports a custom data class.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.5#4: If DataClass bitmask in MBIM_DEVICE_CAPS_INFO structure contains 80000000h, then CustomDataClassOffset and CustomDataClassSize shall not be zero.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and DataClass bitmask in the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command contains 80000000h.

Test step(s):

1. Verify that CustomDataClassOffset and CustomDataClassSize fields of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to MBIM_CID_DEVICE_CAPS command are nonzero.

CID_07 Validation of MaxSessions Field in DEVICE_CAPS_INFO structure

This test validates the value in MaxSessions field of DEVICE_CAPS_INFO structure.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.1.5#5: DEVICE_CAPS_INFO's MaxSessions field value should be <= 256d.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully.

Test step(s):

1. Verify that MaxSessions value in MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_CAPS command is <= 256d.

6.18.2 Validation of MBIM_CID_RADIO_STATE

This section contains test case that validates specifics of MBIM_CID_RADIO_STATE command.

CID_08 Validation of HwRadioState for Devices without a Hardware Radio Switch

This test verifies that HwRadioState field in MBIM_RADIO_STATE_INFO structure contains MBIMRadioOn if the device does not have a hardware radio switch.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.3.6#1: If the device does not specify MBIMCtrlCapsHwRadioSwitch the function must return MBIMRadioOn in HwRadioState field.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_CAPS" sequence has been executed successfully and MBIMCtrlCapsHwRadioSwitch bit is not set in ControlCaps field of the MBIM_DEVICE_CAPS_INFO structure returned in the MBIM_COMMAND_DONE response.

Test step(s):

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageType – set to 3d (MBIM_COMMAND_MSG)
   - MessageLength – set to this Encapsulated Command message length
   - TransactionId – set to old TransactionId + 1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 0d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
   - CID – set to 3d (MBIM_CID_RADIO_STATE)
   - CommandType – set to 0d (Query)
   - InformationBufferLength – set to 0
   - InformationBuffer – NULL
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that HwRadioState field in the MBIM_RADIO_STATE_INFO structure returned in the MBIM_COMMAND_DONE response contains 1 (MBIMRadioOn).

6.18.3 Validation of MBIM_CID_CONNECT

This section contains tests that validate the specifics of MBIM_CID_CONNECT command.

CID_09 Validation of Properly Setting IP Type

This test verifies that the function only activates the context which activation is requested by the host.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.12.1#2: On MBIM_CID_CONNECT set request the Host may specify an IP type to activate. If a value other than MBIMContextIPTypeDefault is specified, the function must only activate that context.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "Connect" sequence.
2. Send an NTB with a dummy IPv6 packet.
3. Verify that no “looped” packet has been received (i.e., the function ignores the ipv6 packet).

CID_10 Validation of MBIM_COMMAND_DONE for Set Request

This test verifies that an MBIM_COMMAND_DONE response is received in response to a set request.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.12.1#3: Functions must only send MBIM_COMMAND_DONE for MBIM_CID_CONNECT’s Set request after they have successfully activated or deactivated an IP data stream session, or detected an error.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "Connect" sequence.

CID_11 Validation of Using MBIM_SET_CONNECT' SessionId

This test validates SessionId field in the MBIM_CONNECT_INFO structure returned in the response to MBIM_CID_CONNECT command.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.12.1#4: Function must use the value in MBIM_SET_CONNECT' SessionId member when completing set requests.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute "Connect" sequence with SessionId set to 1d.
2. Verify that SessionId in the MBIM_CONNECT_INFO structure returned in the MBIM_COMMAND_DONE response is set to 1d.

CID_12 Validation of the Response to Deactivation Request in Case of a Non-Active Context

This test validates the function's behavior when the host submits a request to deactivate a non-active context.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.12.1#6: If the function receives a request to de-activate a context that is not currently activated, it shall respond with MBIM_STATUS_CONTEXT_NOT_ACTIVATED.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Execute steps 1 and 2 of "Connect" sequence with ActivationCommand set to 0d (MBIMActivationCommandDeactivate) and SessionId set to the ID of a session for which the context is not activated.
2. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_CONTEXT_NOT_ACTIVATED.

6.18.4 Validation of MBIM_CID_IP_CONFIGURATION

This section contains a test case that validates the specifics of MBIM_CID_IP_CONFIGURATION command.

CID_13 Validation of the Response in Case of a Non-Activated Context

This test validates the device's response in case when the context MBIM_CID_IP_CONFIGURATION refers to has not been activated.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.20.7: If the function receives MBIM_CID_IP_CONFIGURATION query with SessionId specifying a context that is not currently activated, it shall respond with MBIM_STATUS_CONTEXT_NOT_ACTIVATED.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.

Test step(s):

1. Send MBIM_COMMAND_MSG message using the following parameters:
   - MessageType – set to 3d (MBIM_COMMAND_MSG)
   - MessageLength – set to size of this Encapsulated command
   - TransactionId – set to old TransactionId + 1
   - TotalFragments – set to 1d
   - CurrentFragment – set to 0d
   - DeviceServiceId – set to a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT)
   - CID – set to MBIM_CID_IP_CONFIGURATION
   - CommandType – set to 0d (Query)
   - InfromationBufferLength – set to 60d
   - InformationBuffer (contains MBIM_IP_CONFIGURATION_INFO structure)
     - SessionId – set to the ID of a session for which the context is not activated.
2. Retrieve an MBIM_COMMAND_DONE response with TransactionId, DeviceSeviceId and CID from step 1.
3. Verify that the MBIM_COMMAND_DONE response has been returned with Status == MBIM_STATUS_CONTEXT_NOT_ACTIVATED.

6.18.5 Validation of MBIM_CID_DEVICE_SERVICES

This section contains a test case that validates the specifics of MBIM_CID_DEVICE_SERVICES command.

CID_14 Validation of CidCount

This test verifies the presence of a proper number of entries in the list of CIDs.

Assertion(s) used in the test:

[MBIM 1.0] - 10.5.29.1#2: There must be CidCount number of entries in the list of CIDs located in MBIM_DEVICE_SERVICE_ELEMENT structure.

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_SERVICES" sequence has been executed successfully.

Test step(s):

1. Verify that CidCount number of entries is present in the list of CIDs provided in the data buffer of each MBIM_DEVICE_SERVICE_ELEMENT structure returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_SERVICES command as a part of MBIM_DEVICE_SERVICES_INFO structure.

6.18.6 Validation of Mandatory CIDs

This section contains a test case that validates the support of the mandatory CIDs.

CID_15 Validation of Mandatory Functionality

This test validates the support of the mandatory CIDs.

Assertion(s) used in the test:

[MBIM 1.0] - 11.2: The mandatory to implement functionality comprises the following CIDs from the Basic Connectivity Service:

- MBIM_CID_DEVICE_CAPS
- MBIM_CID_SUBSCRIBER_READY_INFO
- MBIM_CID_RADIO_STATE
- MBIM_CID_PIN
- MBIM_CID_HOME_PROVIDER
- MBIM_CID_REGISTER_STATE
- MBIM_CID_SIGNAL_STATE
- MBIM_CID_CONNECT
- MBIM_CID_IP_CONFIGURATION_INFO
- MBIM_CID_DEVICE_SERVICES

- MBIM_CID_PACKET_SERVICE

Precondition(s):

1. "MBIM Open" sequence has been executed successfully.
2. "MBIM_CID_DEVICE_SERVICES" sequence has been executed successfully.

Test step(s):

1. Verify that MBIM_DEVICE_SERVICE_ELEMENT entry with DeviceServiceId == a289cc33-bcbb-8b4f-b6b0-133ec2aae6df (UUID_BASIC_CONNECT) has been returned in the MBIM_COMMAND_DONE response to the MBIM_CID_DEVICE_SERVICES command as a part of MBIM_DEVICE_SERVICES_INFO structure and the list of CIDs provided in the data buffer of this entry contains all the mandatory CIDs.