Media Agnostic Universal Serial Bus Specification

Release 1.0a

July 29, 2015

Scope of This Release

This document is the Release 1.0 of this specification.

Contributors

Will Harris	Advanced Micro Devices
Simon Black	Atmel Corporation
Chris Kelly	Atmel Corporation
Roel Peeters	Atmel Corporation
Ananthateerta Ashwini	Broadcom Corporation
Randal Erman	Broadcom Corporation
Gang Lu	Broadcom Corporation
Sandy (Alexander) MacInnis	Broadcom Corporation
Murat Mese	Broadcom Corporation
Payam Torab	Broadcom Corporation
Freeman Wang	Broadcom Corporation
Jing Wang	Broadcom Corporation
Hui Xu	Broadcom Corporation
Dan Ellis	DisplayLink Ltd.
Richard Petrie	DisplayLink Ltd.
Carlos Cordeiro	Intel Corporation
Marek Dabek	Intel Corporation
Kris Fleming	Intel Corporation
John Howard	Intel Corporation
Abdul Rahman Ismail	Intel Corporation
Oren Kedem	Intel Corporation
Maciej Kurczewski	Intel Corporation
Elad Levy	Intel Corporation
Guoqing Li	Intel Corporation
Wojciech Omilian	Intel Corporation
Venkatesh Rajendran	Intel Corporation
Bahareh Sadeghi (Editor)	Intel Corporation
Etan Shirron	Intel Corporation
Solomon Trainin	Intel Corporation
Rafal Wielicki	Intel Corporation
Xiaowen Lu	MCCI Corporation
Terry Moore	MCCI Corporation
Chris Yokum	MCCI Corporation
Chao-Chun Wang	MediaTek Inc.
James Yee	MediaTek Inc.
Randy Aull	Microsoft Corporation
Constantine Elster	Qualcomm Inc.
Xiaolong Huang	Qualcomm Inc.
Ali Raissinia	Qualcomm Inc.
Vamsi Samavedam	Qualcomm Inc.
Lochan Verma	Qualcomm Inc.
Xiaodong Wang	Qualcomm Inc.

Chiu Ngu	Samsung Electronics Co.
Huai-Rong Shao	Samsung Electronics Co.
Karthik Srinivasa Gopalan	Samsung Electronics Co.
Dmitry Cherniavsky	Silicon Image

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

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The motivation for the Media Agnostic (MA) Universal Serial Bus (USB) is to provide USB connectivity over medium other than USB, for example, wireless or IP links, while making maximal use of the existing USB infrastructure, present in the form of

- open application programming interfaces in every major operating system,
- USB class specifications by USB-IF device working groups,
- a wealth of open-source/OS specific drivers for USB classes, and
- vendor-specific device drivers for a multitude of USB vendor-specific devices.

For example, it is possible for a peripheral device vendor to reuse a device driver designed for high-speed USB 2.0 in conjunction with an MA USB device implementation that embeds a USB function through a high-speed interface (i.e., without a physical USB connection to the function), and achieve gigabit transfer rates.

1.2 Objective of the specification

This specification defines all protocol packets and associated behavior required to build interoperable MA USB host and MA USB device implementations that would ultimately enable devices from different vendors to interoperate in an open architecture while maintaining and leveraging the existing USB infrastructure (device drivers, software interfaces, etc.).

Special care has been taken to keep the number of new concepts and abstractions to a minimum, and instead build upon existing and proven mechanisms in USB wherever possible.

1.3 Scope of the document

This specification is primarily targeted to peripheral developers, operating system developers, and system OEMs. It can be used for developing new products and applications.

The primary target of the current version of the specification is support of Wi-Fi and WiGig media. It is expected that the future version(s) of this specification will address all the requirements for support of other media.

1.4 Document organization

Chapters 1 through 5 provide an overview for all readers. They describe the architectural elements of the MA USB protocol, namely the MA USB host and MA USB device, further architectural blocks within those elements, data and management interfaces to USB infrastructure and network interface, and data transfer models for all USB transfer types.

Chapters 6 through 9 describe the protocol packets and associated behavior in detail.

Chapter 6 serves as a reference for all MA USB packets (control and data).

Chapter 7 describes the MA USB device operation, including all control and data functions in support of the USB device integrated into the MA USB device. Also defined in Chapter 7 are MA USB device states, not to be confused with USB device states, which indicate the MA USB device availability to the MA USB host. Many of the MA USB host functions are defined in the course of defining the device framework. Chapter 8 defines the remaining functions, most importantly, starting an MA USB session, and managing the MA USB session to each MA USB device.

Chapter 9 describes the MA USB hub, which is an MA USB device that integrates a USB hub. An MA USB hub performs extra functions to enable enumeration and data transfer for USB devices connected to its integrated hub.

All key parameters of the MA USB protocol are defined in Chapter 10. Parameters fundamental to interoperability are defined as constants. All compliant implementations are required to have the same setting for these parameters. Other parameters, labeled as protocol variables, are defined together with a recommended setting. Implementations may choose different values for these parameters depending on the application and design specifics.

2 Normative references

The following referenced documents are indispensable for the application of this standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

[IEEE 802.11]	IEEE Std 802.11™-2012--IEEE Standard for Information Technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications
[USB 2.0]	Universal Serial Bus Specification, Revision 2.0
[USB 3.1]	Universal Serial Bus Specification, Revision 3.1
[WMC 1.1]	CDC Subclass for Wireless Mobile Communication Devices, Revision 1.1
[OTG&EH3]	USB On-The-Go and Embedded Host Supplement to the USB 3.0 Specification, Revision 1.1

3 Definitions, acronyms and abbreviations

3.1 Definitions

For the purposes of this standard, the following terms and definitions apply. See normative references for terms that are not defined in this clause.

EP handle: A 16-bit number uniquely identifying a USB endpoint within an MA USB device. The endpoint belongs to a USB device (including a USB hub) either integrated into the MA USB device or connected through a wired USB connection to the hub integrated into the MA USB device. The combination of MA USB device address and EP handle uniquely identifies a USB device endpoint within an MA USB Service Set.

Universal Serial Bus (USB) address: A unique address in the form of a seven-bit value assigned by the USB system software to the USB device.

USB Device Handle: A 16-bit number uniquely identifying a USB device behind an MA USB device; also called device handle for short.

Media Agnostic Universal Serial Bus (MA USB): The Protocol Adaptation Layer (PAL) defined in this specification, which enables connectivity between a USB host and one or more USB devices, including USB hubs, over medium other than USB, including wireless and IP links.

MA USB device: An MA USB architectural element that integrates a USB device, and manages USB transfers targeting the USB device over a network connection. The integrated device may be connected through wired USB (USB cable, USB chip-to-chip interconnect, etc.) or other technologies, but through the MA USB device, it is presented to the host system as a USB device compliant with Revision 2.0 or 3.1 of the USB specification.

MA USB device address: An 8-bit number uniquely identifying an MA USB device within an MA USB Service Set.

MA USB hub: An MA USB device that integrates either a USB 2.0 hub or a USB 3.1 hub (which combines a USB 2.0 hub and an Enhanced SuperSpeed hub). It provides physical downstream facing USB ports to attach removable or non-removable USB devices. An MA USB hub performs all USB hub functions for control and management of its downstream facing USB ports. It may or may not have a physical upstream port, but to the USB host system it enumerates as a USB 2.0 hub or, if the integrated device is a USB 3.1 hub, as a USB 2.0 hub and an Enhanced SuperSpeed hub. An MA USB hub is also capable of scheduling and completing USB 2.0 or USB 3.1 transactions targeting USB devices connected to the hub downstream ports.

MA USB host: An architectural element of the MA USB PAL that includes a physical link interface and USB host logic as defined in USB Specifications.

MA USB Service Set (MSS): The collection of an MA USB host and all MA USB devices it manages. Each MSS defines an independent addressing domain, i.e., an MA USB device address is unique within the scope of the MSS to which the MA USB device belongs.

MA link: The end-to-end connection between an MA USB host and an MA USB device or an MA USB hub. An MA link may consist of multiple network segments.

3.2 Acronyms and abbreviations

ACK	Acknowledgment
BSS	Basic Service Set
DSAP	Destination Service Access Point
DWORD	Double Word
EP	Endpoint
FS	Full-Speed USB
GB	Gibibyte (1,073,741,824 bytes)
HID	Human Interface Device
HS	High-Speed USB
HSIC	High Speed Inter-Chip
IAD	Interface Association Descriptor
KB	Kibibyte (1,024 bytes)
LLC	Logical Link Control layer
LS	Low-Speed USB
MA USB	Media Agnostic Universal Serial Bus
MAC	Medium Access Control
MB	Mebibyte (1,048,576 bytes)
MGT	MA USB Global Time
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
MSS	MA USB Service Set
MTU	Maximum Transmission Unit
OS	Operating System
PAL	Protocol Adaptation Layer
PBSS	Personal Basic Service Set
PDU	Protocol Data Unit
PHY	Physical layer
SAP	Service Access Point
SNAP	Subnetwork Access Protocol
SSAP	Source Service Access Point
SSID	Service Set Identifier
TID	Traffic Identifier
USB	Universal Serial Bus
USBDI	USB Driver Interface (OS-specific)
WHCM	Wireless Handset Control Module

4 Architectural overview

4.1 Architectural elements

MA USB protocol functions are asymmetrically split between a host element, responsible for creating the topology and at the center of all data flows, and one or more MA USB devices representing peripheral devices. The collection of an MA USB host and MA USB devices it manages is called an MA USB Service Set (MSS).

4.1.1 MA USB host

An MA USB host performs USB host functions as defined by the USB 2.0 and 3.1 specifications, except those that are specific to the USB medium, as well as extra functions to manage the MA USB devices and MA link interface. Specifically, an MA USB host manages the

- MA USB service set and communication sessions to MA USB devices
- MA USB devices, including capability exchange with MA USB devices and configuring them
- Addition and removal of USB devices integrated or plugged into MA USB devices, including configuring each USB device and its possible alternate interfaces
- USB transfers targeting each USB device, including flow control

MA USB host performs all USB transfers over the MA link interface, and in accordance with frame formats and transmission rules of the MA link.

Figure 1 illustrates the basic building blocks of an MA USB host. The USB host logic block performs USB host functions defined by the USB 2.0 and USB 3.1 specifications, except those that involve the physical USB medium. The MA USB host PAL, defined in this specification, manages the MA USB devices and transport of USB payload over the MA link.

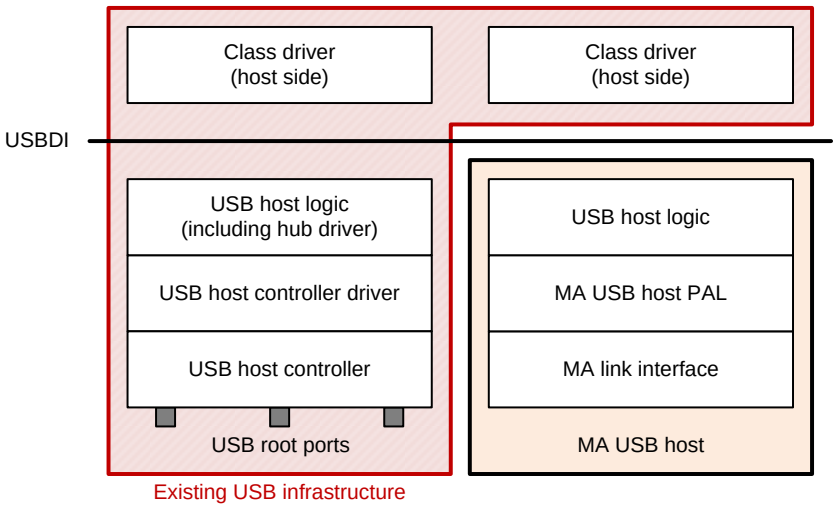


Figure 1—Building blocks of an MA USB host and relationship to existing USB infrastructure

Also shown in Figure 1 is how MA USB host building blocks fit into existing USB infrastructure. An MA USB host provides the same abstraction of the USB bus that the USB driver in an Operating System provides, plus possibly additional functions to control the MA link interface. As a result, existing software drivers for various USB classes (Mass Storage, Human Interface Devices (HID), Audio

Device, etc.) can run over the MA USB protocol layer with no change. In particular, all MA USB abstractions are strictly "PAL-to-PAL" and are not exposed to these drivers.

When defining the MA USB topology and applications, the term "MA USB host" is also used to refer to the computing platform (e.g., a personal computer) that houses the MA USB host defined in this section, as well as all class drivers and application software provided through the Operating System or USB product vendors.

4.1.2 MA USB device

An MA USB device is the counterpart to the MA USB host that enables remote connectivity to one or more peripheral devices. Specifically, an MA USB device manages the

- MA USB service set and communication sessions to the MA USB host
- Addition and removal of USB devices, integrated or plugged into the MA USB device, including USB device reset and address assignment

MA USB device performs all USB transfers over the MA link interface, and in accordance with frame formats and transmission rules of the MA link.

Figure 2 illustrates the basic building blocks of an MA USB device. The USB device logic block performs USB device functions defined by the USB 2.0 or USB 3.1 specification, except those that involve the physical USB medium. The MA USB device PAL, defined in this specification, manages the transport of USB payload over the MA link interface.

NOTE —While this specification frequently refers to host and device PALs, it does not recognize PALs as valid standalone elements. An MA USB host PAL or device PAL is assumed to be integrated with physical elements such as a wireless radio and USB endpoints to define an MA USB host or MA USB device. The terms "host PAL" and "device PAL" are used when the PAL distinction is important, for example, when specifying PAL-to-PAL timeout values for MA USB transfer protocols. When the distinction is trivial, or when specifying a behavior for a physical device with interface to an MA Link, the inclusive terms "host" or "device" is used.

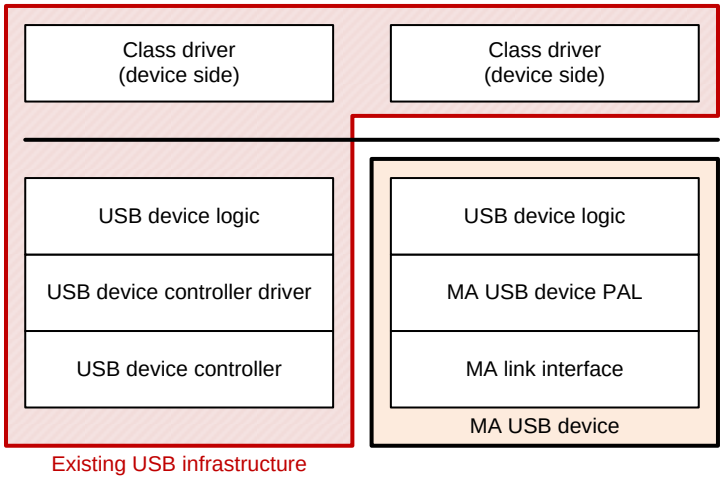


Figure 2—Building blocks of an MA USB device and relationship to existing USB infrastructure

Also shown in Figure 2 is how MA USB device building blocks replace or complement the existing USB infrastructure in a USB peripheral device. Similar to the MA USB host case, existing device-side class drivers can run over the MA USB protocol layer with no change. Also similar to the MA USB host case, all MA USB abstractions are strictly "PAL-to-PAL" and are not exposed to these drivers.

The interface between the USB device logic block and device-side class drivers is application-specific and outside the scope of this specification. The interface between the MA USB device PAL and the USB device logic is platform-specific.

An MA USB device manages, with some autonomy, an integrated USB device. The integrated device may be removable or non-removable, and may be connected through wired USB (USB cable, USB chip-to-chip interconnect, etc.) or other technologies, but through the MA USB device, it is presented as a USB device compliant with Revision 2.0 or 3.1 of the USB specification. It can belong to any USB class, including the Hub class. The integrated USB device is said to operate "behind" the MA USB device, and the MA USB device is said to "host" it, although many (but not all) USB host functions are performed inside the MA USB host. In summary, a USB device integrated into an MA USB device (or hosted by the MA USB device) may be logical or physical, and may be removable or non-removable.

In an MA USB device other than an MA USB hub, the MA USB protocol capabilities may be profiled according to the function of the integrated USB device. For example, an MA USB HID device may support the MA USB protocol functions pertaining to interrupt transfers and not isochronous or bulk transfers.

When defining the MA USB topology and applications, the term "MA USB device" is also used to refer to the computing platform (e.g., portable electronic device) that houses the MA USB device defined in this section, as well as all device-side class drivers and application software required to implement a peripheral device.

4.1.3 MA USB hub

An MA USB hub is an MA USB device that integrates a USB 2.0 hub or a USB 3.1 hub (which combines a USB 2.0 hub and an Enhanced SuperSpeed hub). It provides physical downstream facing USB ports to attach removable or non-removable USB devices. An MA USB hub performs all USB hub functions for control and management of its downstream facing USB ports. It may or may not have a physical upstream port, but to the USB host system it enumerates as a USB 2.0 hub, or, if the integrated device is a USB 3.1 hub, as a USB 2.0 hub and an Enhanced SuperSpeed hub. As shown in Figure 3 an MA USB hub also contains a logical entity, USB transaction engine, responsible for performing transactions on a USB bus (equivalent functionalities of a USB controller, as requested by the MA USB host).

In addition to MA USB device functions, an MA USB hub manages

- Attachment and removal of wired USB devices on downstream facing ports
- Scheduling and completion of USB transactions targeting USB devices connected to the downstream facing USB ports
- USB address assignment

Additionally, an MA USB hub provides power to attached USB devices.

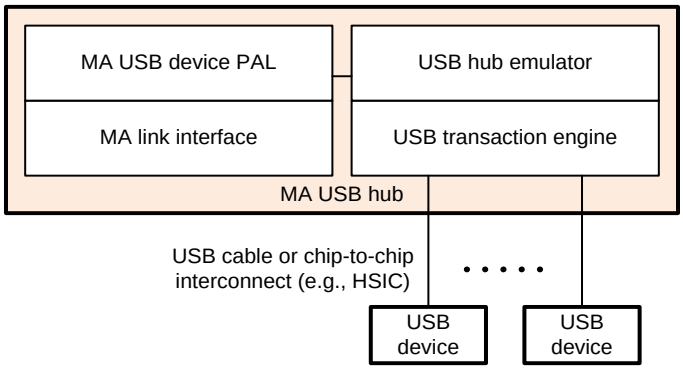


Figure 3—Building blocks of an MA USB hub

An MA USB hub not only manages the USB transfers that target its integrated USB 2.0 or USB 3.1 hub, but also manages the USB transfers targeting all USB devices downstream of the integrated hub. An MA USB hub is the only type of MA USB device that integrates a USB hub, and is the only type of MA USB device that manages transfers targeting multiple USB devices.

When defining the MA USB topology and applications, the term "MA USB hub" is also used to refer to the computing platform that houses the MA USB hub defined in this section. An example of such a platform is shown in Figure 4 where an MA USB hub hosts three USB devices, with two of them physical and one logical, and one of them removable and two non-removable.

NOTE — Throughout this specification, the term MA USB device also refers to MA USB hubs, unless stated otherwise.

NOTE — An MA USB hub is the only type of MA USB device that can manage USB transfers targeting multiple USB devices. All other MA USB device types manage transfers to exactly one integrated USB device.

NOTE — An MA USB hub need not expose any physical downstream ports.

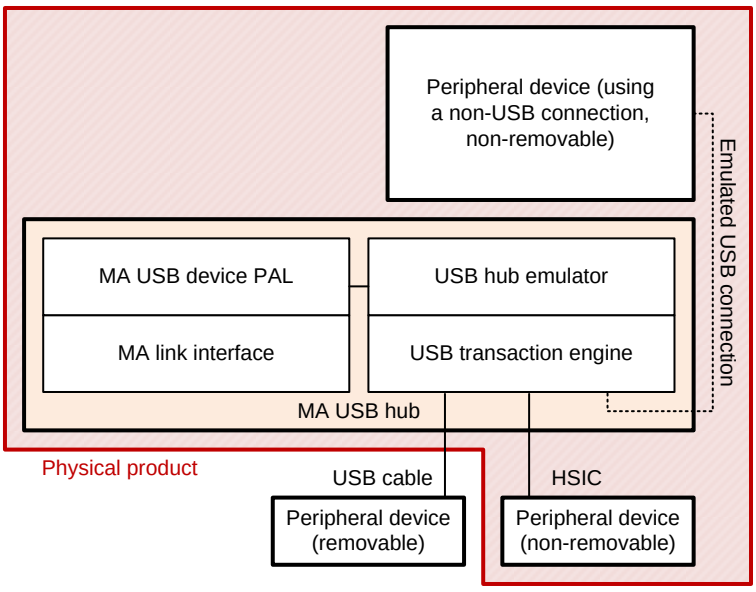
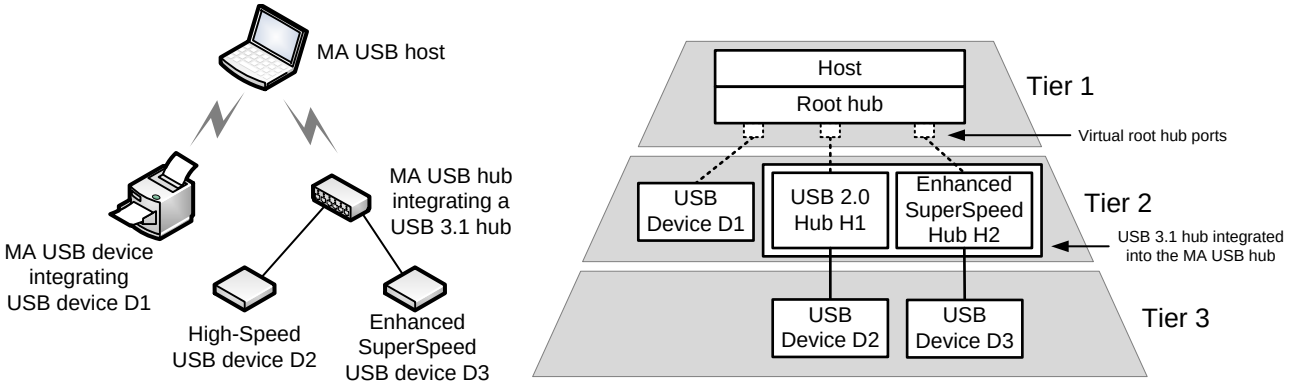


Figure 4—Example MA USB hub hosting multiple USB devices

4.2 USB topology

From the host system perspective, each MA USB host instance introduces a separate virtual bus. The MA USB host PAL introduces a USB host and root hub at Tier 1 of the USB hierarchy [USB 2.0]. The

USB device integrated into an MA USB device (including the physical or virtual hub integrated into an MA USB hub) falls in Tier 2 of the hierarchy. USB devices behind an MA USB hub fall in Tier 3. Figure 5 illustrates an example of an MA USB system and its equivalent USB topology.



(a) Example MA USB Service Set (MSS) (b) Equivalent USB topology for the MSS

Figure 5—MA USB Service Set and equivalent USB topology

4.3 MA USB communication model

The MA USB communication model assumes a logical separation of management and non-management (i.e., control and data) packets. As shown in Figure 6, all management packets are assumed to be exchanged over a single (logical) management channel, and all control and data packets are assumed to be exchanged over one or more (logical) data channels. All channels are bidirectional. The physical realization of management and data channels is implementation specific, but channels are expected to meet specific timings, on top of which protocol operation details such as timeouts are defined.

NOTE — MA USB channels terminate on host and device PALs. The latency associated with each MA USB channel comprises the latencies across all networking layers between the host and device PALs.

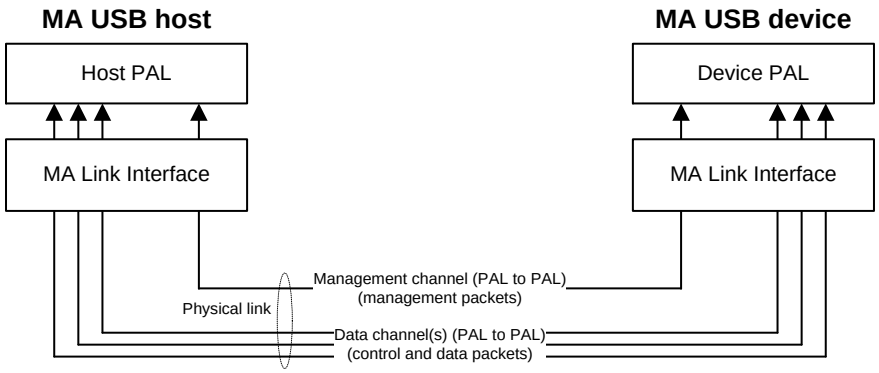


Figure 6—MA USB communication model

An MA USB management packet released to a management channel is expected to be received at the target MA USB device or host PAL no later than aManagementChannelDelay after its release time. An MA USB control or data packet released to a data channel is expected to be received at the target MA USB device or host PAL no later than aDataChannelDelay after its release time.

NOTE — All MA USB protocol constant names start with the "a" prefix.

Figure 7 illustrates the MA USB management and data channel latencies.

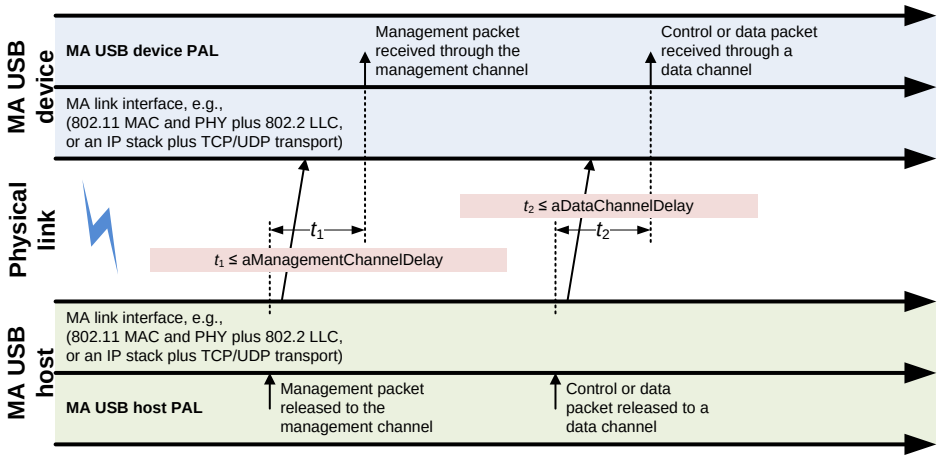
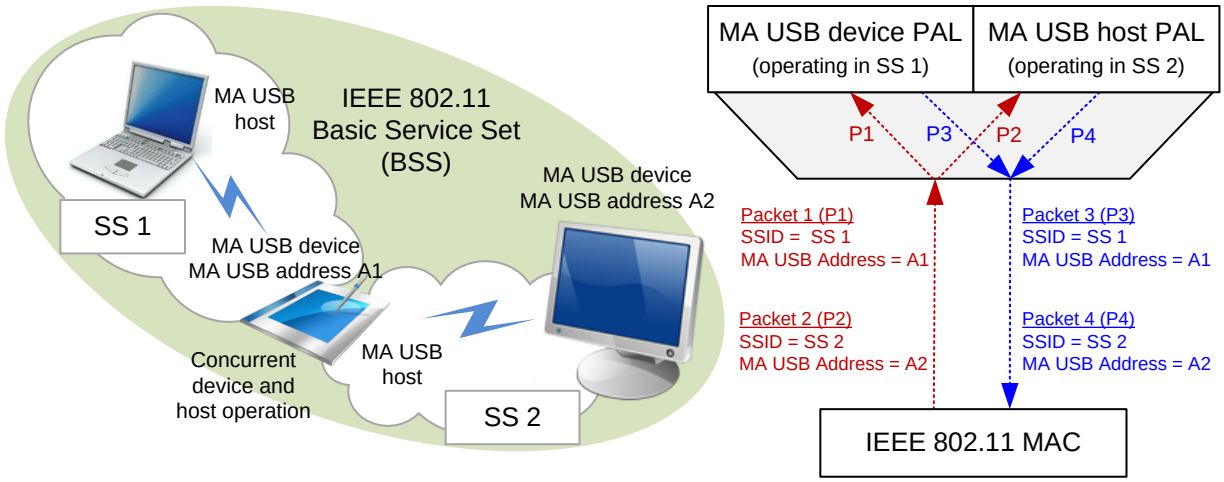


Figure 7—MA USB Management and data channel latencies

4.4 MA USB addressing

At the center of the MA USB addressing scheme is the concept of the MA USB Service Set (MSS): A MSS is a group representing an MA USB host and all MA USB devices under its control. Each MA USB device manages the transfers targeting the USB device it integrates, and in the case of an MA USB hub the transfers targeting the hub it integrates, as well as the transfers targeting all USB devices downstream from the integrated hub. The bus-level addresses of all these USB devices are locally assigned by the MA USB device, and are unique within the scope of that MA USB device. Thus, the number of USB devices operating in an MA USB Service Set is practically limited by the number of MA USB devices (including hubs) in the MSS, and the resources (computing power, airtime) available to those MA USB devices and to the MA USB host.

Multiple MA USB elements (e.g., an MA USB device and an MA USB host, or two MA USB device instances) may share a common MA link interface (e.g., a common wireless radio). In addition, the MA USB architecture enables operation over a range of network technologies (e.g., IEEE 802.11 and Internet Protocol) with different addressing and multiplexing schemes. Figure 8 shows an example of concurrent operation in an IEEE 802.11 Basic Service Set (BSS), where a single network element is operating as an MA USB device and an MA USB host at the same time, and using a common 802.11 radio. The received MA USB packets are delivered to the proper MA USB PAL instance (a host PAL and a device PAL in this case) using the SSID and the Device Address fields in the packet. A similar example is when a single network element runs two instances of the MA USB device PAL in parallel for concurrent operation in two MA USB Service Sets.



(a) Example of a MA USB dual-role element with concurrent operation as a MA USB host and a MA USB device over a common IEEE 802.11 radio

(b) Routing of MA USB packets to the correct MA USB device or host PAL instance inside the dual-role element

Figure 8—Concurrent MA USB device and host operation in an IEEE 802.11 BSS

4.4.1 MA USB device address

Within a MSS, each MA USB device is uniquely identified by an 8-bit MA USB address assigned by the MA USB host. There is no structure to the MA USB device address. Address values of 0x00 (unassigned) and 0xFF (any device) are reserved for protocol operation. Thus, each MSS can include up to 254 MA USB devices.

NOTE — Address value of 0x00 (unassigned) is not used in this version of specification.

There is no MA USB address assigned to the MA USB host. All MA USB packets are originated by or targeted at the MA USB host, which makes including an MA USB host address field in MA USB packets redundant; instead, a single bit in each MA USB packet indicates whether the packet is originated by or targeted at the MA USB host. See Section 6.2.1.2 for details.

4.4.2 Device handle

Each USB device behind an MA USB device is uniquely identified by a locally-assigned 16-bit identifier, referred to as the device handle. Each device handle is unique within the scope of an MA USB device. Device handles are unstructured and carry no specific format. Device handle values of 0x0000 (unassigned) and 0xFFFF (any device handle) are reserved for protocol operation.

4.4.3 Endpoint handle

Within an MA USB device, each USB endpoint is uniquely identified by a locally-assigned 16-bit identifier referred to as the endpoint (EP) handle. EP handles follow a specific format defined in Section 6.2.1.5.

4.4.4 Container ID

An MA USB device may be able to connect over USB 2.0 or 3.1 as well as MA links. It is also possible that multiple different types of MA links are supported. To ensure that the MA USB device may be detected as the same device regardless of whether it is connected over USB 2.0 or 3.1 link or an MA

link, MA USB devices which support connectivity over USB 2.0 or USB 3.1 in addition to MA links shall support the Container ID descriptor as defined in [USB 3.1]. The same container ID shall be provided by an MA USB device over all USB and MA links supported.

4.5 Media dependent functions

While the MA USB PAL functions are media agnostic, the end-to-end operation of MA USB relies on functions of the network providing the connectivity between the MA USB host and the MA USB device. These media specific functions are specified in this section. Additionally, throughout the specification media specific descriptions are provided in the form of MEDIA DEPENDENT NOTEs.

The current version of this specification defines two modes of operation for MA USB,

- Operation over a non-extended 802.11 basic service set (802.11 mode): MA USB host and device belong to the same 802.11 BSS, IBSS or PBSS. MA USB host and device PALs are direct clients of the IEEE 802.2 LLC sublayer [IEEE 802.2]. All protocol packets are encapsulated inside SNAP PDUs and presented to the 802.11 MAC SAP as 802.11 MSDUs.
- Operation over IP protocol suite (IP mode): MA USB host and device are separated by an IP network. MA USB host and device PALs are direct clients of the TCP (UDP for isochronous streams) transport layer. All protocol packets are carried inside IP datagrams and delivered through TCP (UDP for isochronous streams) connections.

Figure 9 illustrates the two MA USB modes of operation.

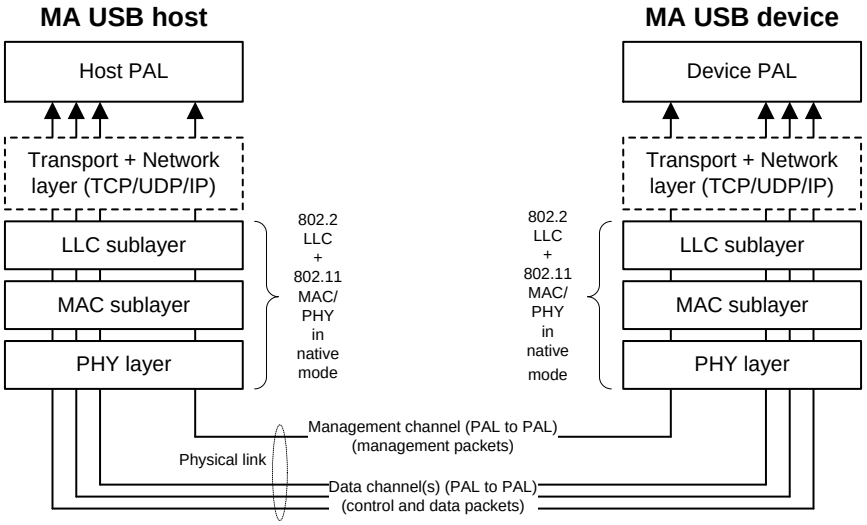


Figure 9—MA USB protocol hierarchy

4.5.1 Relation of MA USB addressing to network addressing

The MA USB protocol makes use of PAL-level device addresses that are independent of the addresses used at lower network layers. Specifically, there is no dependence between the MA USB address through which an MA USB PAL instance is reached, and the network address of the MA USB device or host containing that PAL instance.

Figure 10 illustrates the MA USB addressing scheme and its relation to network-level addresses. At the PAL level, each MA USB device PAL instance is uniquely identified by the MA USB device address within a MSS, or by the (SSID, MA USB device Address) tuple within the underlying network broadcast domain. At the network level, target addresses change to IP addresses in IP mode, or full LLC

addresses in 802.11 mode, where each full LLC address is the logical concatenation of an EUI-48 MAC address and an LLC Source or Destination Service Access Point (LLC SSAP/DSAP).

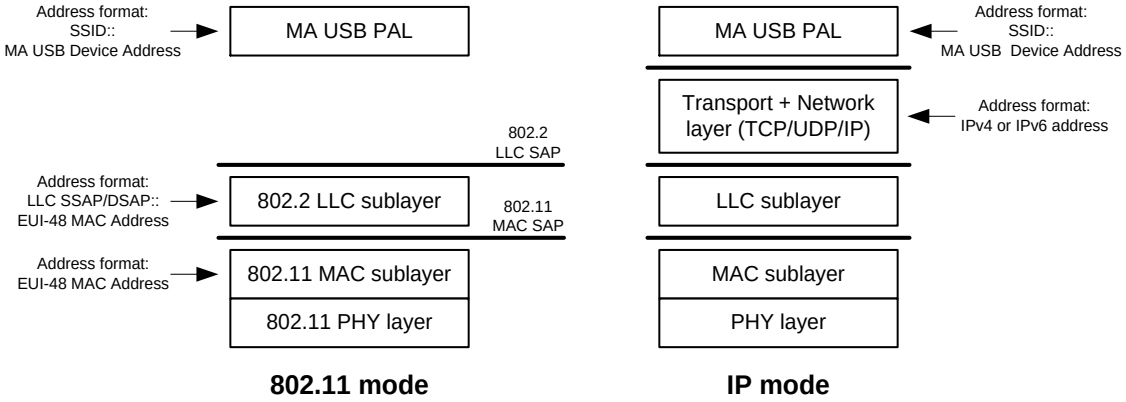


Figure 10—Network addressing in 802.11 and IP modes

An *address resolution* function inside the MA USB host PAL associates the MA USB device address of each MA USB device PAL, and the network address through which the device PAL can be reached. Similarly, an address resolution function inside each MA USB device PAL derives the network address through which the MA USB host PAL can be reached.

The network-independent addressing scheme in MA USB makes it possible to continue the protocol operation as the network address or even the network interface changes. The MA USB Ping protocol (Section 5.2.2) enables an MA USB PAL instance to discover the network address through which a peer MA USB PAL instance can be reached.

NOTE — While MA USB PAL operation may persist across network address or interface changes, MA USB transfers may timeout as a result of network transients. Examples of network-level address changes are an 802.11 MAC address change in 802.11 mode as a result of radio switch (e.g., switching from 60GHz to 5GHz), or an IP address change in IP mode resulting from dynamic address allocation.

4.5.2 MA USB PAL identification

4.5.2.1 Identification in 802.11 mode

Identification of MA USB protocol packets in 802.11 mode is out of scope of this specification.

4.5.2.2 Identification in IP mode

Packet identification in IP mode is based on TCP/UDP port numbers. The MA USB management channel is established over a TCP connection dedicated to MA USB, and therefore all MA USB management packets received through the TCP port associated with the management channel are identified through the receive port number. MA USB control and data packets may be received through the same dedicated TCP connection or other dedicated TCP connections (and for isochronous streams through dedicated UDP connections). In each case, the receive port number is used to identify the MA USB packets.

4.5.3 Network requirements

MA USB expects the network to provide secure connections (i.e., authentication, authorization, confidentiality, and data integrity) for all communications between an MA USB host and the MA USB devices and hubs in an MSS. In addition it is expected that the network will provide reliable and in-order delivery for all MA USB packets other than isochronous data packets. When operating in 802.11 mode, MA USB assumes *controlled* reliability over the MA USB link: Each dropped (lost) MAC-level

protocol data unit (MPDU) is retransmitted in a controlled manner based on MAC-level settings such as MSDU lifetime. Reliability settings in the lower layers for MA USB packets are application and implementation-dependent, and can range from no retransmission to practically unlimited number of retransmissions. In addition, MA USB utilizes PAL-level retries to recover from link loss (including MAC and PHY failures), which applies to a portion of or to an entire *MA USB transfer*, and is repeated for a specified number of times that depends on the transfer mode.

4.5.3.1 Requirements for 802.11 mode

This specification does not introduce any network requirements for 802.11 mode.

4.5.3.2 Requirements for IP mode

MA USB operation in IP mode makes use of TCP and UDP transport protocols. Support for TCP protocol is mandatory. For isochronous streams support for UDP protocol is mandatory.

4.5.3.3 Media specific protocol constants

Table 70 provides the list of MA USB protocol constants, including media dependent constants. The values of the media dependent constants are out of scope of this specification and need to be specified for each specific medium.

4.5.4 Device discovery

4.5.4.1 Device discovery in 802.11 mode

The discovery of MA USB capable devices in the 802.11 mode is out of scope of this specification.

4.5.4.2 Device discovery in IP mode

Discovery of MA USB enabled devices in the IP mode is outside the scope of the current version of this specification.

4.5.5 Packetization

4.5.5.1 Packetization in 802.11 mode

When operating in 802.11 mode, each MA USB packet is carried as a single 802.11 MAC service data unit (MSDU), i.e., there is no MA USB segmentation and reassembly function defined across MAC service data units. In particular, MA USB packets are limited in size by the maximum MSDU size for the particular 802.11 MAC and PHY that carry those packets.

4.5.5.2 Packetization in IP mode

When operating in IP mode, all MA USB packets other than isochronous data packets are delivered as byte streams over TCP connections, and therefore are not constrained by the network MTU size. Isochronous data packets are carried in UDP datagrams, and are limited in size by the payload available to each UDP datagram.

5 Data flow model

5.1 Communication flow

MA USB retains many of USB concepts such as endpoints, pipes and transfer types. Refer to [USB 2.0] for a full description of these concepts. The basic data flow starts from a USBDI-level read or write request over a pipe interface associated with a USB endpoint, which translates into one or more MA USB transfers. Each MA USB transfer typically generates several MA USB protocol data units (PDUs), or packets.

On the MA USB device side, each MA USB data packet corresponds to possibly several USB transactions on a wired USB segment terminating on a target USB device.

5.2 Protocol overview

5.2.1 Packet exchange

5.2.1.1 Management packet exchange

MA USB devices are managed through MA USB packets of type 0 (Management type). All management packets are transmitted over the management channel, and follow a request-response format. For convenience, in the context of a request-response management packet exchange, the MA USB PAL instance that is transmitting the request packet is referred to as the requesting PAL, and the MA USB PAL instance that is expected to transmit the response packet is referred to as the responding PAL.

> NOTE — In certain cases a response packet may be unnecessary or optional. For example, a Synchronization Request (SynchReq) packet (Section 6.3.40) may set the Response Required field to 0 to indicate an optional response.

Unless otherwise specified, a responding PAL shall release the response packet to the management channel within aManagementResponseTime from the moment it receives the corresponding request packet over the management channel.

> NOTE — MA USB PAL timings do not include management or data channel latencies; the equivalent timings for an MA USB host or device (which in addition to a PAL instance include interfaces to management and data channels) include management or data channel latencies as appropriate. For example, the above PAL timing requirement is equivalent to the response packet appearing over the medium within aManagementResponseTime + aManagementChannelDelay from the moment the corresponding request packet is received over the medium, or within aManagementResponseTime + 2×aManagementChannelDelay from the moment the corresponding request packet is released to the management channel.

Unless otherwise specified, if a requesting PAL does not receive a response packet by aManagementRequestTimeout after it has successfully submitted the request packet to the management channel, it shall retransmit the request packet by releasing another copy of the request packet to the management channel, keeping all packet fields the same as those in the original request packet, except the Retry field, which shall be set to1. If necessary, the requesting PAL shall repeat transmitting the request packet for a number of times not exceeding aManagementRetries. If the corresponding response packet is not received after aManagementRetries retries and the request packet is not a PingReq packet (Section 6.3.34), the requesting PAL may start the Ping protocol (Section 5.2.2), and if the Ping protocol is successful, the requesting PAL may attempt to transmit the request packet again up to the same maximum number of times attempted before the Ping protocol. If the Ping protocol fails, or if the requesting PAL decides not to exercise the Ping protocol, the requesting PAL shall transition to the Session Down state (Section 8.1.1.1) and inform the lower layers of the session state transition.

NOTE — Number of retries does not include possible local retries before the request packet is successfully released to the management channel. An example of a local retry is resubmitting the request packet to a local transmission queue after observing the queue to be full.

MEDIA DEPENDENT NOTE — Section 5.2.4.2 of [IEEE 802.11] lists more possible reasons for local failures in 802.11 mode.

A responding PAL shall respond to a request packet that has the Retry field set to 1, even if it has responded to an earlier instance of the request packet. The responding PAL is not required to repeat any action in response to the duplicate request packet other than retransmitting the corresponding response packet.

Figure 11 illustrates the timings related to exchange of management packets.

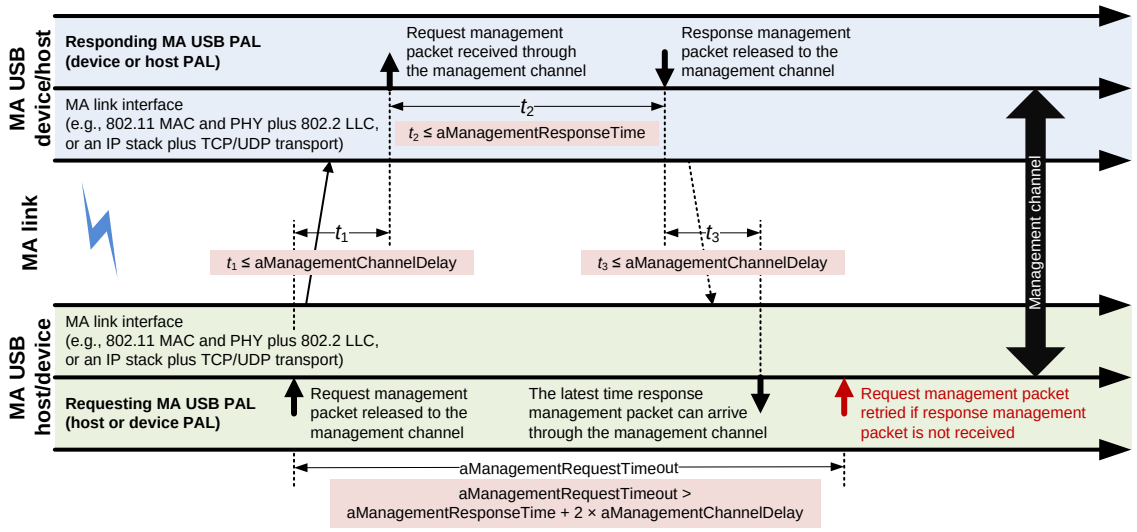


Figure 11—Default timings for management packet exchange

5.2.1.2 Data packet exchange

MA USB transfers are executed through MA USB packets of type 2 (Data type). All data packets are transmitted over one or more data channels.

MA USB transfers generally involve several data packets. The allowable latencies between these packets, and the behavior associated with potential timeouts are more complex than a simple request-response management exchange; transfer timings and associated timeout behaviors are specified in Sections 5.4, 5.5 and 5.10.

There are protocol conditions during a non-isochronous MA USB transfer where two data packets can be identified as having a request-response relationship, with the MA USB PAL transmitting the request packet (the requesting PAL) expecting an immediate response packet from the target MA USB PAL (the responding PAL). In the absence of a timely response packet, the requesting PAL needs to take corrective actions such as retransmitting the request packet or initiating the Ping protocol. These protocol conditions, referred to as *immediate exchanges*, are described in Sections 5.4 and 5.5. The default timings and retransmission behavior for immediate exchanges are described next.

NOTE — An example of a data packet pair that may form an immediate exchange is a Transfer Request (TransferReq) and a Transfer Response (TransferResp) packet. Another example is a Transfer Response (TransferResp) and a Transfer Acknowledgment (TransferAck) packet. Not all data packet exchanges are immediate. See sections 5.4 and 5.5 for details.

Unless otherwise specified, a responding PAL in an immediate response exchange shall release the response packet to the assigned data channel within aTransferResponseTime from the moment it receives the corresponding request packet over the assigned data channel.

NOTE — MA USB PAL timings do not include management or data channel latencies; the equivalent timings for an MA USB host or device (which in addition to a PAL instance include interfaces to management and data channels) include management or data channel latencies as appropriate. For example, the above PAL timing requirement is equivalent to the response packet appearing over the medium within aTransferResponseTime + aDataChannelDelay from the moment the corresponding request packet is received over the medium, or within aTransferResponseTime + 2×aDataChannelDelay from the moment the corresponding request packet is released to the assigned data channel.

Unless otherwise specified, if a requesting PAL in an immediate exchange does not receive a response packet by aTransferTimeout after it has successfully submitted the request packet to the assigned data channel, it shall retransmit the request packet by releasing another copy of the request packet to the assigned data channel, keeping all packet fields the same as those in the original request packet, except the Retry field, which shall be set to 1. If necessary, the requesting PAL shall repeat transmitting the request packet for a number of times not exceeding a protocol constant that depends on the transfer type, specifically, aControlTransferRetries, aBulkTransferRetries and aInterruptTransferRetries for control, bulk and interrupt transfers, respectively. If the corresponding response packet is not received after the maximum number of retries, the requesting PAL shall start the Ping protocol (Section 5.2.2) to verify and possibly re-establish network connectivity with the responding PAL, unless the requesting PAL is an MA USB device PAL, in which case starting the Ping protocol is optional. If the Ping protocol is successful the requesting PAL may still decide to fail the transfer and report the appropriate USBDI error indicating the transfer failure to the application, or it may attempt to transmit the request packet again up to the same maximum number of times attempted before the Ping protocol. If the Ping protocol fails, the requesting PAL shall report the appropriate USBDI error indicating the transfer failure to the application, move to Session Down state (Section 8.1.1.1), and inform the lower layers of the transition of the session to the Session Down state.

NOTE — In certain cases data packets may carry updated field values when retried; the Retry field is set to 0 in these packets. See Sections 5.4 and 5.5 for details.

NOTE — Number of retries does not include possible local retries before the request packet is successfully released to a data channel. An example of a local retry is resubmitting the request packet to a local transmission queue after observing the queue to be full.

NOTE — A requesting MA USB device PAL may choose not to exercise the Ping protocol and wait indefinitely for a packet from the MA USB host PAL, or follow another implementation-dependent behavior.

NOTE — It is possible that a responding PAL in a transient or corrupted state fails to respond to a request data packet but responds to a PingReq packet; a requesting PAL may or may not tie the status of an individual transfer to the Ping protocol outcome.

NOTE — MA USB host and device PAL implementations should report transfer failures to the application in a timely manner.

A responding PAL shall respond to a request packet that has the Retry field set to 1, even if it has responded to an earlier instance of the request packet. The responding PAL is not required to repeat any action in response to the duplicate request packet other than retransmitting the corresponding response packet.

Figure 12 illustrates the immediate exchange default timings.

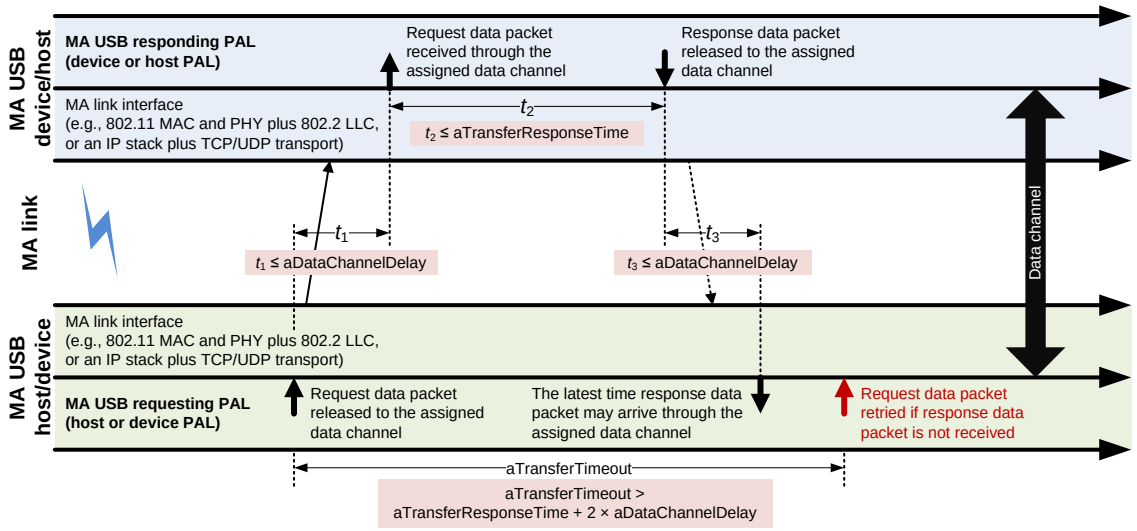


Figure 12—Default timings for data packet immediate exchange

In addition to immediate exchange, there are protocol conditions during a non-isochronous MA USB transfer where a transmitting PAL is expected to transmit data packets faster than a minimum rate, or the receiving PAL may take inquiring actions to verify the transfer is still alive. These protocol conditions are described in Sections 5.4 and 5.5.

Unless otherwise specified, a transmitting PAL that is expected to keep a transfer alive shall release successive data packets to the assigned data channel no more than aTransferRepeatTime apart.

Unless otherwise specified, a receiving PAL that expects a transfer to be kept alive shall start an inquiring action such as releasing another transfer request packet to the assigned data channel if it does not receive a new data packet by aTransferKeepAlive after it receives the last data packet over the assigned data channel.

NOTE — The default timeout period of aTransferKeepAlive can dynamically increase to a multiple of aTransferKeepAlive during an IN transfer. See Section 5.4 for details.

Figure 13 illustrates the default timings to keep an MA USB transfer alive.

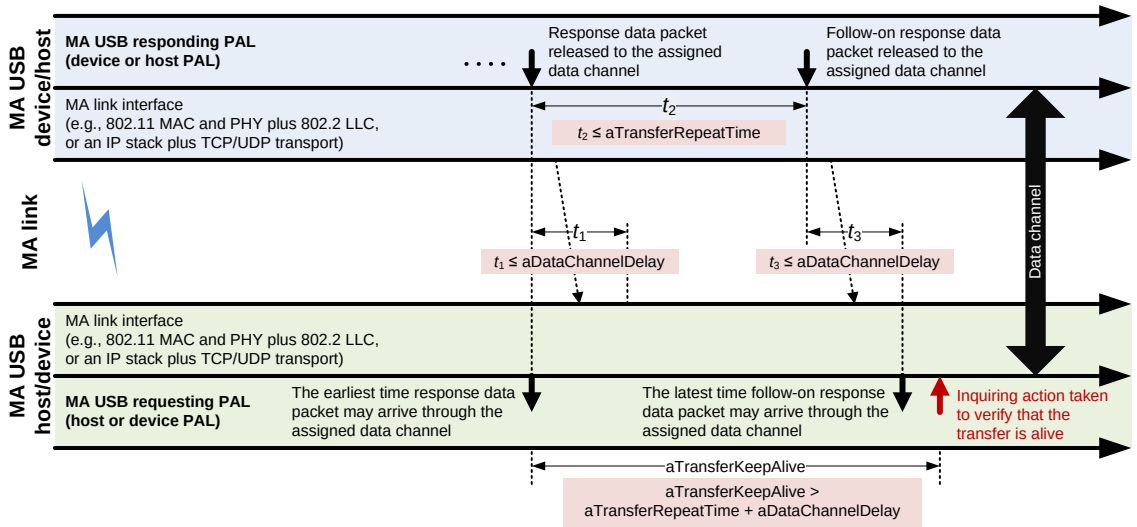


Figure 13—Default timings to keep an MA USB transfer alive

5.2.2 Ping protocol

The Ping protocol enables an MA USB host or MA USB device to verify and possibly re-establish connectivity with a target MA USB device or the MA USB host. The protocol involves exchange of two management packets: A Ping Request (PingReq) packet (Section 6.3.34) and a Ping Response (PingResp) packet (Section 6.3.35). Receiving a PingResp packet from a target MA USB device enables the address resolution function of an MA USB host PAL to associate the MA USB device address of the target MA USB device with a local and remote network address pair through which the target MA USB device can be reached. Similarly, receiving a PingResp packet from the MA USB host enables the address resolution function of an MA USB device PAL to identify the local and a remote network address through which the MA USB host can be reached.

NOTE — The Ping protocol can be used to re-discover a peer MA USB PAL over a different network interface without using a network-level discovery protocol. For example, an MA USB host PAL can transmit a PingReq packet over different network interfaces (e.g., over different 802.11 radios in 802.11 mode operation) to target the same MA USB device. Receiving a PingResp packet over any of the network interfaces enables the MA USB host PAL to associate the target MA USB device address with a local network interface and a remote network address through which the target MA USB device can be reached.

A PingReq packet transmitted by the MA USB host can target all MA USB devices in its Service Set by setting the Device Address field (Section 6.2.1.6) in the PingReq packet to the broadcast address of 0xFF. Any MA USB device that receives a PingReq packet with the Device Address field set to its device address or broadcast address and the SSID field matching the MA USB device Service Set shall respond with a PingResp packet with the Device Address field set to the MA USB address of that device. A PingReq packet transmitted by the MA USB device shall carry the device MA USB address in the Device Address field. An MA USB host receiving a PingReq packet with the SSID field matching its Service Set shall respond with a PingResp packet with the Device Address field set to the same value as the Device Address field in the PingReq packet. An MA USB device shall not transmit a PingReq packet with the Device Address field set to broadcast address.

NOTE — Broadcast management packets may be transmitted using network-level multicast or broadcast, which may make them less reliable than unicast packets.

The Ping protocol packet exchange follows the same timings as other management packet exchanges (Section 5.2.1.1); in particular, if a PingResp packet is not received within aManagementRequestTimeout after releasing the last PingReq packet retry to the management channel, the MA USB PAL transmitting the PingReq packet shall transition to the Session Down state (Section 8.1.1.1) and inform the lower layers of the session state transition.

A trigger for start of the Ping protocol shall result in a PingReq packet only if there is not another PingReq packet pending a response.

5.2.3 Data transfer

Compared to USB transfers, which are defined using rigid packet sizes at the bus level, MA USB transfers take a more flexible form: Each MA USB transfer carries a flexible number of bytes within the byte stream made available through a USB pipe interface, to or from the remote USB endpoint (on an MA USB device) that the pipe is associated with. Put differently, the bytes written to or read from a USB pipe are made available through one or more MA USB transfers, executed sequentially to preserve the byte stream order and integrity.

All MA USB transfers are initiated by the MA USB host. An MA USB IN transfer delivers USB payload from a target USB endpoint to the pipe interface associated with that endpoint, and an MA USB OUT transfer delivers USB payload from a pipe interface to the target USB endpoint that the pipe is associated with. An MA USB IN transfer is not complete until all requested USB payload has been

placed in the receive buffer designated by the MA USB host PAL. An MA USB OUT transfer is not complete until all transmitted USB payload is successfully delivered to the target USB endpoint. In particular, successful delivery of MA USB packets over the network is not sufficient to declare an MA USB transfer complete.

The starting point for MA USB transfers is a read or write request made through the USBDI interface at the MA USB host. MA USB host services each read or write request through one or more MA USB transfers. The number and size of each MA USB transfer is implementation-specific, and generally depends on the buffer space available to the MA USB host, the buffer space available to the target MA USB device, or both. Each MA USB transfer is identified by a *Transfer Request ID* or *Request ID* for short. All MA USB packets belonging to the same MA USB transfer carry the same Request ID. Concurrent MA USB transfers targeting different endpoints may use the same Request ID.

5.3 Transfer models

An MA USB transfer moves data between a local buffer residing in the MA USB host, which is fully allocated to the transfer before the transfer is initiated, and a remote buffer residing in a target MA USB device, which is managed (allocated, resized and released) by the MA USB device. Except for isochronous transfers that allow data loss, MA USB transfers employ a sliding-window protocol to ensure reliable delivery; the sliding-window protocol has been optimized based on a few assumptions,

- As a protocol running over modern data link or network layer technologies, MA USB expects reliable and in-order data delivery by the MA link itself *as part of the normal operation*, but as a transport layer protocol, it must be equipped to handle both unreliable and out-of-order data reception in uncommon scenarios such as link loss or switching from one MA link interface to another.
- MA USB can be expected to run over MA link technologies with large MTU values relative to the target MA USB device buffer size, and learning the buffer size available to a transfer can help the transmitter with packetizing decisions.
- To make better use of limited memory under concurrent transfers, devices should be able to adjust the buffer size available to each transfer while the transfer is in progress.

Figure 14 shows the taxonomy of MA USB transfers. IN transfers do not need flow control, as a receive buffer not smaller than the entire transfer size is assumed to be available on the host before the transfer is initiated. All non-isochronous IN transfers follow the same protocol defined in Section 5.4.

OUT transfers generally require flow control, as the receive buffer on the target MA USB device is not required to be as large as the transfer size. This specification defines two models for non-isochronous OUT transfers, with the primary difference between the models being in how they achieve flow control. The *protocol-managed* (p-managed) model uses protocol-level (MA USB) packets for flow control. Support of the p-managed transfer model is mandatory for all MA USB hosts and devices. The *link-managed* (l-managed) model offloads the flow control function to the underlying link protocol by carrying the transfer payload (targeting a single OUT endpoint) over a dedicated flow-controlled link-level connection. In this model, MA USB host does not make any assumptions about the memory available to the transfer on the target MA USB device. Support for the l-managed transfer model is optional for MA USB hosts and MA USB devices.

Isochronous transfers follow a different flow, and are described in Section 5.10.

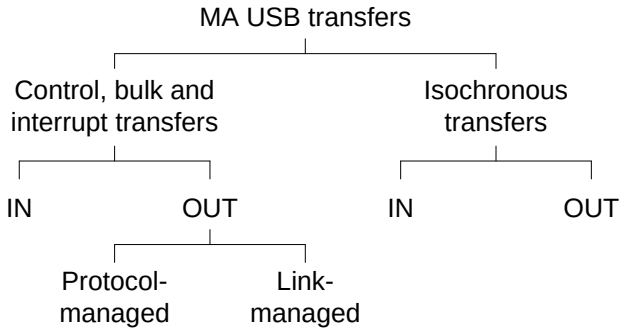


Figure 14—Taxonomy of MA USB transfers

5.4 IN transfers

An MA USB IN data flow is a unidirectional stream of bytes delivered from a target USB IN endpoint or a stream buffer within an Enhanced SuperSpeed bulk IN endpoint that supports the Enhanced SuperSpeed Stream Protocol [USB 3.1] to the MA USB host.

NOTE — Throughout this specification the term “endpoint flow” refers to the unidirectional byte stream targeting an endpoint, and the term “stream flow" refers to the unidirectional byte stream targeting a stream within an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol.

The MA USB IN data flow is illustrated in Figure 15. The MA USB host PAL initiates an MA USB IN transfer by transmitting a TransferReq packet (Section 6.5.2) to a target MA USB device. The MA USB host PAL assigns a Request ID to each transfer request and associated TransferReq packet. Request ID is reset to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and is incremented by 1 for each new transfer request, with wraparound to 0 after reaching the maximum value of aMaxRequestID.

Each TransferReq packet includes a Remaining Size field that carries the number of remaining bytes the MA USB host PAL expects to receive to complete the transfer.

Each TransferReq packet also includes a Sequence Number field that carries the sequence number value the MA USB host is expecting to receive next. The sequence number is set to 0 upon any configuration event that returns the state of the target endpoint or stream flow to the initial state.

NOTE — Configuration events that return the state of an endpoint or stream flow to the initial state are triggered in response to various events at the MA USB host or device, and are normally communicated through appropriate management packets. As a result, the time an endpoint or stream flow makes the transition to its initial state depends on whether the event is viewed from the MA USB host or from the target MA USB device perspective. For example, the MA USB host is assumed to have initialized a target endpoint flow when it transmits an EPHandleDeleteReq or USBDevDisconnectReq packet that targets the endpoint; the target MA USB device on the other hand, can be assumed to have initialized the endpoint flow when it transmits a corresponding response packet to the MA USB host with a status code that indicates the endpoint state was reset (e.g. status code of SUCCESS).

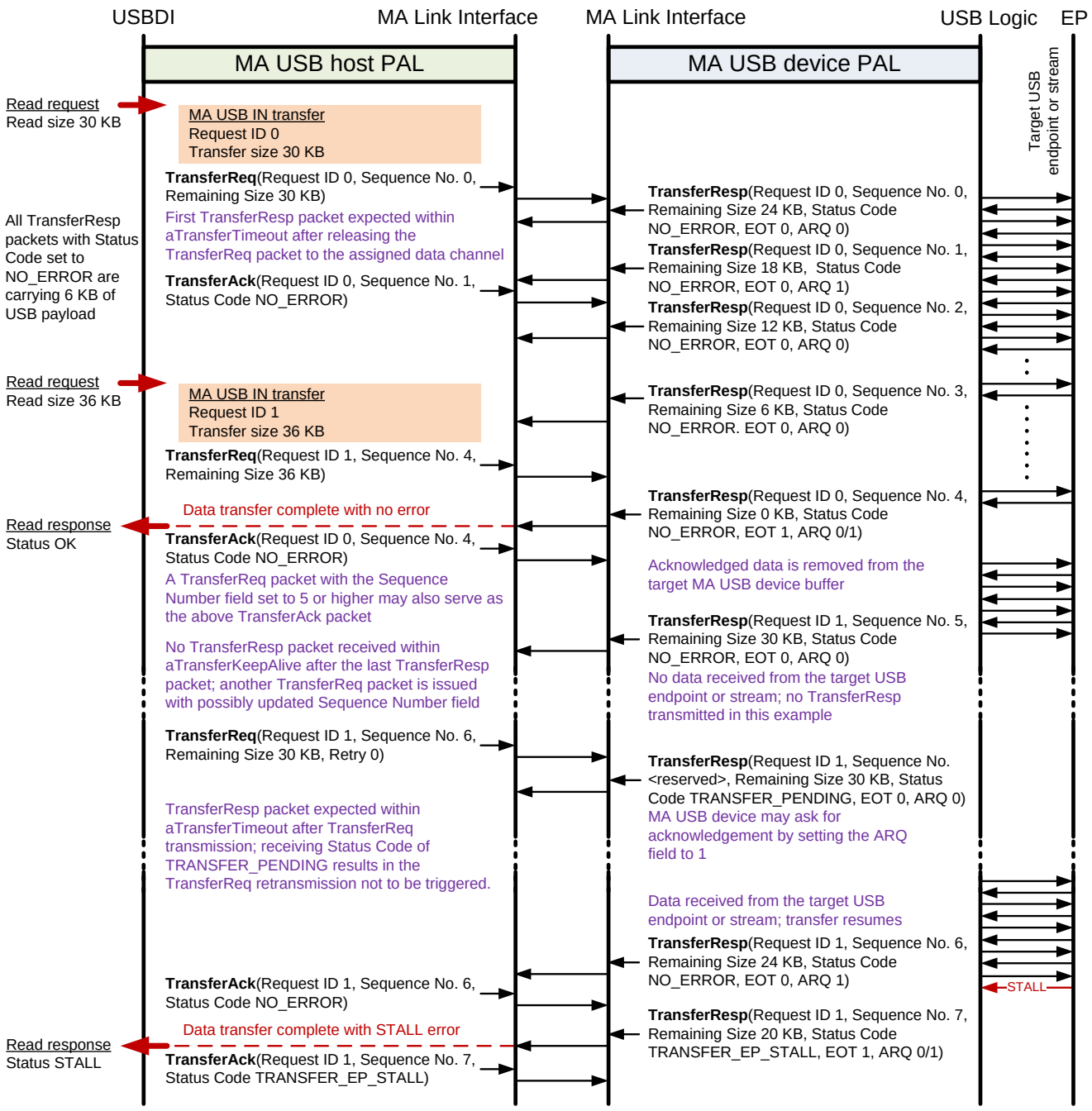


Figure 15—MA USB IN transfer

In response to a TransferReq packet, the target MA USB device shall transmit one or more TransferResp packets (Section 6.5.3) to the MA USB host to transfer the USB payload from the target USB endpoint or stream in the strict order it was received from the target endpoint or stream. Each TransferResp packet carries the same EP Handle, Stream ID, and Request ID as in the TransferReq packet that initiated the transfer. To enable the MA USB host to identify missing data packets, each TransferResp packet carries a Sequence Number field, which is set to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state. The Sequence Number value is incremented by 1 after each new (i.e., not retried) TransferResp packet, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber. The Sequence Number values keep increasing across successive transfers.

To avoid ambiguity in tracking the TransferResp packets pending completion, the MA USB device shall have no more than (aMaxSequenceNumber +1)/2 (half the size of the Sequence Number space) outstanding TransferResp packets.

In the absence of an active IN transfer, and if the target endpoint or stream has USB data to transmit, a TransferReq packet and the first corresponding TransferResp packet form an immediate exchange with timings and behavior defined in Section 5.2.1.2, except that a retried TransferReq packet may carry an updated Sequence Number value, in which case it is transmitted with the Retry field set to 0. While an MA USB IN transfer on a target endpoint or stream is in progress, the MA USB host may transmit additional TransferReq packets to target the same endpoint or steam, with the value of the Request ID field incremented by 1 for each new transfer request. The target MA USB device queues the received TransferReq packets in a request buffer in the increasing order of Request ID values for subsequent processing. Upon completion of an IN transfer, the target MA USB device shall immediately process the next TransferReq packet in its request buffer if it is not empty, resulting in an immediate exchange with timings and behavior defined in Section 5.2.1.2, except that a retried TransferReq packet may carry an updated Sequence Number value and is transmitted with the Retry bit set to 0.

NOTE — A retried TransferReq packet for an active transfer may not change the size of the transfer.

NOTE — The target MA USB device may transmit a null TransferResp packet (a TransferResp packet with no payload) with the Status Code field set to TRANSFER_PENDING in response to a TransferReq packet if the target endpoint or stream has no USB payload to transmit, i.e., the MA USB device does not have to wait for the MA USB host retrying a TransferReq packet before it can communicate the target endpoint or stream pending status.

The MA USB host shall not retry a TransferReq packet unless all previous transfer requests have been completed, and the MA USB host has not received a corresponding TransferResp packet for more than aTransferTimeout since the last time the MA USB host acknowledged the last TransferResp packet belonging to the immediately preceding MA USB transfer.

To avoid ambiguity in tracking the outstanding TransferReq packets, the MA USB host shall have no more than (aMaxRequestID +1)/2 (half the size of the Request ID space) outstanding TransferReq packets for each target IN endpoint or stream. Also, the total number of outstanding transfers across all IN and OUT endpoints and streams shall not exceed the number returned by the target MA USB device in the Number of Outstanding Requests field in the CapResp packet (Section 6.3.3).

NOTE — MA USB devices with multiple endpoints or streams are recommended to use a shared buffer to store the state information for all outstanding transfers, as the number of outstanding transfer on each target endpoint or stream is decided by the MA USB host.

A TransferReq packet queued by the target MA USB device is not acknowledged unless it is invalid (e.g., a gap is detected in the packet Request ID field), in which case the target MA USB device PAL shall discard the invalid TransferReq packet, and shall return a null TransferResp packet, i.e., a TransferResp packet with no payload, within aTransferResponseTime from the moment it received the invalid TransferReq packet, with the TransferResp packet Request ID field set to the next Request ID value the MA USB device is expecting, and the Status Code field set to MISSING_REQUEST_ID if the received TransferReq packet is valid but shows a gap in its Request ID field, or INVALID_REQUEST if the received TransferReq packet is invalid independent of the value of its Request ID field. In response to a TransferResp packet with the Status Code field set to MISSING_REQUEST_ID or INVALID_REQUEST, the MA USB host shall invalidate all outstanding TransferReq packets with request ID fields set to values larger than indicated in the TransferResp packet and start transmitting new TransferReq packets starting with the Request ID value in the TransferResp packet.

NOTE — These new TransferReq packets may carry updated information such as new values for the Remaining Size and Sequence Number fields, and are not considered retransmissions.

NOTE — The target MA USB device may use the Sequence Number field in an incoming TransferReq packet to update its internal state, even if there is a gap in the packet Request ID field.

NOTE — It is possible that a target MA USB device silently drops a valid incoming TransferReq packet if it cannot admit the packet into its request buffer; this scenario is no different from the packet being lost over the medium, and is recovered though a detected gap in subsequent TransferReq packets, or through an MA USB host timeout.

The target MA USB device fulfills an IN transfer request by transmitting one or more TransferResp packets. The interval between the release times of two successive TransferResp packets belonging to the same IN transfer to the assigned data channel shall not exceed aTransferRepeatTime, unless no data is available from the target USB endpoint or stream, in which case the target MA USB device may indicate a pending status and enter longer periods of inactivity as described later in this section.

NOTE — The above timing means that the interval between two successive TransferResp packets corresponding to the same IN transfer appearing over the medium is not to exceed aTransferResponseTime + aDataChannelDelay as long as there is USB payload to transmit.

For an active IN transfer involving multiple TransferResp packets, if the MA USB host PAL expects a TransferResp packet and does not receive the packet within aTransferKeepAlive from the moment it received the last TransferResp packet through the assigned data channel, it shall transmit a TransferReq packet to the target MA USB device with the Request ID field set to the value identifying the IN transfer, the Sequence Number field set to the value the MA USB host is expecting next, and the Remaining Size field set to the remaining number of bytes expected to complete the transfer. This TransferReq packet starts an immediate exchange with timings and behavior defined in Section 5.2.1.2, except that a retried TransferReq packet may carry an updated Sequence Number value and is transmitted with the Retry bit set to 0.

NOTE — A retried TransferReq packet for an active transfer may not change the size of the transfer.

An MA USB device that is experiencing delay in receiving data from a target USB endpoint or stream shall indicate a pending status and transition to a longer timeout by transmitting a null TransferResp packet, i.e., a TransferResp packet with no payload, with the Status Code field set to TRANSFER_PENDING and optionally the ARQ field set to 1; the Sequence Number and Remaining Size fields in the null TransferResp packet are set to aInvalidSequenceNumber and reserved, respectively. If the ARQ field in the null TransferResp packet is set to 1, the packet starts an immediate exchange with the MA USB host that requires the host to acknowledge the null TransferResp packet through a TransferAck packet with the same Request ID and Status Code field values as those in the null TransferResp packet; the timings and behavior for the immediate exchange are defined in Section 5.2.1.2. The first TransferResp packet with payload that follows a null TransferResp packet with Status Code field set to TRANSFER_PENDING shall have the ARQ field set to 1, resulting in an immediate exchange with the MA USB host, with timings and behavior defined in Section 5.2.1.2.

NOTE — The null TransferResp packet indicating a pending status can be transmitted as part of the flow of TransferResp packets transmitted to the MA USB host (unsolicited), or in response to a TransferReq packet (solicited).

NOTE — A TransferResp packet with payload (i.e., valid Sequence Number field value) and the Status Code field set to NO_ERROR can be acknowledged by a TransferAck packet with the same Request ID value, the same Sequence Number value, and the same Status Code value (NO_ERROR), or alternatively, by a TransferReq packet (including a new outstanding transfer request) with the same or larger Request ID value, larger Sequence Number value, and the same Status Code value (NO_ERROR). A TransferResp packet with the Status Code field set to any value other than NO_ERROR (e.g., a null TransferResp packet with the Status Code field set to TRANSFER_PENDING and the Sequence Number field set to aInvalidSequenceNumber) can be acknowledged only by a TransferAck packet with the same Request ID value, the same Sequence Number value (unless the value of the field in the TransferResp packet is set to aInvalidSequenceNumber, in which case the value of the Sequence Number field in the TransferAck packet is also set to aInvalidSequenceNumber), and the same Status Code value. Also see the rules for removing IN transfer data from the MA USB device buffer later in this section.

Once the MA USB host PAL receives a TransferResp packet with the Status Code field set to TRANSFER_PENDING, it shall reset the TransferReq retry logic, and set the transfer timeout period to $K \times$ aTransferKeepAlive, where $K \geq 0$ is a per-transfer variable with the default value of aDefaultKeepAliveDuration that the MA USB host establishes by responding to an EPSetKeepAliveReq packet (Section 6.3.52) transmitted by the target MA USB device. The MA USB host PAL shall maintain the extended timeout period as long as the target endpoint or stream is in pending state, and shall change it to aTransferKeepAlive after receiving the first TransferResp packet with the Status Code field set to a value other than TRANSFER_PENDING.

NOTE — The special value of $K = 0$ is reserved for infinite transfer timeout, meaning that the MA USB host PAL will not have to poll the endpoint once it receives a null TransferResp packet with the Status Code field set to TRANSFER_PENDING.

NOTE — The TransferReq packet that the MA USB host PAL transmits after not receiving a TransferResp packet for the target endpoint or stream in pending state for $K \times$ aTransferKeepAlive starts an immediate exchange with timings and behavior defined in Section 5.2.1.2. In particular, once the MA USB host transmits a TransferReq packet, the period between successive retries of the packet in case the MA USB host PAL does not receive a response packet stays at aTransferKeepAlive. Receiving a new TransferResp packet that repeats the TRANSFER_PENDING Status Code value resets the transfer timeout period to $K \times$ aTransferKeepAlive.

NOTE — The target MA USB device may request a new value for the multiplicative factor *K*, with the intention of establishing the factor for the target endpoint, specifically, for all IN transfers targeting the endpoint or streams within the endpoint; however, the MA USB transfer to which a new value of *K* applies is decided by the MA USB host PAL and can be an arbitrary transfer in the future, which makes *K* (and the transfer timeout period) transfer-dependent.

NOTE — Following communicating the pending status of the target endpoint or stream to the MA USB host, the MA USB device does not have to wait for the MA USB host to retry a TransferReq packet before it can transmit a TransferResp packet carrying payload.

NOTE — The pending status of one or more endpoints, does not prevent the MA USB host from communicating with other endpoints on the MA USB device. The target MA USB device shall mark the last TransferResp packet belonging to an IN transfer by setting the EoT field to 1, which results in an immediate exchange that requires the MA USB host to acknowledge the packet through a TransferAck packet, or optionally, through an outstanding TransferReq packet if the TransferResp packet has the Status Code field set to NO_ERROR. If acknowledging by a TransferAck packet, the TransferAck packet shall have the same values for the Request ID, Sequence Number and Status Code fields as the corresponding values in the TransferResp packet. If the total data delivered to the MA USB host is less than the amount indicated in the Remaining Size field of the TransferReq packet, then the TransferResp packet carrying EOT field set to 1 shall have the Status Code field set to TRANSFER_SHORT_TRANSFER.

NOTE — The target MA USB device may also set the ARQ field to 1 when it sets the EoT field to 1; a TransferResp packet with the EoT field or the ARQ field set to 1 results in an immediate exchange with the MA USB host with timings and behavior defined in Section 5.2.1.2, except that the target MA USB device is generally not required to retry the TransferResp packet if it does not receive an acknowledgement. An exception is the first TransferResp packet with payload transmitted after the target endpoint or stream has no longer a pending status, which has to be retried if needed.

A null TransferResp packet with the Status Code field set to TRANSFER_PENDING shall not have the EoT field set to 1.

If the target endpoint returns a STALL handshake during a transfer, the target MA USB device shall proceed to deliver the entire payload it has received from the target endpoint or stream to the MA USB host before concluding the transfer in error. Specifically, the target MA USB device shall set the Status Code field to NO_ERROR in all TransferResp packets it transmits as part of the transfer, except the last TransferResp packet, which shall have the Status Code field set to TRANSFER_EP_STALL, the EoT field set to 1, and optionally the ARQ field set to1. The last TransferResp packet initiates an immediate

exchange with the MA USB host, requiring the host to acknowledge the TransferResp packet through a TransferAck packet, with timings and behavior defined in Section 5.2.1.2. The TransferAck packet shall have the same values for the Request ID, Sequence Number and Status Code fields as the corresponding values in the TransferResp packet. The target MA USB device shall queue any new transfer request that targets a stalled endpoint or stream for possible later processing.

NOTE — The last TransferResp packet belonging to an IN transfer that experiences a STALL condition has a nonzero Remaining Size field, although the EoT field in the packet is set to 1.

The MA USB host PAL may stop tracking the state of an IN transfer and release the associated resources once it has acknowledged the last TransferResp packet belonging to the transfer, and the MA USB host PAL does not receive a TransferResp packet for aMaxTransferLifetime since it last released the acknowledgement packet to the last TransferResp packet to the assigned data channel.

NOTE — This condition is defined to ensure the target MA USB device has received the acknowledgement packet.

In response to a TransferResp packet with gap in the Sequence Number field, the MA USB host PAL shall release to the assigned data channel a TransferReq packet with the Request ID field identifying the transfer request the earliest missing Sequence Number value belongs to, the Sequence Number field set to the earliest missing Sequence Number value, the Remaining Size field set to the remaining size of the transfer identified by the Request ID field value, and the Status Code field set to NO_ERROR, within aTransferResponseTime from the moment the MA USB host PAL receives the TransferResp packet.

NOTE — The target MA USB device may transmit multiple TransferResp packets belonging to successive IN transfers, without waiting for the last TransferResp packet of each transfer to be acknowledged. A TransferReq packet with the Sequence Number field set to *N* also acknowledges TransferResp packets with Sequence Number values smaller than *N*, up to the first TransferResp packet with a nonzero Status Code value.

A target MA USB device shall not remove any transfer data associated with an MA USB IN transfer from its buffer unless it receives one of the following packets,

- A TransferAck packet acknowledging all TransferResp packets with Sequence Number values less than or equal to the value of the Sequence Number field in the packet in which case all acknowledged data is removed from the MA USB device buffer.
- A TransferReq packet (including new outstanding transfer requests) acknowledge all TransferResp packets with Sequence Number values less than the value of the Sequence Number field in the packet, in which case all acknowledged data is removed from the MA USB device buffer.
- A DevResetReq packet (Section 6.3.18) or DevDisconnectReq packet (Section 6.3.36), in which case all data for all endpoints and streams is removed from the MA USB device buffer.
- A ClearTransfersReq packet (Section 6.3.14), in which case all data related to corresponding endpoint(s) is removed from the MA USB device buffer.
- A USBDevDisconnectReq packet (Section 6.3.26) for the target USB device, in which case all data related to the target USB device is removed from the MA USB device buffer.
- A CancelTransferReq packet (Section 6.3.42), in which case all data related to the target request is removed from the MA USB device buffer.

A target MA USB device that receives a CancelTransferReq packet (Section 6.3.42) should remove from its buffer all the data corresponding to the transfer identified by the CancelTransferReq packet, except for the data that is already received from the target endpoint or stream (and hence allocated a sequence number) and has not been acknowledged by the host. The specific behavior of the MA USB device and the MA USB host depends on the state of the transfer at the time that the CancelTransferReq is processed by the MA USB device:

- If the transfer was cancelled before any data was moved from the USB device (i.e., the Cancellation Status field in CancelTransferResp packet set to 1): The MA USB devices shall respond to a TransferReq packet for the cancelled transfer with a TransferResp packet with EOT set to 1, Status Code field set to TRANSFER_CANCELLED, the Remaining Size field set to 0, and ARQ field set to 1. Additionally the MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.
- If the transfer was cancelled after some data was moved from the USB device (i.e., the Cancellation Status field in CancelTransferResp packet set to 2): The MA USB device shall transmit all the data received from the USB device to the MA USB host in TransferResp packets. The last TransferResp packet of the transfer shall carry EOT field set to 1 with the Status Code field set to TRANSFER_CANCELLED. The TransferResp packets other than the last in the transfer may carry Status Code field set to TRANSFER_CANCELLED or SUCCESS. Additionally the MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.
- If the transfer was completed (i.e., the Cancellation Status field in CancelTransferResp packet set to 3): The MA USB device shall transmit all the data received from the USB device to the MA USB host in TransferResp packets, with the Status Code field set to the appropriate values (the same values carried if the transfer was not cancelled). The last TransferResp packet of the transfer shall carry EOT field set to 1. The MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.
- If the transfer was not yet received (i.e., the Cancellation Status field in CancelTransferResp packet set to 4): The MA USB device is not required to keep cancel information for a transfer it has not yet received. If a TransferReq packet is received at the MA USB device, the MA USB device shall respond to it with no required knowledge of whether the transfer was previously cancelled. The MA USB host, following the receipt of the TransferResp packet, may either retransmit the CancelTransferReq or wait for the completion of the transfer.
- If the transfer with RequestID was serviced as part of ClearTransfersReq processing without any data being moved from the USB Device (i.e., the Cancellation Status field in CancelTransferResp packet set to 5): The MA USB device is not required to keep the cancel information for a transfer it already cleared, It shall generate CancelTransferResp without generating TransferResp. This status indicates that the MA-USB device doesn't expect TransferAck from MA USB host. The MA USB host, following receipt of the CancelTransferResp packet, shall not wait for TransferResp packets for this transfer.

A target MA USB device that receives a ClearTransfersReq packet (Section 6.3.14) should remove from its buffer all the data corresponding to the transfers identified by the ClearTransfersReq packet (i.e., all the data related to all the transfers with Request ID values preceding the value of Start Request ID field in the ClearTransfersReq packet), except for the data that is already received from the target endpoint (and hence allocated a sequence number) and has not been acknowledged by the host. The MA USB device shall deliver all the data received from the target endpoint to the MA USB host through TransferResp packets; if the transfer is cancelled before its completion (i.e., not all data related to the

transfer is received from the target endpoint), the last TransferResp packet carrying data related to the transfer shall carry EOT field set to 1 with the Status Code field set to TRANSFER_CANCELLED. The TransferResp packets related to transfers that are completed without cancellation shall not set the Status Code field to TRANSFER_CANCELLED.

In addition, the MA USB device shall discard any TransferReq packet received for a cancelled transfer (TransferReq packets carrying a Request ID value less than the value indicated in the Start Request ID field in the ClearTransfersReq packet). The MA USB device shall respond to a ClearTransfersReq packet with a ClearTransfersResp packet, only after all the TransferResp packets for the data already received from cancelled transfers are generated. The MA USB host shall reset the Sequence Number value to 0 before transmitting a TransferReq packet with Request ID value indicated in the Start Request ID field in the ClearTransfersReq packet.

5.4.1 Transfer description

The transfer description in this section does not cover all scenarios resulting from interaction of different events, and is provided to serve as an implementation guideline. The transfer description in Section 5.4 takes precedence over this description in case of a conflict.

The operation of the MA USB host PAL and a target MA USB device PAL is defined in terms of a set of state variables and processes, where each process is a basic unit of execution. Processes do not interrupt each other, meaning that a state variable is not modified during a process execution, unless modified by the process itself. State variables used to describe the host operation are marked with an 'R' (responder) subscript (e.g., $SeqNumber_R$), and state variables used to describe the device operation are marked with an 'O' (originator) subscript (e.g., $SeqNumber_O$). An index notation is used for variables that have a scope of a single MA USB transfer, e.g., $RemSize_R[r]$ denotes a state variable belonging to an MA USB transfer request with Request ID equal to *r*.

Figure 16 illustrates the data packets and header fields used to perform MA USB IN transfers.

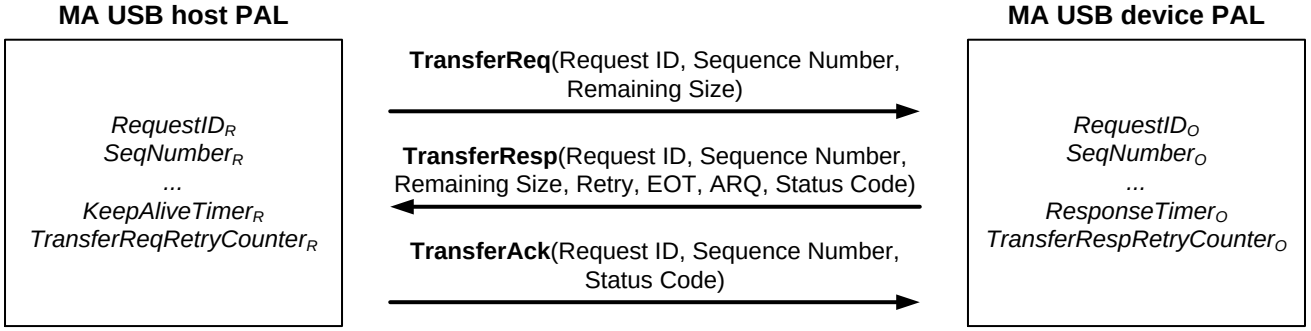


Figure 16—Data packets and header fields used for IN transfers

5.4.1.1 MA USB host PAL operation

The MA USB host operation is defined in terms of a series of state variables maintained in the context of a single target endpoint or stream,

- $RequestID_R$ (8 bits, unsigned): The Request ID value of the next original transfer request; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when a new transfer request is generated, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- $ActiveRequestID_R$ (8 bits, unsigned): The Request ID field of the active transfer request, i.e., the request expected to be served by the next original TransferResp packet with payload;; initialized

to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when the entire payload belonging to a transfer has been received (not necessarily acknowledged), with wraparound to 0 after reaching the maximum value of aMaxRequestID.

- *EarliestRequestID*$_R$ (8 bits, unsigned): The Request ID value of the earliest transfer request whose state needs to be tracked; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when it is determined that the acknowledgement for the entire payload belonging to the transfer has been received by the target MA USB device PAL, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *SeqNumber*$_R$ (24 bits, unsigned): Expected value of the Sequence Number field of the next original TransferResp packet to be received; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when an original TransferResp packet is received and admitted to the receive buffer, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- *KeepAliveTimer*$_R$ (signed): A decrementing counter to track the elapsed time between successive TransferResp packets belonging to the active transfer; set to aTransferKeepAlive each time the MA USB host moves to processing a new transfer, and reset to aTransferKeepAlive or a transfer-dependent multiple of aTransferKeepAlive each time a TransferResp packet is received, with the reset value depending on the transfer state and the Status Code value received in the TransferResp packet; decremented by aTransferTimerTick at every transfer timer tick event.
- *TransferReqRetryCounter*$_R$ (unsigned): A decrementing counter to track the number of retries for a TransferReq packet that requires an immediate exchange (Section 5.2.1.2); set to *TransferReqRetries*$_R$ when a new round of retries is to be attempted, and decremented by 1 after each retry.
- *TransferReqRetries*$_R$ (unsigned): The reload value of the *TransferReqRetryCounter*$_R$ counter, set to aControlTransferRetries, aBulkTransferRetries or aInterruptTransferRetries for control, bulk and interrupt transfers respectively.

The remaining state variables have a finer scope of a single MA USB transfer targeting the endpoint or stream; specifically, for a given MA USB IN transfer with Request ID equal to *r*,

- *RemSize*$_R$[*r*] (32 bits, unsigned): The number of bytes expected to be received; set to the transfer size in bytes at transfer initialization, and decremented throughout the transfer.
- *TransferError*$_R$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon detecting an error in the transfer.
- *TransferComplete*$_R$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE when it is no longer necessary to track the state of the transfer.
- *EndOfTransferDetected*$_R$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon receiving the last TransferResp packet belonging to the transfer.
- *TransferAcknowledged*$_R$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon releasing a TransferReq or TransferAck packet to the assigned data channel that acknowledges the last TransferResp packet belonging to the transfer.
- *LastTransferSN*$_R$[*r*] (24 bits, unsigned): The value of the Sequence Number field in the TransferResp packet that carries the last portion of the payload belonging to the transfer.

- *TransferCompletionTimer*$_R$[*r*] (signed): A decrementing counter to verify the transfer acknowledgement was received; set to aMaxTransferLifetime after the last TransferResp packet belonging to the transfer was acknowledged, and decremented by aTransferTimerTick at every transfer timer tick event.

- $K_R[r]$ (unsigned): A transfer-dependent multiplicative factor used to reload *KeepAliveTimer*$_R$ when a TransferResp packet with the Status Code field set to TRANSFER_PENDING is received.

NOTE — The target MA USB device may request a new value for the multiplicative factor $K_R[r]$, with the intention of establishing the factor for the target endpoint, specifically for all IN transfers targeting the endpoint or streams within the endpoint; however, the MA USB transfer to which a new value of $K_R[r]$ applies is decided by the MA USB host and can be an arbitrary transfer in the future, which makes $K_R[r]$ transfer-dependent.

The MA USB host PAL operation is defined in terms of the processes described below.

Initialization process

Invoked on any configuration event intended to return the state of the endpoint or stream flow to the initial state.

I. Set *RequestID*$_R$ = 0.
II. Set *SeqNumber*$_R$ = 0.

NOTE — The *SeqNumber*$_R$ variable keeps increasing across successive transfers.

Transfer initialization process (starting a new transfer)

Invoked every time a new transfer request is initiated.

I. Select r = *RequestID*$_R$ as the Request ID assigned to the transfer.
II. Initialize the following variables to manage the transfer,
    (a) Set *TransferError*$_R$[*r*] = *FALSE*.
    (b) Set *TransferComplete*$_R$[*r*] = *FALSE*.
    (c) Set *EndOfTransferDetected*$_R$[*r*] = *FALSE*.
    (d) Set *TransferAcknowledged*$_R$[*r*] = FALSE.
    (e) Set *RemSize*$_R$[*r*] = Transfer size in bytes, as indicated by the application.
III. Submit a TransferReq packet to the TransferReq transmission process with the Request ID field set to *r*, the Sequence Number field set to *SeqNumber*$_R$, and the Remaining Size field set to *RemSize*$_R$[*r*].
IV. If there is no active request,
    (a) Set *ActiveRequestID*$_R$ = *RequestID*$_R$.
    (b) Set *EarliestRequestID*$_R$ = *RequestID*$_R$.
    (c) Set *KeepAliveTimer*$_R$ = aTransferTimeout.
    (d) Set *TransferReqRetryCounter*$_R$ = *TransferReqRetries*$_R$.
V. Set *RequestID*$_R$ = *RequestID*$_R$ + 1.

NOTE — The difference between *RequestID*$_R$ and *EarliestRequestID*$_R$ does not exceed (aMaxRequestID + 1)/2 at any point in time; the difference may be further limited by the total number of outstanding transfers that the target MA USB device supports across all its endpoints and streams.

NOTE — The Status Code field in all outgoing TransferReq packets is set to NO_ERROR.

NOTE — Handling possible local transmission failures is implementation-dependent.

TransferReq transmission process

Invoked to release a TransferReq packet to the assigned data channel.

I. Let *SN* denote the value of the Sequence Number field in the TransferReq packet.
II. Release the TransferReq packet to the assigned data channel.
III. If there are open transfer requests,
  (a) For each open transfer request *u* in the interval $EarliestRequestID_R \leq u < ActiveRequestID_R$, and in increasing order of *u*,
    (1) If $EndOfTransferDetected_R[u]$ = TRUE and $TransferError_R[u]$ = FALSE and $LastTransferSN_R[u] < SN$,
      (i) Set $TransferAcknowledged_R[r]$ = TRUE.
      (ii) Set $TransferCompletionTimer_R[u]$ = aMaxTransferLifetime.
    (2) Otherwise, break the loop and exit the process.

TransferResp reception process

Invoked upon reception of a TransferResp packet.

I. Let *r*, *SN*, *Retry*, *EndOfTransfer*, *AckRequest* and *Status* respectively denote the values of the Request ID, Sequence Number, Retry, EoT, ARQ and Status Code fields in the received TransferResp packet. Let *PayloadSize* denote the size of the payload in the received TransferResp packet.
II. If $r < EarliestRequestID_R$ or $r \geq RequestID_R$ drop the packet.
III. Otherwise, if $r > ActiveRequestID_R$,
  (a) If *Status* = INVALID_REQUEST or *Status* = MISSING_REQUEST_ID,
    (1) Invalidate all outstanding transfer requests with Request ID values *r* to $RequestID_R - 1$.
    (2) Set $RequestID_R = r$.
  (b) Otherwise, if $SN = SeqNumber_R$,
    (1) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to MISSING_REQUEST_ID.
  (c) Otherwise, run Step VI below (which effectively generates a response based on the Sequence Number value in the TransferResp packet).
IV. Otherwise, if $TransferComplete_R[r] = TRUE$ or $TransferError_R[r] = TRUE$ drop the packet.
V. Otherwise, if *Status* = TRANSFER_PENDING,
  (a) Set $KeepAliveTimer_R = K_R[r] \times$ aTransferKeepAlive.
  (b) Set $TransferReqRetryCounter_R[r] = TransferReqRetries_R$.
  (c) If *AckRequest* = 1, submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r* and the Status Code field set to TRANSFR_PENDING.

NOTE — The value of the Sequence Number field in the above TransferAck packet is set to aInvalidSequenceNumber.

NOTE — The $KeepAliveTimer_R$ variable is set to 0 when $K_R[r] = 0$, which stops the MA USB host PAL from polling the pending endpoint.

VI. Otherwise,
  (a) Set $KeepAliveTimer_R$ = aTransferKeepAlive.
  (b) Set $TransferReqRetryCounter_R[r] = TransferReqRetries_R$
  (c) If $SN = SeqNumber_R$,
    (1) If $RemSize_R[r] \geq PayloadSize$,
      (i) Admit the received payload into the host buffer.

(ii) Set $SeqNumber_R = SeqNumber_R + 1$.
(iii) *Set* $RemSize_R[r] = RemSize_R[r] – PayloadSize$.
(iv) If *EndOfTransfer* = 1,
(a) Set $EndOfTransferDetected_R[r]$ = TRUE.
(b) Set $LastTransferSN_R[r]$ = SN.
(c) Set $ActiveRequestID_R = ActiveRequestID_R + 1$.
(d) If *Status* = NO_ERROR,
(1) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to *Status* (NO_ERROR), or alternatively, initiate a new transfer, which will result in transmitting a TransferReq packet serving as acknowledgement.
(2) If $RemSize_R[r] > 0$, set $TransferError_R[r]$ = TRUE.

NOTE — A short read is reported to the application with a suitable error code, but does not result in transmitting an error code to the target MA USB device.

(e) Otherwise,
(1) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to *Status*, and set $TransferError_R[r]$ = TRUE.
(f) Return the entire payload received for the transfer to the application, together with a status code that matches the *Status* and $TransferError_R[r]$.
(v) Otherwise, if *AckRequest* = 1,
(a) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to *Status*, or alternatively, and only if *Status* = NO_ERROR, initiate a new transfer, which will result in transmitting a TransferReq packet serving as acknowledgement.
(2) Otherwise,
(i) Drop the TransferResp packet, or optionally admit up to $RemSize_R[r]$ bytes of the receive payload into the host buffer.
(ii) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to *TRANSFER_SIZE_ERROR*.
(iii) Set $TransferError_R[r]$ = *TRUE*.

NOTE — The target MA USB device PAL response to a received TRANSFER_SIZE_ERROR is implementation-dependent; the MA USB host normally takes corrective actions such as clearing all outstanding transfers on the endpoint or stream in this case.

(d) Otherwise,
(1) Drop the TransferResp packet.
(2) Submit a TransferReq packet to the TransferReq transmission process with the Request ID field set to *r*, the Sequence Number field set to $SeqNumber_R$, and the Remaining Size field set to $RemSize_R[r]$.

TransferAck transmission process

Invoked to release a TransferAck packet to the assigned data channel.

I. Let *SN* and *Status* respectively denote the value of the Sequence Number and Status Code fields in the TransferAck packet.
I. Release the TransferAck packet to the assigned data channel.
II. If $Status \neq$ TRANSFER_PENDING,
  (a) For each open transfer request u in the interval $EarliestRequestID_R \leq u < ActiveRequestID_R$, and in increasing order of u,
    (1) If $EndOfTransferDetected_R[u]$ = TRUE,
      (i) If $TransferError_R[u]$ = FALSE and $LastTransferSN_R[u] \leq SN$, or $TransferError_R[u]$ = TRUE and $u = r$,
        (a) Set $TransferAcknowledged_R[r]$ = TRUE.
        (b) Set $TransferCompletionTimer_R[u]$ = aMaxTransferLifetime.
      (ii) Otherwise, break the loop,
    (2) Otherwise, break the loop.

Timer process

Invoked at every transfer timer tick event, as long as there are open transfer requests.

I. For each open transfer request u in the interval $EarliestRequestID_R \leq u < ActiveRequestID_R$, and in increasing order of u,
  (a) If $TransferAcknowledged_R[u]$ = TRUE,
    (1) Set $TransferCompletionTimer_R[u] = TransferCompletionTimer_R[u]$ – aTransferTimerTick.
    (2) If $TransferCompletionTimer_R[u] \leq 0$, set $TransferComplete_R[u]$ = TRUE.
    (3) Otherwise, set $EarliestRequestID_R = u$ and break the loop (go to Step II).
  (b) Otherwise, set $EarliestRequestID_R = u$ and break the loop (go to Step II).
II. If $KeepAliveTimer_R > 0$,
  (a) Set $KeepAliveTimer_R = KeepAliveTimer_R$ – aTransferTimerTick.
  (b) If $KeepAliveTimer_R \leq 0$,
    (1) If $TransferReqRetryCounter_R > 0$,
      (i) Submit a TransferReq packet to the appropriate transmission process with the Request ID field set to *r* and the Sequence Number field set to $SeqNumber_R$.
      (ii)
      (iii) Set $KeepAliveTimer_R$ = aTransferKeepAlive. Set $TransferReqRetryCounter_R = TransferReqRetryCounter_R - 1$.
    (2) Otherwise,
      (i) Start the Ping protocol (Section 5.2.2).
      (ii) If the Ping protocol is successful, optionally,
        (a) Set $KeepAliveTimer_R$ = aTransferKeepAlive.
        (b) Set $TransferReqRetryCounter_R = TransferReqRetries_R$.
        (c) Quit the Timer process and wait for the next transfer timer tick event.
      (iii) If the Ping protocol fails, or if the optional path in the previous step is not followed,
        (a) Set $TransferError_R[ActiveRequestID_R] = TRUE$.

(b) Indicate a transfer timeout error to the application, and wait for a corrective action.

NOTE — The Status Code field in all outgoing TransferReq packets is set to NO_ERROR; also the Retry field in all these packets is set to 0, as they may include updated Sequence Number values.

NOTE — The MA USB host moves the session state to Session Down state (Section 8.1.1.1) after Ping protocol failure, which may result in a configuration event that returns the state of all endpoint and stream flows on the target MA USB device to the initial state. Also, regardless of the Ping protocol outcome, the MA USB host cannot proceed to the next IN transfer request if the current transfer fails as a result of timeout, unless a corrective action such as clearing all outstanding transfer requests and restarting the endpoint takes place. The above transfer-oriented description does not capture broader configuration events applied to the target endpoint or stream, or to the MA USB device.

5.4.1.2 MA USB device PAL operation

The MA USB device operation is defined in terms of a series of state variables maintained in the context of a single target endpoint or stream,

- *RequestID*$_O$ (8 bits, unsigned): Expected Request ID value of the next original transfer request; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1when a new transfer request is accepted, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *ActiveRequestID*$_O$ (8 bits, unsigned): The Request ID value of the active transfer request, i.e., the request served by the next original TransferResp packet with payload; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when the entire payload belonging to a transfer has been transmitted (not necessarily acknowledged), with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *EarliestRequestID*$_O$ (8 bits, unsigned): The Request ID value of the earliest transfer request whose state needs to be tracked; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when the entire payload belonging to the transfer has been acknowledged, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *SeqNumber*$_O$ (24 bits, unsigned): The value to be placed into the Sequence Number field of the next original TransferResp packet that the device transmits; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 after each original TransferResp packet transmission, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- *EarliestUnacknowledged*$_O$ (24 bits, unsigned): The value of the Sequence Number field of the earliest transmitted but unacknowledged TransferResp packet; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented throughout the transfer as described below, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- *KeepAliveTimer*$_O$ (signed): A decrementing counter to track the elapsed time between successive TransferResp packets belonging to the active transfer; set to aTransferRepeatTime every time a TransferResp packet belonging to the active transfer is released to the assigned data channel, as well as every time a new transfer becomes active and decremented by aTransferTimerTick at every transfer timer tick event.

- *ResponseTimer*$_O$ (signed): A decrementing counter to track the response time for a TransferResp packet that requires an immediate exchange (Section 5.2.1.2); set to aTransferTimeout every time a TransferResp packet that requires an immediate exchange is released to the assigned data channel, and decremented by aTransferTimerTick at every transfer timer tick event.
- *TransferRespRetryCounter*$_O$ (unsigned): A decrementing counter to track the retries of a TransferResp packet that requires an immediate exchange (Section 5.2.1.2); set to *TransferRespRetries*$_O$ when a new round of retries is to be attempted, and decremented by 1 after each retry.
- *TransferRespRetries*$_O$ (unsigned): The reload value of the *TransferRespRetryCounter*$_O$ counter, set to aControlTransferRetries, aBulkTransferRetries or aInterruptTransferRetries for control, bulk and interrupt transfers, respectively.
- *Delayed*$_O$ (boolean): Indicates if the rate of TransferResp packet transmission for the active transfer has fallen below a minimum rate; initialized to FALSE at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and also when a new transfer request becomes active, and set to TRUE if the interval between two successive TransferResp packets belonging to the active transfer exceeds aTransferRepeatTime.

The remaining state variables have a finer scope of a single MA USB transfer targeting the endpoint or stream; specifically, for a given MA USB IN transfer with Request ID equal to *r*,

- *RemSize*$_O$[*r*] (32 bits, unsigned): The number of bytes that the MA USB host can accept for the transfer; set to the transfer size in bytes at transfer initialization, and decremented throughout the transfer.
- *TransferError*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon detecting an error in the transfer.
- *TransferComplete*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE when it is no longer necessary to track the state of the transfer.
- *EndOfTransferDetected*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon releasing the last TransferResp packet belonging to the transfer to the assigned data channel.
- *LastTransferSN*$_O$[*r*] (24 bits, unsigned): The value of the Sequence Number field in the TransferResp packet that carries the last portion of the payload belonging to the transfer.

The MA USB device PAL operation is defined in terms of the processes described below.

Initialization process

Invoked on any configuration event intended to return the state of the endpoint or stream flow to the initial state.

I. Set *RequestID*$_O$ = 0.
II. Set *ActiveRequestID*$_O$ = 0.
III. Set *EarliestRequestID*$_O$ = 0.
IV. Set *SeqNumber*$_O$ = 0.
V. Set *EarliestUnacknowledged*$_O$ = 0.
VI. Set *Delayed*$_O$ = FALSE.

NOTE — The *SeqNumber*$_O$ and *EarliestUnacknowledged*$_O$ variables keep increasing across successive transfers.

TransferReq reception process

Invoked upon reception of a TransferReq packet.

I. Let *r*, *SN* and *RemSize* respectively denote the value of the Request ID, Sequence Number and Remaining Size fields in the received TransferReq packet.

II. If $EarliestRequestID_O - [(aMaxRequestID + 1)/2] \leq r < EarliestRequestID_O$, or $EarliestUnacknowledged_O - [(aMaxSequenceNumber + 1)/2] \leq SN < EarliestUnacknowledged_O$, or $SeqNumber_O < SN < SeqNumber_O + [(aMaxSequenceNumber + 1)/2]$,
   (a) Drop the TransferReq packet.
   (b) Submit a null TransferResp packet with no payload to the TransferResp transmission process with the Request ID field set to $RequestID_O$, the Retry, EoT and ARQ fields set to 0, and the Status Code field set to INVALID_REQUEST.

NOTE — The value of the Sequence Number field in a null TransferResp packet is set to aInvalidSequenceNumber.

III. Otherwise, if $r > RequestID_O$,
   (a) Drop the TransferReq packet.
   (b) Submit a null TransferResp packet with no payload to the TransferResp transmission process with the Request ID field set to $RequestID_O$, the Retry, EoT and ARQ fields set to 0, and the Status Code field set to MISSING_REQUEST_ID.

NOTE — The value of the Sequence Number field in a null TransferResp packet is set to aInvalidSequenceNumber.

IV. Otherwise,
   (a) Set $EarliestUnacknowledged_O = SN$.
   (b) For each open transfer request *u* in the interval $EarliestRequestID_O \leq u < ActiveRequestID_O$, and in increasing order of *u*,
      (1) If $EndOfTransferDetected_O[u]$ = TRUE and $TransferError_O[u]$ = FALSE,
         (i) If $LastTransferSN_O[u] < SN$, set $TransferComplete_O[u]$ = TRUE.
         (ii) Otherwise, set $EarliestRequestID_O = u$, and break the loop.
      (2) Otherwise, set $EarliestRequestID_O = u$, and break the loop.
   (c) If $r = ActiveRequestID_O$,
      (1) If $Delayed_O$ = TRUE, submit a null TransferResp packet with no payload to the TransferResp transmission process with the Request ID field set to *r*, the Remaining Size field reserved, the Retry field set to 0, the EoT field set to 0, the ARQ field set to 0 or optionally set to 1, and the Status Code field set to TRANSFER_PENDING.

NOTE — The value of the Sequence Number field in a null TransferResp packet with the Status Code field set to TRANSFER_PENDING is set to aInvalidSequenceNumber.

   (a) If $r = RequestID_O$,
      (1) Create a new transfer with Request ID value *r*.
      (2) Set $TransferError_O[r]$ = FALSE.
      (2) Set $TransferComplete_O[r]$ = FALSE.
      (3) Set $EndOfTransferDetected_O[r]$ = FALSE.
      (4) Set $RemSize_O[r] = RemSize$.
      (5) Set $LastTransferSN_O[r] = 0$.
      (6) Set $RequestID_O = RequestID_O + 1$.

TransferResp transmission process

Invoked to release a TransferResp packet to the assigned data channel, as long as $SeqNumber_O <$ $EarliestUnacknowledged_O + [(aMaxSequenceNumber + 1)/2]$.

I. Let *EndOfTransfer* and *AckRequest* respectively denote the values of the EoT and ARQ fields in the TransferResp packet.
II. Release the TransferResp packet to the assigned data channel.
III. If *EndOfTransfer* = 1 or *AckRequest* = 1, and $ResponseTimer_O \leq 0$,
   (a) Set $ResponseTimer_O$ = aTransferTimeout.
   (b) Set $TransferRespRetryCounter_O = TransferRespRetries_O$.

TransferResp generation process

Invoked to generate TransferResp packets as long as there is an active transfer request.

I. While $ActiveRequestID_O < RequestID_O$,
   (a) Set $r = ActiveRequestID_O$.
   (b) Let *PayloadSize* (implementation-dependent) denote the payload size for the next TransferResp packet to be generated, with $0 < PayloadSize \leq RemSize_O[r]$.
   (c) Define boolean variables *EndOfTransfer* and *AckRequest*, initialized to 0.
   (d) Define a variable *Status*, initialized to NO_ERROR.
   (e) If $RemSize_O[r] = PayloadSize$, or if $RemSize_O[r] > PayloadSize$ and no more data is going to be available as a result of an error,
      (1) Set *EndOfTransfer* = 1, and optionally set *AckRequest* = 1.
      (2) If $TransferError_O[r] = 1$, set *Status* to an appropriate error code.
   (f) Optionally, set *AckRequest* = 1.
   (g) Set $RemSize_O[r] = RemSize_O[r] - PayloadSize$.
   (h) Submit a TransferResp packet to the TransferResp transmission process, with the Request ID field set to *r*, the Sequence Number field set to $SeqNumber_O$, the Remaining Size field set to $RemSize_O[r]$, the Retry field set to 0, the EoT field set to *EndOfTransfer*, the ARQ field set to *AckRequest*, and the Status Code field set to *Status*.
   (i) Set $SeqNumber_O = SeqNumber_O + 1$.
   (j) If *EndOfTransfer* = 1,
      (1) If $TransferError_O[r] = 0$, set $ActiveRequestID_O = ActiveRequestID_O + 1$.
      (2) Otherwise, stop the process and wait for corrective action.

NOTE — Example of a corrective action is the MA USB host clearing all transfers and restarting the endpoint.

NOTE — The payload of a transmitted TransferResp packet is not released from the MA USB device memory until it is acknowledged by the MA USB host.

TransferAck reception process

Invoked upon reception of a TransferAck packet.

I. Let *r*, *SN* and *Status* respectively denote the value of the Request ID, Sequence Number and Status Code fields in the received TransferAck packet.
II. If $r > RequestID_O$ or $EarliestRequestID_O - [(aMaxRequestID + 1)/2] \leq r <$ $EarliestRequestID_O$ drop the TransferAck packet.
III. If *Status* = TRANSFER_PENDING,
   (a) If $r \neq ActiveRequestID_O$, drop the TransferAck packet.
   (b) Otherwise, possibly start power saving measures, understanding that the MA USB host will inquire about the status of the transfer based on an extended timeout.

NOTE — See the definition of $K_R[r]$ and the extended timeout in the MA USB host PAL operation (Section 5.4.1.1).

NOTE — The value of the Sequence Number field in a TransferAck packet with the Status Code field set to TRANSFER_PENDING is set to aInvalidSequenceNumber.

IV. Otherwise, if $EarliestUnacknowledged_O - [(aMaxSequenceNumber + 1)/2] \leq SN < EarliestUnacknowledged_O$, or $SeqNumber_O < SN < SeqNumber_O + [(aMaxSequenceNumber + 1)/2]$, drop the TransferAck packet.

V. Otherwise,

VI. if $EarliestUnacknowledged_O \leq SN < SeqNumber_O$,

(a) Set $EarliestUnacknowledged_O = SN + 1$.

(b) For each open transfer request u in the interval $EarliestRequestID_O \leq u < ActiveRequestID_O$, and in increasing order of u,

(1) If $EndOfTransferDetected_O[u]$ = TRUE,

(i) If $TransferError_O[u]$ = FALSE and $LastTransferSN_O[u] < SN$, or $TransferError_O[u]$ = TRUE and $u = r$,

(a) Set $TransferComplete_O[u]$ = TRUE.

(ii) Otherwise, set $EarliestRequestID_O = u$ and break the loop.

(2) Otherwise, set $EarliestRequestID_O = u$ and break the loop.

Timer process

Invoked at every transfer timer tick event, as long as there is an active transfer request.

I. If $KeepAliveTimer_O > 0$,

(a) Set $KeepAliveTimer_O = KeepAliveTimer_O$ – aTransferTimerTick.

(b) If $KeepAliveTimer_O \leq 0$,

(1) Set $Delayed_O$ = TRUE.

II. Optionally, if $ResponseTimer_O > 0$,

(a) Set $ResponseTimer_O = ResponseTimer_O$ – aTransferTimerTick.

(b) If $ResponseTimer_O \leq 0$,

(1) If $TransferRespRetryCounter_O > 0$,

(i) Submit the earliest TransferResp packet that requires acknowledgement to the TransferResp transmission process, with all packet fields the same, except the Retry field, which is set to 1.

(ii) Set $ResponseTimer_O$ = aTransferTimeout.

(iii) Set $TransferRespRetryCounter_O = TransferRespRetryCounter_O - 1$.

(2) Otherwise, wait indefinitely for corrective actions, or optionally,

(i) Start the Ping protocol (Section 5.2.2).

(ii) If the Ping protocol is successful, optionally,

(a) Set $ResponseTimer_O$ = aTransferTimeout.

(b) Set $TransferRespRetryCounter_O = TransferRespRetries_O$.

(c) Quit the Timer process and wait for the next transfer timer tick event.

(iii) If the Ping protocol fails, or if the optional path in the previous step is not followed,

(a) Set $TransferError_O[ActiveRequestID_O]$ = TRUE.

(b) Indicate a transfer timeout error to the application, and wait for a corrective action.

5.5 Protocol-managed OUT transfers

An MA USB OUT data flow is a unidirectional stream of bytes delivered to a target USB OUT endpoint or a stream buffer within an Enhanced SuperSpeed bulk OUT endpoint that supports the Enhanced SuperSpeed Stream Protocol [USB 3.1] from the MA USB host.

NOTE — Throughout this specification the term “endpoint flow” refers to the unidirectional byte stream targeting an endpoint, and the term “stream flow" refers to the unidirectional byte stream targeting a stream within an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol.

The MA USB host data flow is illustrated in Figure 17. The MA USB host PAL initiates an MA USB OUT transfer by transmitting a TransferReq packet (Section 6.5.2) to a target MA USB device, which also carries some payload belonging to the transfer; remaining payload is transferred through additional TransferReq packets. All TransferReq packets belonging to the transfer, except possibly the last TransferReq packet, shall include a multiple of *maximum packet size* of data supported by the target endpoint, where the *maximum packet size* is defined by the wMaxPacketSize field of the endpoint descriptor. The MA USB host assigns a request ID to each transfer request and all associated TransferReq packets. Request ID is reset to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and is incremented by 1 for each new transfer request, with wraparound to 0 after reaching the maximum value of aMaxRequestID. To avoid ambiguity in tracking the outstanding TransferReq packets, the MA USB host shall have no more than (aMaxRequestID +1)/2 (half the size of the Request ID space) outstanding TransferReq packets for each target OUT endpoint or stream. Also, the total number of outstanding transfers across all IN and OUT endpoints and streams shall not exceed the number returned by the target MA USB device in the Number of Outstanding Requests field in the CapResp packet (Section 6.3.3).

NOTE — MA USB devices with multiple endpoints or streams are recommended to use a shared buffer to store the state information for all outstanding transfers, as the number of outstanding transfer on each target endpoint or stream is decided by the MA USB host.

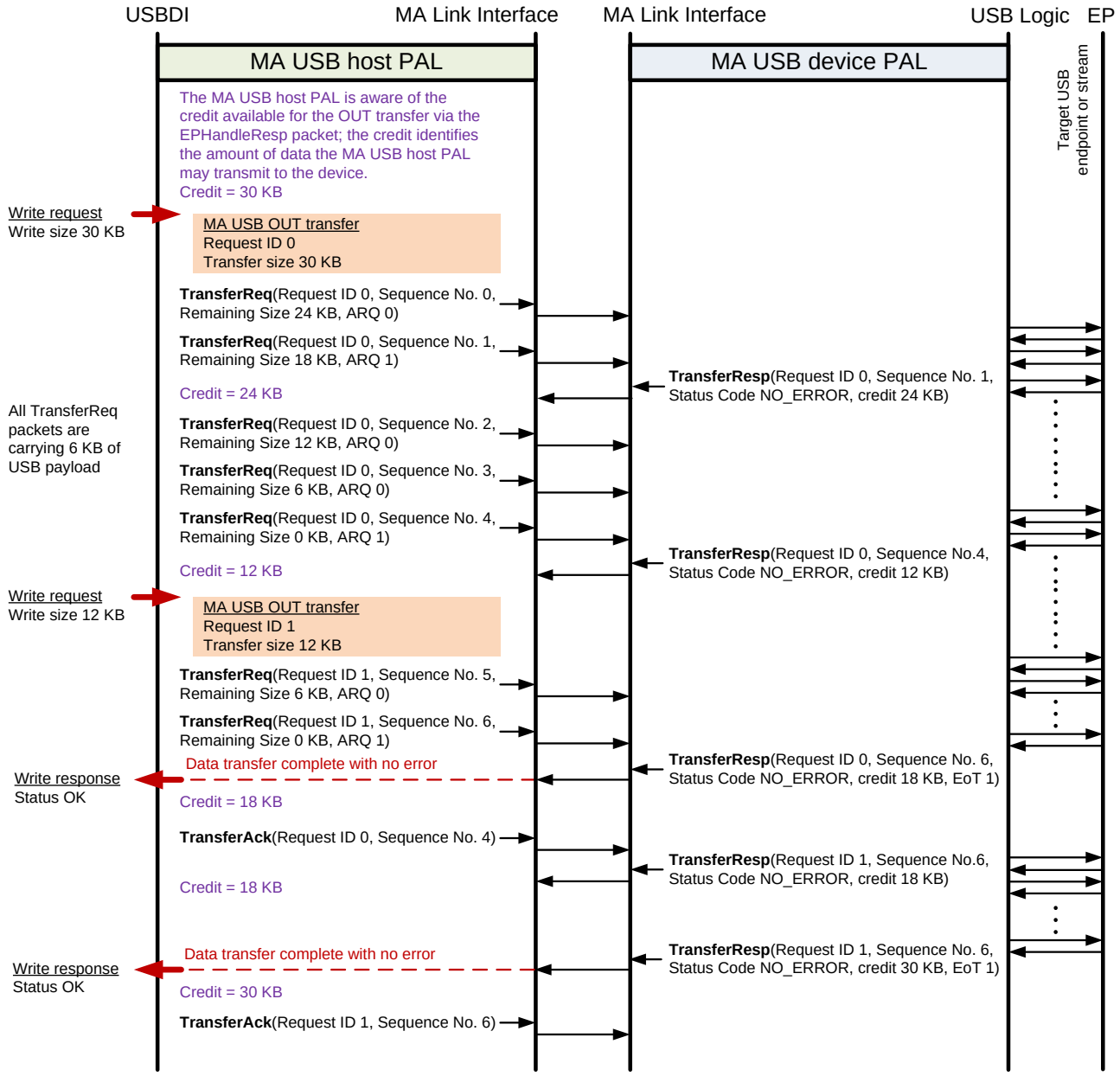


Figure 17—P-managed MA USB OUT transfers

To complete an MA USB OUT transfer, the MA USB host PAL transmits one or more TransferReq packets (Section 6.5.2) to the target MA USB device PAL to transfer the USB payload to the target USB endpoint or stream in the strict order it is available in the host system. Each TransferReq packet carries the same EP Handle, Stream ID, and Request ID as in the TransferReq packet that initiated the transfer. To enable the MA USB device PAL to identify missing data packets, each TransferReq packet carries a Sequence Number field, which is set to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state. The Sequence Number value is incremented by 1 after each new (i.e., not retried) TransferReq packet, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber. Sequence Number values keep increasing across successive transfers.

NOTE — Configuration events that return the state of an endpoint or stream flow to the initial state are triggered in response to various events at the MA USB host or device, and are normally communicated through appropriate management packets. As a result, the time an endpoint or stream flow makes the transition to its initial state depends

on whether the event is viewed from the MA USB host or from the target MA USB device perspective. For example, the MA USB host is assumed to have initialized a target endpoint flow when it transmits an EPHandleDeleteReq or USBDevDisconnectReq packet that targets the endpoint; the target MA USB device on the other hand, can be assumed to have initialized the endpoint flow when it transmits a corresponding response packet to the MA USB host with a status code that indicates the endpoint state was reset (e.g. status code of SUCCESS).

To avoid ambiguity in tracking the TransferResp packets pending completion, the MA USB host PAL shall have no more than (aMaxSequenceNumber +1)/2 (half the size of the Sequence Number space) outstanding TransferReq packets.

Each TransferReq packet includes a Remaining Size field that carries the number of remaining bytes the host expects to transmit to complete the transfer.

While an MA USB OUT transfer on a target endpoint or stream is in progress, the MA USB host PAL may transmit additional TransferReq packets to target the same endpoint or steam, with the value of the Request ID field incremented by 1 for each new transfer request. The target MA USB device queues the received TransferReq packets in the increasing order of Request ID and Sequence Number values for subsequent processing.

In response to a TransferReq packet with an unexpected Request ID value, or a TransferReq packet that is invalid independent of its Request ID and Sequence Number values, the target MA USB device PAL shall discard the TransferReq packet, and release to the assigned data channel a TransferResp packet within aTransferResponseTime from the moment it receives the invalid TransferReq packet, with the TransferResp packet Request ID and Sequence Number fields set to the Request ID and Sequence Number values of the last TransferReq packet received in order, and the Status Code field set to MISSING_REQUEST_ID if the received TransferReq packet is valid but shows a gap in its Request ID field, or INVALID_REQUEST if the received TransferReq packet is invalid independent of the value of its Request ID field. In response to a TransferResp packet with the Status Code field set to MISSING_REQUEST_ID or INVALID_REQUEST, the MA USB host PAL shall invalidate all outstanding TransferReq packets and start transmitting new TransferReq packets starting with the Request ID and Sequence Number values in the TransferResp packet.

NOTE — A retried TransferReq packet for an active transfer may not carry a different payload size, and may not change the size of the transfer.

NOTE — It is possible that a target MA USB device silently drops a valid incoming TransferReq packet if it cannot admit the packet into its buffer; this scenario is no different from the packet being lost over the medium, and is recovered though a detected gap in subsequent TransferReq packets, or through an MA USB host timeout.

In response to a valid TransferReq packet with the ARQ field set to 1, the target MA USB device PAL shall release a TransferResp packet to the assigned data channel with the Request ID and Sequence Number fields set to the Request ID and Sequence Number values the last TransferReq packet the MA USB device PAL has received in order. A TransferReq packet with the ARQ field set to 1, and the resulting TransferResp packet form an immediate exchange with timings and behavior defined in Section 5.2.1.2. A TransferResp packet acknowledges receipt of all TransferReq packets up to and including the Request ID and Sequence Number fields in the TransferResp packet.

NOTE — The acknowledged TransferReq packets may belong to multiple transfer requests with successive Request ID values; however, at least one TransferResp packet is transmitted for each transfer request to indicate the transfer conclusion.

The target MA USB device PAL shall deliver all received USB payload to the target endpoint or stream in the strict order of Sequence Number value, and transfer requests shall be completed in strict order of Request ID value. To notify the MA USB host PAL of the completion of the transfer on the target USB endpoint or stream, the MA USB device PAL shall transmit a TransferResp packet with the Request ID field identifying the completed transfer, the Sequence Number field set to the latest Sequence Number

value received by the device PAL, updated Credit field, the EoT field set to 1, and the Status Code field set accordingly.

NOTE — The time a TransferResp packet with EoT field set to 1 is transmitted and the value of the Request ID field in that packet are independent of other packet exchanges for subsequent transfers.

The MA USB host PAL shall acknowledge the receipt of a valid TransferResp packet with the EoT field set to 1 by transmitting a TransferAck packet to the MA USB device PAL with the Request ID, Sequence Number and Status Code fields set to the same values as those in the TransferResp packet. The TransferResp and TransferAck packets form an immediate exchange with timings and behavior defined in Section 5.2.1.2.

In response to a valid TransferReq packet with gap in the Sequence Number value, the target MA USB device PAL shall release a TransferResp packet to the assigned data channel within aTransferResponseTime from the moment the MA USB device PAL receives the TransferReq packet, with the Request ID and Sequence Number fields set to the Request ID and Sequence Number values the MA USB device PAL is expecting next, and the Status Code field set to MISSING_SEQUENCE_NUMBER.

If the target endpoint returns a STALL handshake during a transfer, the target MA USB device shall transmit a TransferResp packet to the MA USB host, with the Request ID field set to the Request ID value of the transfer a STALL handshake was returned for and Sequence Number field set to the Sequence Number value of the last TransferReq packet received in order, the EoT field is set to1, and the Status Code field set to TRANSFER_EP_STALL. Receiving a TransferResp packet with the Status Code field set to TRANSFER_EP_STALL results in completion of the transfer on the MA USB host. The MA USB device shall discard but acknowledge any subsequent data received for the transfer that experienced the STALL condition. Furthermore, until the MA USB host clears the STALL condition and the target EP handle returns to the Active state, the target MA USB device shall buffer and acknowledge each TransferReq packet received that does not belong to the transfer request that experienced the STALL condition by transmitting a TransferResp packet with the Status Code field set to INVALID_EP_HANDLE_STATE. The MA USB host shall not remove any unacknowledged data associated with an endpoint in STALL condition.

NOTE — Once the STALL condition is experienced, any TransferReq packet received results in transmitting a TransferResp packet with the Status Code field set to INVALID_EP_HANDLE_STATE.

If the MA USB device PAL receives a CancelTransferReq packet corresponding to an active transfer, it should discard all the data corresponding to the transfer identified in the CancelTransferReq packet, however it shall keep account of the sequence numbers of the TransferReq packets received. The specific behavior of the MA USB device and the MA USB host depends on the state of the transfer at the time that the CancelTransferReq is processed at the MA USB device:

- If the transfer was cancelled before any data was moved to the USB device (i.e., the Cancellation Status field in CancelTransferResp packet set to 1): The MA USB devices shall respond to a TransferReq packet for the cancelled transfer with a TransferResp packet with EOT set to 1, Status Code field set to TRANSFER_CANCELLED, and Sequence Number field set to aInvalidSequenceNumber. Additionally the MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.
- If the transfer was cancelled after some data was moved to the USB device (i.e., the Cancellation Status field in CancelTransferResp packet set to 2): The MA USB devices shall transmit a TransferResp packet with EOT set to 1, Status Code field set to TRANSFER_CANCELLED.

Additionally the MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.

- If the transfer was completed (i.e., the Cancellation Status field in CancelTransferResp packet set to 3): The MA USB devices shall transmit a TransferResp packet with EOT set to 1, Status Code field set to the appropriate value (the same value carried if the transfer was not cancelled). The MA USB device shall keep the state of the transfer until it receives the TransferAck packet from the MA USB host. The MA USB host shall transmit the TransferAck packet only after it has received all the TransferResp packets as well as the CancelTransferResp packet related to the cancelled transfer.
- If the transfer was not yet received (i.e., the Cancellation Status field in CancelTransferResp packet set to 4): The MA USB device is not required to keep cancel information for a transfer it has not yet received. If a TransferReq packet is received at the MA USB device, the MA USB device shall respond to it with no required knowledge of whether the transfer was previously cancelled. The MA USB host, following receipt of the CancelTransferResp packet, may either retransmit the CancelTransferReq or wait for the completion of the transfer.
- If the transfer with RequestID was serviced as part of ClearTransfersReq processing without any data being moved to the USB Device (i.e., the Cancellation Status field in CancelTransferResp packet set to 5): The MA USB device is not required to keep the cancel information for a transfer it already cleared, It shall generate CancelTransferResp without generating TransferResp. This status indicates that the MA-USB device doesn't expect TransferAck from MA USB host. The MA USB host, following receipt of the CancelTransferResp packet, shall not wait for TransferResp packets for this transfer.

If the MA USB device PAL receives a ClearTransfersReq packet corresponding to an endpoint with active transfers, it should discard all the data corresponding to the endpoint identified in the ClearTransfersReq packet, however it shall keep account of the sequence numbers of the TransferReq packets received. If the MA USB device receives a TransferReq packet for a cancelled transfer (TransferReq packets carrying a Request ID value less than the value indicated in the Start Request ID field in the ClearTransfersReq packet) and it has not yet delivered any data related to the transfer to the device, it shall discard the TransferReq packet, otherwise it shall respond with a TransferResp packet; if the transfer is cancelled before its completion (i.e., not all data related to the transfer is delivered to the target endpoint), the last TransferResp packet related to the transfer shall carry EOT field set to 1 with the Status Code field set to TRANSFER_CANCELLED. The Status Code field shall not be set to TRANSFER_CANCELLED if the transfer is completed. The MA USB host shall not remove the data for any unacknowledged TransferReq packets of a cancelled transfer before receiving the ClearTransfersResp packet. The MA USB host shall reset the Sequence Number value to 0 before transmitting a TransferReq packet with Request ID value indicated in the Start Request ID field in the ClearTransfersReq packet.

5.5.1 MA USB device buffer management for OUT transfers

The buffer space on the MA USB device is limited and variable for different implementations. In order to ensure efficient use of the medium, the OUT transfer protocol utilizes a credit-based end-to-end flow control to meter the flow of data from the MA USB host to the MA USB device to what can be successfully accepted into the MA USB device buffer.

The MA USB device PAL is responsible for managing its buffer space; it is required to inform the MA USB host PAL of its available buffer space for each USB endpoint behind the MA USB device PAL. The MA USB device PAL notifies the MA USB host PAL of the available buffer space for each endpoint by use of credits. The credits are the number of bytes of data that the MA USB device PAL is ready to accept into its buffer at the time that the credit information was submitted by the MA USB device PAL for transmission. In order to allow devices flexibility in managing the local buffer, an MA USB device will report to the MA USB host the Credit Consumption Unit (CCU) for each endpoint; CCU is the unit by which the MA USB host shall keep track of the buffer space available on the MA USB device. An MA USB device may allow the MA USB host to not strictly follow the credit advertised for the target endpoint by setting the Elastic Buffer Capability field in the MA USB Capability Response (CapResp) packet to 1. If the MA USB device supports the Elastic Buffer Capability, it may return a negative credit value if the memory currently consumed for target endpoint exceeds the credit advertised for the endpoint. The MA USB device additionally may choose to report a DROPPED_PACKET status every time it has to drop an incoming TransferReq packet because of buffer shortage.

The MA USB device delivers the initial credit for an endpoint to the MA USB host using the Buffer Size field in the EPHandleResp packet. Credits are allocated at the MA USB device PAL for each endpoint. Note that in case of an Enhanced SuperSpeed bulk OUT endpoint that supports Enhanced SuperSpeed Stream Protocol [USB 3.1], the allocated credits are the total available for all the streams. The counter used to track credits for each endpoint is initialized to the value of the Buffer Size field in the EPHandleResp packet at any configuration event intended to return the state of an endpoint to the initial state.

Credits are de-allocated when a deliverable TransferReq packet arrives and its data payload is accepted into the buffer. The MA USB device PAL uses the Sequence Number value in the TransferReq packet to determine whether the payload in the TransferReq packet is deliverable to the target endpoint. The data associated with a TransferReq packet targeted to an endpoint is deliverable if the data for all preceding TransferReq packets targeted to the endpoint has successfully been accepted. If the TransferReq packet has the Retry field set to 1 and the associated data has already been accepted by the MA USB device PAL, the data is dropped.

Credits are allocated when data is removed from the buffer. A target MA USB device PAL shall not remove any transfer data associated with an MA USB OUT transfer from its buffer unless it has successfully transmitted the data to the target endpoint or it receives one of the following packets:

- A DevResetReq packet (Section 6.3.18) or DevDisconnectReq packet (Section 6.3.36), in which case all the data for all endpoints and streams is removed from the MA USB device buffer.
- A ClearTransfersReq packet (Section 6.3.14), in which case all data related to corresponding endpoint(s) is removed from the MA USB device buffer.
- A USBDevDisconnectReq packet (Section 6.3.26) for the target USB device, in which case all data related to the target USB device is removed from the MA USB device buffer.
- A CancelTransferReq packet (Section 6.3.42), in which case all data related to the target request is removed from the MA USB device buffer.

The MA USB device PAL may modify the buffer space allocated to each endpoint at any time.

MA USB device shall provide credits for each of the endpoints in multiples of Credit Consumption Unit of that endpoint. The credits are delivered to MA USB host PAL using the following mechanisms:

- In Buffer size field in the EPHandleResp packet.

- The MA USB device may update the credit by retransmitting the latest TransferResp packet which had the EoT field set to 1, and including an updated value for the credit in the Credit field.
- The MA USB device may reduce the credit by retransmitting the latest TransferResp packet which had the EoT field set to 1, and including the reduced value for the credit in the Credit field. If the Elastic Buffer Capability bit in the CapResp packet is set to 0 then the local accounting for this credit allocation using the TransferResp packet is applied only after reception of the matching TransferAck packet.

The MA USB device PAL should attempt to reduce the overhead of transmitting TransferResp packets while ensuring the MA USB host does not stall waiting for credits.

The MA USB host PAL is able to compute the amount of credit available for a given endpoint at any given time by computing the difference between the last credit value and its associated Sequence Number value received from the MA USB device PAL and the amount of unacknowledged data transmitted. (Note that in case of retransmissions, the retransmitted data does not count toward unacknowledged data.) The MA USB host shall set its local counter for tracking credits on an endpoint to the value of the Credit field in a received TransferResp or EPHandleResp packet.

The MA USB host shall accurately track credits available and used during data streaming. Credits are consumed (released) when the TransferResp packet associated with those credits is received.

The MA USB host shall account for consumed credits in multiples of Credit Consumption Unit of that endpoint.

> NOTE — For example if an MA USB device reports a Credit Consumption Unit of 512 bytes for a particular endpoint, the MA USB host will always account for used credits for that endpoint in units of 512 regardless of the number of bytes transmitted; i.e., an MA USB payload of 2064 byte accounts for 5 Credit Consumption Units of 512 bytes.

If the Elastic Buffer Capability field in the CapResp packet is set to 0, the MA USB host PAL shall not transmit TransferReq packets targeted for an endpoint unless it has credits for transmitting data to that endpoint. Moreover, it shall limit the amount of data transmission to the target endpoint to the credit available for that endpoint.

5.5.2 Transfer description

The transfer description in this section does not cover all scenarios resulting from interaction of different events, and is provided to serve as an implementation guideline. The transfer description in Section 5.5 takes precedence over this description in case of a conflict.

Similar to IN transfers, the operation of the MA USB host PAL and a target MA USB device PAL is defined in terms of a set of state variables and *processes*, where each process is a basic unit of execution. Processes do not interrupt each other, meaning that a state variable is not modified during a process execution, unless it is modified by the process itself. State variables used to describe the device operation are marked with an 'R' (responder) subscript (e.g., $SeqNumber_R$), and variables used to describe the host operation are marked with an 'O' (originator) subscript (e.g., $SeqNumber_O$). An index notation is used for variables that have a scope of a single MA USB transfer request, e.g., $RemSize_R[r]$ denotes a state variable belonging to an MA USB transfer request with Request ID equal to r.

Figure 18 illustrates the data packets and header fields used to perform MA USB OUT transfers.

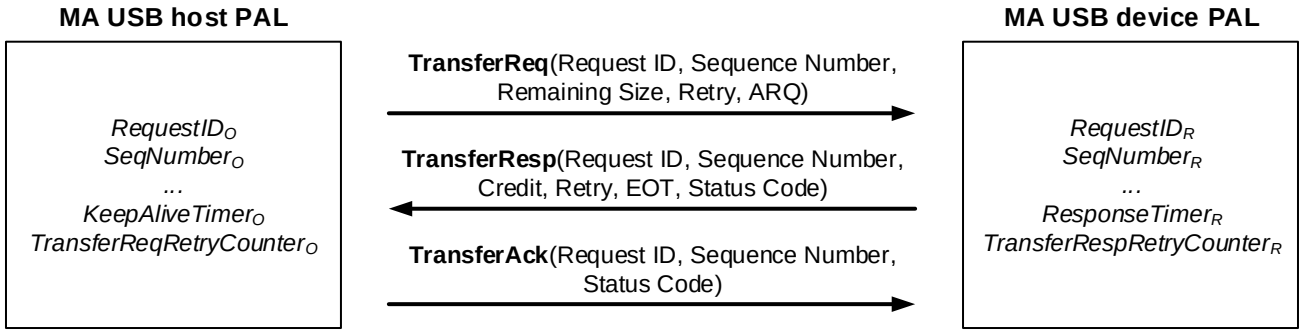


Figure 18—Data packets and header fields used for p-managed OUT transfers

5.5.2.1 MA USB host PAL operation

The MA USB host operation is defined in terms of a series of state variables maintained in the context of a single target endpoint or stream,

- *RequestID$_O$* (8 bits, unsigned): The Request ID value of the next original transfer request; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when a new transfer request is generated, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *ActiveRequestID$_O$* (8 bits, unsigned): The Request ID field of the active transfer request, i.e., the request expected to be served by the next original TransferReq packet; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when the entire payload belonging to the transfer has been transmitted (not necessarily acknowledged), with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *EarliestRequestID$_O$* (8 bits, unsigned): The Request ID value of the earliest transfer request whose state needs to be tracked; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when it is determined that the acknowledgement to the transfer completion has been received by the target MA USB device PAL, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- *SeqNumber$_O$* (24 bits, unsigned): The Sequence Number value to be placed into the next original TransferReq packet that the host transmits; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when an original TransferReq packet is transmitted, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- *EarliestUnacknowledged$_O$* (24 bits, unsigned): The Sequence Number value of the earliest transmitted but unacknowledged TransferReq packet; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented throughout the transfer as described below, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- *TransferReqRetries$_O$* (unsigned): The reload value of the *TransferReqRetryCounter$_O$*[] counters, set to aControlTransferRetries, aBulkTransferRetries or aInterruptTransferRetries for control, bulk and interrupt transfers respectively.

- *RxBufSize*$_O$ (25 bits, signed): The size (in bytes) of the target MA USB device buffer available to transfers based on the most recent TransferResp packet received. An *RxBufSize*$_O$ value of 0 or negative suggests that the MA USB device may not be able to accept more TransferReq packets, or may drop some of the already transmitted TransferReq packets assuming no change in the free buffer available to transfers since the received TransferResp packet was.

The remaining state variables have a finer scope of a single MA USB transfer targeting the endpoint or stream; specifically, for a given MA USB OUT transfer with Request ID equal to *r*,

- *RemSize*$_O$[*r*] (32 bits, unsigned): The number of bytes that the MA USB host needs to transmit to complete the transfer; set to the transfer size (in bytes) at transfer initialization, and decremented throughout the transfer.
- *TransferError*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon detecting an error in the transfer.
- *TransferComplete*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE when it is no longer necessary to track the state of the transfer.
- *TransferAcknowledged*$_O$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE upon releasing a TransferAck packet to the assigned data channel that acknowledges the TransferResp packet that indicated the completion of the transfer on the target endpoint or stream.
- *ResponseTimer*$_O$[*r*] (signed): A decrementing counter to track the response time for a TransferReq packet that requires an immediate exchange (Section 5.2.1.2); set to aTransferTimeout every time a TransferReq packet that requires an immediate exchange is released to the assigned data channel, and decremented by aTransferTimerTick at every transfer timer tick event.
- *TransferReqRetryCounter*$_O$[*r*] (unsigned): A decrementing counter to track the number of retries for a TransferReq packet belonging to the transfer that requires an immediate exchange (Section 5.2.1.2); set to *TransferReqRetries*$_R$ when a new round of retries is to be attempted, and decremented by 1 after each retry.
- *AckTransferSN*$_O$[*r*] (24 bits, unsigned): The value of the Sequence Number field in the last TransferReq packet that belongs to the transfer and has the ARQ field set to 1.

NOTE — The last TransferReq packet for a transfer that has the ARQ field set to 1 can be different from the TransferReq packet that carries the last portion of the payload belonging to the transfer. Also, setting the ARQ field to 1 in a TransferReq packet (including a retransmitted packet) cancels the immediate exchange associated with any earlier TransferReq packet with the ARQ field set to 1and belonging to the same transfer, i.e., for each transfer request, there can be only one TransferReq packet that is waiting for acknowledgement.

The MA USB host PAL operation is defined in terms of the processes described below.

Initialization process

Invoked on any configuration event intended to return the state of the endpoint or stream flow to the initial state.

I. Set *RequestID*$_O$ = 0.
II. Set *SeqNumber*$_O$ = 0.
III. Set *EarliestUnacknowledged*$_O$ = 0.
IV. Set *RxBufSize*$_O$ = Initial size of the target MA USB device buffer available to transfers as specified in Section 5.5.

NOTE — The $SeqNumber_O$ and $EarliestUnacknowledged_O$ variables keep increasing across successive transfers.

Transfer initialization process (starting a new transfer)

Invoked every time a new transfer request is initiated.

I. Select $r = RequestID_O$ as the Request ID assigned to the transfer.
II. Initialize the following variables to manage the transfer,
  (a) Set $TransferError_O[r] = FALSE$.
  (b) Set $TransferComplete_O[r] = FALSE$.
  (c) Set $EndOfTransferDetected_O[r]$ = FALSE.
  (d) Set $TransferAcknowledged_O[r]$ = FALSE.
  (e) Set $ResponseTimer_O[r] = -1$.
  (f) Set $RemSize_O[r]$ = Transfer size in bytes, as indicated by the application.
III. If there is no active request,
  (a) Set $ActiveRequestID_O = RequestID_O$.
  (b) Set $EarliestRequestID_O = RequestID_O$.
IV. Set $RequestID_O = RequestID_O + 1$.

NOTE — The difference between $RequestID_O$ and $EarliestRequestID_O$ does not exceed (aMaxRequestID + 1)/2 at any point in time; the difference may be further limited by the total number of outstanding transfers that the target MA USB device supports across all its endpoints and streams.

NOTE — The Status Code field in all outgoing TransferReq packets is set to NO_ERROR.

NOTE — Handling of possible local transmission failures is implementation-dependent.

TransferReq generation process

Invoked to generate TransferReq packets as long as there is a transfer request.

I. While $ActiveRequestID_O < RequestID_O$,
  (a) Set $r = ActiveRequestID_O$.
  (b) Let *PayloadSize* (implementation-dependent) denote the payload size for the next TransferReq packet to be generated, with $0 < PayloadSize \leq RemSize_O[r]$.
  (c) Set $RemSize_O[r] = RemSize_O[r] - PayloadSize$.
  (d) Submit a TransferReq packet to the TransferReq transmission process, with the Request ID field set to *r*, the Sequence Number field set to $SeqNumber_O$, the Remaining Size field set to $RemSize_O[r]$, the Retry field set to 0, and the ARQ field set to 0 or 1.
  (e) Set $SeqNumber_O = SeqNumber_O + 1$.
  (f) Set $RemSize_O[r] = RemSize_O[r] - PayloadSize$.
  (g) If $RemSize_O[r] = 0$, set $ActiveRequestID_O = ActiveRequestID_O + 1$.

NOTE — Transfer payload is not removed from the MA USB host memory until an acknowledgement is received that indicates the completion of the transfer on the target endpoint or stream.

TransferReq transmission process

Invoked to release a TransferReq packet to the assigned data channel.

I. Let *r*, *SN* and *AckRequest* respectively denote the values of the Request ID, Sequence Number and ARQ fields in the TransferReq packet. Let *PayloadSize* denote the payload size in the TransferReq packet.
II. If $\lceil PayloadSize/CCUnit \rceil \times CCUnit \leq RxBufSize_O[r]$, and $SeqNumber_O < EarliestUnacknowledged_O + [(aMaxSequenceNumber + 1)/2]$,
  (a) Release the TransferReq packet to the assigned data channel

(b) Set $RxBufSize_O = RxBufSize_O - \lceil PayloadSize/CCUnit \rceil \times CCUnit$
(c) If *AckRequest* = 1,
(1) Set $AckTransferSN_O[r] = SN$.
(2) Set $ResponseTimer_O[r]$ = aTransferTimeout.
(3) Set $TransferReqRetryCounter_O[r] = TransferReqRetries_O$.

III. Otherwise, try to transmit the packet at a later time (flow control event).

NOTE — CCUnit is the target endpoint credit consumption unit, as indicated by the Credit Consumption Unit (CCU) field in the EPHandleResp packet returning the EP handle of the target endpoint (Section 7.3.2.2).

NOTE — If the target MA USB device PAL indicates support for Elastic Buffer Capability, the MA USB host may still proceed with transmitting a TransferReq packet with $RxBufSize_O < \lceil PayloadSize/CCUnit \rceil \times CCUnit$. Implementations must balance the trade-off between aggressive transmission and possible packet loss as a result of buffer overrun at the target MA USB device.

TransferResp reception process

Invoked at the reception of a TransferResp packet.

I. Let *r*, *SN*, *Credit*, *EndOfTransfer* and *Status* respectively denote the values of the Request ID, Sequence Number, Credit, EoT, and Status Code fields of the TransferResp packet.
II. If $EarliestRequestID_O - [(aMaxRequestID + 1)/2] \leq r < EarliestRequestID_O$, or $r > RequestID_O$ drop the packet.
III. Otherwise, if $SN < EarliestUnacknowledged_O$ or $SN \geq SeqNumber_O$, drop the packet.
IV. Otherwise, if $TransferComplete_O[r]$ = TRUE drop the packet.
V. Otherwise,
(a) If *EndOfTransfer* = 1,
(1) Submit a TransferAck packet to the TransferAck transmission process with the Request ID field set to *r*, the Sequence Number field set to *SN*, and the Status Code field set to *Status*.
(b) If *Status* = *SUCCESS*,
(1) Set $EarliestUnacknowledged_O = SN + 1$.
(c) If *Status* = *MISSINNG_SEQUENCE_NUMBER* or *DROPPED_PACKET*,
(1) Set $EarliestUnacknowledged_O = SN$.
(2) Rewind the TransferReq packet transmission sequence so that the next TransferReq packet to be transmitted will be the one with Sequence Number value *SN* (the target MA USB device has dropped this TransferReq packet, and has dropped or will drop all TransferReq packets that have been transmitted after this packet).
(d) For each open transfer request *u* in the interval $EarliestRequestID_O \leq u \leq r$ and in the increasing order of *u*,
(1) If $ResponseTimer_O[u] > 0$ and $AckTransferSN_O[u] < EarliestUnacknowledged_O$,
(i) Set $ResponseTimer_O = 0$.
(2) Otherwise, break the loop.
(e) Set $RxBufSize_O = Credit$ – Sum of $\lceil PayloadSize/CCUnit \rceil \times CCUnit$ for all transmitted and unacknowledged TransferReq packets, i.e., TransferReq packets with Sequence Number values $EarliestUnacknowledged_O$ to $SeqNumber_O - 1$.

NOTE — At this point the target MA USB device has dropped or will drop any outstanding TransferReq packets with Sequence Number larger than *SN*. The MA USB host should attempt to stop transmission of any TransferReq packets queued for transmission, and depending on the outcome of the cancellation, it may choose to ignore up to $SeqNumber_O - 1 - SN$ follow-on TransferResp packets from the target MA USB device reporting a

MISSING_SEQUENCE_NUMBER status, except possibly to use these packets to update its estimate of the target MA USB device available buffer. This optional behavior is not reflected in the transfer description.

NOTE — A *DROPPED_PACKET* status indicates that a TransferReq packet was received but could not be admitted into local buffer; the MA USB host PAL may use this additional knowledge to pace its transmission rate.

(1) If *Status* equals any of the non-recoverable transfer errors,
(i) Set $TransferError_O[r] = TRUE$.

TransferAck transmission process

Invoked to release a TransferAck packet to the assigned data channel.

I. Let *r* and *SN* respectively denote the values of the Request ID and Sequence Number fields of the received packet.
II. Release the TransferAck packet to the assigned data channel.
III. For each open transfer request *u* in the interval $EarliestRequestID_O \le u \le r$,
(a) Set $TransferAcknowledged_O[r]$ = TRUE.
(b) Set $TransferCompletionTimer_O[u]$ = aMaxTransferLifetime.

Timer process

Invoked at every transfer timer tick event, as long as there is an active transfer request.

I. For each open transfer request *u* in the interval $EarliestRequestID_R \le u < ActiveRequestID_R$, and in increasing order of *u*,
(a) If $TransferAcknowledged_O[u]$ = TRUE,
(1) Set $TransferCompletionTimer_O$[u] = $TransferCompletionTimer_O$[u] – aTransferTimerTick.
(2) If $TransferCompletionTimer_O[u] \le 0$, set $TransferComplete_O[u]$ = TRUE.
(3) Otherwise, set $EarliestRequestID_O = u$ and break the loop (go to Step II).
(b) Otherwise, set $EarliestRequestID_O = u$ and break the loop (go to Step II).
II. For each open transfer request *u* in the interval $EarliestRequestID_O \le u \le r$, and in the increasing order of *u*,
(a) If $ResponseTimer_O[u] > 0$,
(1) Set $ResponseTimer_O[u] = ResponseTimer_O[u]$ – aTransferTimerTick.
(2) If $ResponseTimer_O[u] \le 0$,
(i) If $TransferReqRetryCounter_O[u] > 0$,
(a) Submit the TransferReq packet that requires acknowledgement to the TransferReq transmission process, with all packet fields the same, except the Retry field, which is set to 1.
(b) Set $ResponseTimer_O[u]$ = aTransferTimeout.
(c) Set $TransferReqRetryCounter_O[u] = TransferReqRetryCounter_O[u] - 1$.
(ii) Otherwise,
(a) Start the Ping protocol (Section 5.2.2).
(b) If the Ping protocol is successful, optionally,
(1) Set $ResponseTimer_O[u]$ = aTransferTimeout.
(2) Set $TransferReqRetryCounter_O[u] = TransferReqRetries_O$.
(3) Quit the Timer process and wait for the next transfer timer tick event.
(c) If the Ping protocol fails,

(1) Set $TransferError_O[u]$ = TRUE.
(2) Indicate a transfer timeout error to the application, and wait for a corrective action.

5.5.2.2 MA USB device PAL operation

The MA USB device operation is defined in terms of a series of state variables maintained in the context of a single target endpoint or stream,

- $RequestID_R$ (8 bits, unsigned): The Request ID value of the next original transfer request; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when a new transfer request is accepted, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- $EarliestRequestID_R$ (8 bits, unsigned): The Request ID value of the earliest transfer request whose state needs to be tracked; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 when the transfer completion on the target endpoint or stream has been acknowledged, with wraparound to 0 after reaching the maximum value of aMaxRequestID.
- $SeqNumber_R$ (24 bits, unsigned): Expected value of the Sequence Number field of the next original TransferReq packet to be received; initialized to 0 at session initialization or upon any configuration event that returns the state of the endpoint or stream flow to the initial state, and incremented by 1 every time an original TransferReq packet is received and successfully admitted to the receive buffer, with wraparound to 0 after reaching the maximum value of aMaxSequenceNumber.
- $ResponseTimer_R$ (signed): A decrementing counter to track the response time for a TransferResp packet that requires an immediate exchange (Section 5.2.1.2); set to aTransferTimeout every time a TransferResp packet that requires an immediate exchange is released to the assigned data channel, and decremented by aTransferTimerTick at every transfer timer tick event.
- $TransferRespRetryCounter_R$ (unsigned): A decrementing counter to track the retries of a TransferResp packet that requires an immediate exchange (Section 5.2.1.2); set to $TransferRespRetries_O$ when a new round of retries is to be attempted, and decremented by 1 after each retry.
- $TransferRespRetries_R$ (unsigned): The reload value of the $TransferRespRetryCounter_R$ counter, set to aControlTransferRetries, aBulkTransferRetries or aInterruptTransferRetries for control, bulk and interrupt transfers, respectively.
- $RxBufSize_R$ (32 bits, unsigned): Size (in bytes) of the receive buffer available to the target endpoint or stream.
- $Occupancy_R$ (unsigned): Size (in bytes) of the payload accepted into the receive buffer.

The remaining state variables have a finer scope of a single MA USB transfer targeting the endpoint or stream; specifically, for a given MA USB OUT transfer with Request ID equal to *r*,

- $RemSize_R[r]$ (32 bits, unsigned): The number of bytes expected to be received; set to the value of the Remaining Size field in the first TransferReq packet belonging to the transfer, and decremented throughout the transfer.
- $TransferError_R[r]$ (boolean): Set to FALSE at transfer initialization, and set to TRUE upon detecting an error in the transfer.

- *TransferComplete*$_R$[*r*] (boolean): Set to FALSE at transfer initialization, and set to TRUE when it is no longer necessary to track the state of the transfer.
- *EndOfTransferDetected*$_R$[*r*]: A boolean variable initialized to *FALSE* at transfer initialization, and set to *TRUE* when the transfer payload has been delivered to the target endpoint or stream, with or without error.

The MA USB device PAL operation is defined in terms of the processes described below.

Initialization process

Invoked on any configuration event intended to return the state of an endpoint or stream flow to the initial state.

I. Set *RequestID*$_R$ = 0.
II. Set *EarliestRequestID*$_R$ = 0.
III. Set *SeqNumber*$_R$ = 0.
IV. Set *Occupancy*$_R$ = 0.
V. Set *RxBufSize*$_R$ = Initial value of the receive buffer size.

NOTE — The *SeqNumber*$_R$ variable keep increasing across successive transfers.

TransferReq reception process

Invoked upon reception of a TransferReq packet.

I. Let *r*, *SN*, *RemSize*, and *AckRequest* respectively denote the values of the Request ID, Sequence Number, Remaining Size, and ARQ fields in the received TransferReq packet. Let *PayloadSize* denote the size of the payload in the TransferReq packet.
II. If *r* < *EarliestRequestID*$_R$, drop the packet and submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to *ActiveRequestID*$_R$, the Sequence Number field set to *SeqNumber*$_R$ – 1, the Credit field set to *RxBufSize*$_R$ – *Occupancy*$_R$, the Retry and EoT fields set to 0, and the Status Code field set to INVALID_REQUEST.
III. Otherwise, if *r* > *RequestID*$_R$, drop the packet and submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to *ActiveRequestID*$_R$, the Sequence Number field set to *SeqNumber*$_R$ – 1, the Credit field set to *RxBufSize*$_R$ – *Occupancy*$_R$, the Retry and EoT fields set to 0, and the Status Code field set to MISSING_REQUEST_ID.
IV. Otherwise, if *SN* > *SeqNumber*$_R$, drop the packet and submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to *ActiveRequestID*$_R$, the Sequence Number field set to *SeqNumber*$_R$ – 1, the Credit field set to *RxBufSize*$_R$ – *Occupancy*$_R$, the Retry and EoT fields set to 0, and the Status Code field set to MISSING_SEQUENCE_NUMBER.
V. Otherwise, if *SN* < *SeqNumber*$_R$, drop the packet.
VI. Otherwise, if *r* = *RequestID*$_R$ and there is an active transfer request, and *RemSize*$_R$[*r*–1] > 0, drop the packet and submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to *ActiveRequestID*$_R$, the Sequence Number field set to *SeqNumber*$_R$ – 1, the Credit field set to *RxBufSize*$_R$ – *Occupancy*$_R$, the Retry and EoT fields set to 0, and the Status Code field set to INVALID_REQUEST.
VII. Otherwise,
    (a) Optionally, invoke the buffer size change process.

(b) If $PayloadSize + Occupancy_R \leq RxBufSize_R$, or if the Elastic Buffer (Section 5.5.1) capability is supported,
  (1) If $r = RequestID_R$,
    (i) Create a new transfer with Request ID value r.
    (ii) Set $TransferError_R[r]$ = FALSE.
    (iii) Set $TransferComplete_R[r]$ = FALSE.
    (iv) Set $EndOfTransferDetected_R[r]$ = FALSE.
    (v) Set $RequestID_R = RequestID_R + 1$.
  (2) Set $SeqNumber_R = SeqNumber_R + 1$.
  (3) Set $Occupancy_R = Occupancy_R + PayloadSize$.
  (4) If $AckRequest = 1$, or $Occupancy_R > RxBufSize_R$, or optionally,
    (i) Submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to $ActiveRequestID_R$, the Sequence Number field set to $SeqNumber_R - 1$, the Credit field set to $RxBufSize_R - Occupancy_R$, the Retry and EoT fields set to 0, and the Status Code field set to NO_ERROR.
(c) Otherwise,
  (1) Drop the TransferReq packet.
  (2) Submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to r, the Sequence Number field set to $SeqNumber_R - 1$, the Credit field set to $RxBufSize_R - Occupancy_R$, the Retry and EoT fields set to 0, and the Status Code field set to DROPPED_PACKET if the Drop Notification (Section 6.3.3.2) capability is supported, and set to MISSING_SEQUENCE_NUMBER otherwise.

TransferResp transmission process

Invoked to release a TransferResp packet to the assigned data channel.

I. Let *EndOfTransfer* denote the value of the EoT fields in the TransferResp packet.
II. Release the TransferResp packet to the assigned data channel.
III. If $EndOfTransfer = 1$, and $ResponseTimer_R \leq 0$,
  (a) Set $ResponseTimer_R$ = aTransferTimeout.
  (b) Set $TransferRespRetryCounter_R = TransferRespRetries_R$.

TransferAck reception process

Invoked upon receiving a TransferAck packet.

I. Let r denote the value of the Request ID field in the received TransferAck packet.
II. If $r < EarliestRequestID_R$ or $r > RequestID_R$, drop the TransferAck packet.
III. Otherwise,
  (a) For each open transfer request u in the interval $EarliestRequestID_R \leq u \leq r$, and in increasing order of u, set $TransferComplete_R[u]$ = TRUE.

Payload delivery process

Invoked to deliver the transfer payload to the target endpoint or stream.

I. Attempt to deliver the payload to the target endpoint or stream.

Payload delivery confirmation process

Invoked to indicate successful or failed completion of payload delivery for transfer request r to the target endpoint or stream.

I. If payload delivery has failed, or optionally,
   (a) Set $EndOfTransferDetected_R[r]$ = TRUE.
   (b) Submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to *r*, the Sequence Number field set to $SeqNumber_R - 1$, the Credit field set to $RxBufSize_R - Occupancy_R$, the Retry field set to 0, the Retry field set to 0, the EoT field set to 1, and the Status Code field set to NO_ERROR or appropriate error code.

NOTE — A TransferResp packet with the Request ID field set to *r* and the EoT field set to 1 indicates successful or failed delivery of the payload belonging to transfer *r* to the target endpoint or stream, as well as successful delivery of the payload belonging to all past transfer requests.

Buffer size change process

Invoked to change the buffer size available to the target endpoint or stream.

I. Change the receive buffer size, and
   (a) Set $RxBufSize_R$ to the new buffer size value.
   (b) Optionally, submit a TransferResp packet to the TransferResp transmission process with the Request ID field set to $ActiveRequestID_R$, the Sequence Number field set to $SeqNumber_R - 1$, the Credit field set to $RxBufSize_R - Occupancy_R$, the Retry and EoT fields set to 0, and the Status Code field set to NO_ERROR.

NOTE — The buffer used by a transfer that has completed delivery to the target endpoint or stream cannot be assumed to be available before receiving the MA USB host PAL acknowledgement to the delivery completion.

Timer process

Invoked at every transfer time event, as long as there is an active transfer request.

I. If $ResponseTimer_R > 0$,
   (a) Set $ResponseTimer_R = ResponseTimer_R$ – aTransferTimerTick.
   (b) If $ResponseTimer_R \leq 0$,
      (1) If $TransferRespRetryCounter_R > 0$,
         (i) Submit the earliest TransferResp packet that requires acknowledgement to the TransferResp transmission process with all packet fields the same, except possibly the Sequence Number field, which is set to $SeqNumber_R - 1$, and the Credit field, which is set to $RxBufSize_R - Occupancy_R$.
         (ii) Set $ResponseTimer_R$ = aTransferTimeout.
         (iii) Set $TransferRespRetryCounter_R = TransferRespRetryCounter_R - 1$.
      (2) Otherwise, wait indefinitely for corrective actions, or optionally,
         (i) Start the Ping protocol (Section 5.2.2).
         (ii) If the Ping protocol is successful, optionally,
            (a) Set $ResponseTimer_R$ = aTransferTimeout.
            (b) Set $TransferRespRetryCounter_R = TransferRespRetries_R$.
            (c) Quit the Timer process and wait for the next transfer timer tick event.
         (iii) If the Ping protocol fails, or if the optional path in the previous step is not followed,
            (a) Set $TransferError_R[EarliestRequestID_R]$ = TRUE.
            (b) Indicate a transfer timeout error to the application, and wait for a corrective action.

5.6 Link-managed OUT transfers

The link-managed (l-managed) transfer model assumes a connection-oriented operation with end-to-end flow control between the target USB endpoint and the MA USB host PAL. It simplifies the transfer protocol by relying on link-level flow control.

A complete l-managed transfer consists of three phases,

- *Set up phase*, when link resources are set up and allocated to the transfer
- *Data phase*, when data flows between the client buffer on the host system and the target USB endpoint
- *Tear down phase*, when the link resources allocated to the transfer are released

MA USB host normally follows the same model for all transfers targeting a given USB endpoint as long as the endpoint is active. For example, to write to a bulk OUT endpoint on a continuous basis, the MA USB host may set up the required link resources to support a link-managed transfer once, use the established flow-controlled connection to transfer data to the target endpoint as long as the endpoint is active, and tear down the connection only after the target USB endpoint is not operational. Alternatively, the MA USB host may frequently release the link resources allocated to a transfer and allocate them to a different transfer targeting a different endpoint on a time-shared basis, making the set up and tear down phases more frequent.

5.6.1 Transfer description

The MA USB host starts the set up phase of an l-managed OUT transfer by sending a Transfer Setup Request (TransferSetupReq) packet (Section 6.4.1) to the target MA USB device over the control or any available data channel, indicating the EP handle the host will be writing to, as well as the link-specific context for the flow-controlled connection it will be setting up for data transfer. MA USB host then sets up the flow-controlled connection required for the transfer. The connection set up may require participation of the target MA USB device (e.g., receive buffer size negotiation), which is already aware of the set up through the TransferSetupReq packet. In the event the connection set up fails, MA USB host may retry setting up the connection for a number of times, or may attempt to set up another connection using a new link-specific context. MA USB host shall share the connection context with the MA USB device through a TransferReq packet before starting any new connection set up. If no connection can be secured to support the transfer, MA USB host shall release any local or link-layer resources allocated to the transfer, and return the appropriate USBDI error to the application to indicate the transfer failure. No action by the target MA USB device is required in this case. If the connection set up is successful, the target MA USB device programs local resources to direct the OUT transfer payload arriving over the established connection to the target endpoint, and sends a Transfer Setup Response (TransferSetupResp) packet (Section 6.4.2) to the MA USB host, completing the set up phase.

If the MA USB host does not receive a TransferSetupResp packet by aTransferTimeout after the connection has been established from the MA USB host perspective, the MA USB host shall send another TransferSetupReq packet with the packet Retry bit set to 1, and repeat this step for a maximum of aTransferSetupRetries times. The parameter aTransferSetupRetries does not include the number of possible local retries before a TransferSetupReq packet is successfully submitted for transmission.

The MA USB host starts the data phase of the transfer by sending one or more TransferReq packets (Section 6.5.2) to the target MA USB device over the established connection. There is no restriction on the TransferReq packets transmission rate; the underlying assumption behind the transfer model is that TransferReq packets will be flow-controlled at the link layer if the target endpoint follows a slower pace than the link-level transfer rate. With the exception of the last TransferReq packet (which does not have

a payload size restriction), the payload size for each TransferReq packet belonging to a transfer is a multiple of the maximum packet size the target endpoint supports, i.e., a multiple of the *wMaxPacketSize* field of the target endpoint descriptor. All TransferReq packets belonging to the same transfer carry the same number for the Request ID field. Request ID is selected by the MA USB host and shall be unique within the scope of a target endpoint. The MA USB host may send other TransferReq packets with different Request IDs before it has received all required TransferResp packet for a given Request ID. The Request ID used for each new transfer shall be strictly increasing, with no gaps. The MA USB host shall not have more than one pending transfer using the same Request ID and targeting the same endpoint. To enable the MA USB device to identify missing data packets, each TransferReq packet carries a sequence number, which is set to zero in the first TransferReq packet for a given transfer, and is incremented by one in every subsequent TransferReq packet. In addition, each TransferReq packet carries in its Transfer Size field the number of remaining bytes for the transfer assuming the payload in the packet is successfully processed by the MA USB device.

The target MA USB device shall send a Transfer Response (TransferResp) packet (Section 6.5.3) to the MA USB host once one of the following conditions occurs,

- The payload in the final TransferReq packet belonging to a transfer (as marked by Remaining Size field set to zero) is successfully delivered to the target endpoint: In this case, the target MA USB device shall send a TransferResp packet with the Request ID and Sequence Number fields set to the same values as those in the final TransferReq packet, the Remaining Size field set to 0, and the Status field set to SUCCESS.
- Delivering the payload in any of the TransferReq packets belonging to a transfer to the target endpoint faces a locally non-recoverable error: In this case, the target MA USB device shall send a TransferResp packet with the Request ID and Sequence Number fields set to the same values as those in the TransferReq packet whose payload experienced the error, the Remaining Size field set to the transfer size minus the number of bytes successfully delivered to the endpoint, and the Status field set to an appropriate error code from the list in Table 6.

The interval between when the final TransferReq packet is received by the target MA USB device and when the corresponding TransferResp packet appears over the link on the MA USB device side shall not exceed aTransferKeepAlive. If the MA USB host does not receive a TransferResp packet by aTransferKeepAlive after it has successfully submitted the final TransferReq packet for a transfer for transmission, it shall attempt to solicit a TransferResp packet by sending another TransferReq packet with all packet fields set to the same values as those in the final TransferReq packet for the transfer, except for the Retry bit, which is set to 1. If necessary, the MA USB host shall repeat sending the TransferReq packet for a maximum number of times equal to aControlTransferRetries for control transfers, aBulkTransferRetries for bulk transfers and aInterruptTransferRetries for interrupt transfers. The number of retries does not include the number of possible local retries before a TransferReq packet is successfully submitted for transmission. If no TransferResp packet is received after the maximum number of retries allowable for the transfer type, the MA USB host shall return the appropriate USBDI error to the application indicating the transfer failure, and shall start the tear down phase of the transfer.

From the MA USB host perspective, an OUT transfer is complete when one of the following conditions occurs,

- MA USB host receives a TransferResp packet with the Status field set to a value other than SUCCESS: In this case, MA USB host shall return to the application a status code that indicates failure of the write request that prompted the MA USB transfer, as well as the reason for the failure. MA USB host may also return the number of bytes confirmed to be delivered to the target endpoint.

- MA USB host receives a TransferResp packet with the Status field set to SUCCESS and the Remaining Size field set to zero: In this case, if the final TransferReq packet associated with the transfer has not been sent, or the Sequence Number in the TransferResp packet does not equal the Sequence Number in the final TransferReq packet, the MA USB host shall return to the application a status code that indicates failure of the write request that prompted the MA USB transfer, as well as the reason for the failure. The MA USB host may additionally return the number of bytes that were confirmed to be delivered to the target endpoint. If the final TransferReq packet associated with the transfer has been sent, and the Sequence Number in the TransferResp packet equals the Sequence Number in the final TransferReq packet, the MA USB host shall assume the OUT transfer is complete with no error. Once all OUT transfers serving a common application-level write request are complete with no error, the MA USB host shall return a status code to the application that indicates the success of the write request.

At any point during the set up or data phase of an l-managed transfer, the MA USB host may start the tear down phase by tearing down the lower-layer link resources set up in support of the transfer, and sending a Transfer Tear Down Confirmation (TransferTearDownConf) packet (Section 6.4.3) to the target MA USB device on the control channel. The TransferTearDownConf packet is not retried at the PAL level. It is simply a confirmation that the MA USB host will not send more TransferReq packets on the target endpoint following the packet. A target MA USB device that detects releasing of lower-layer link resources associated with an l-managed transfer on a target endpoint, or receives a TransferTearDownConf packet for an l-managed transfer on a target endpoint, shall stop sending TransferResp packets for any pending transfer requests, and shall silently discard all transfer requests associated with the target endpoint.

Figure 19 illustrates the three phases of l-managed OUT transfers. In the scenario shown in the figure, the first OUT transfer (which completes with no error) goes through set up and data phases, and the second transfer (which experiences a STALL error) goes through data and tear down phases.

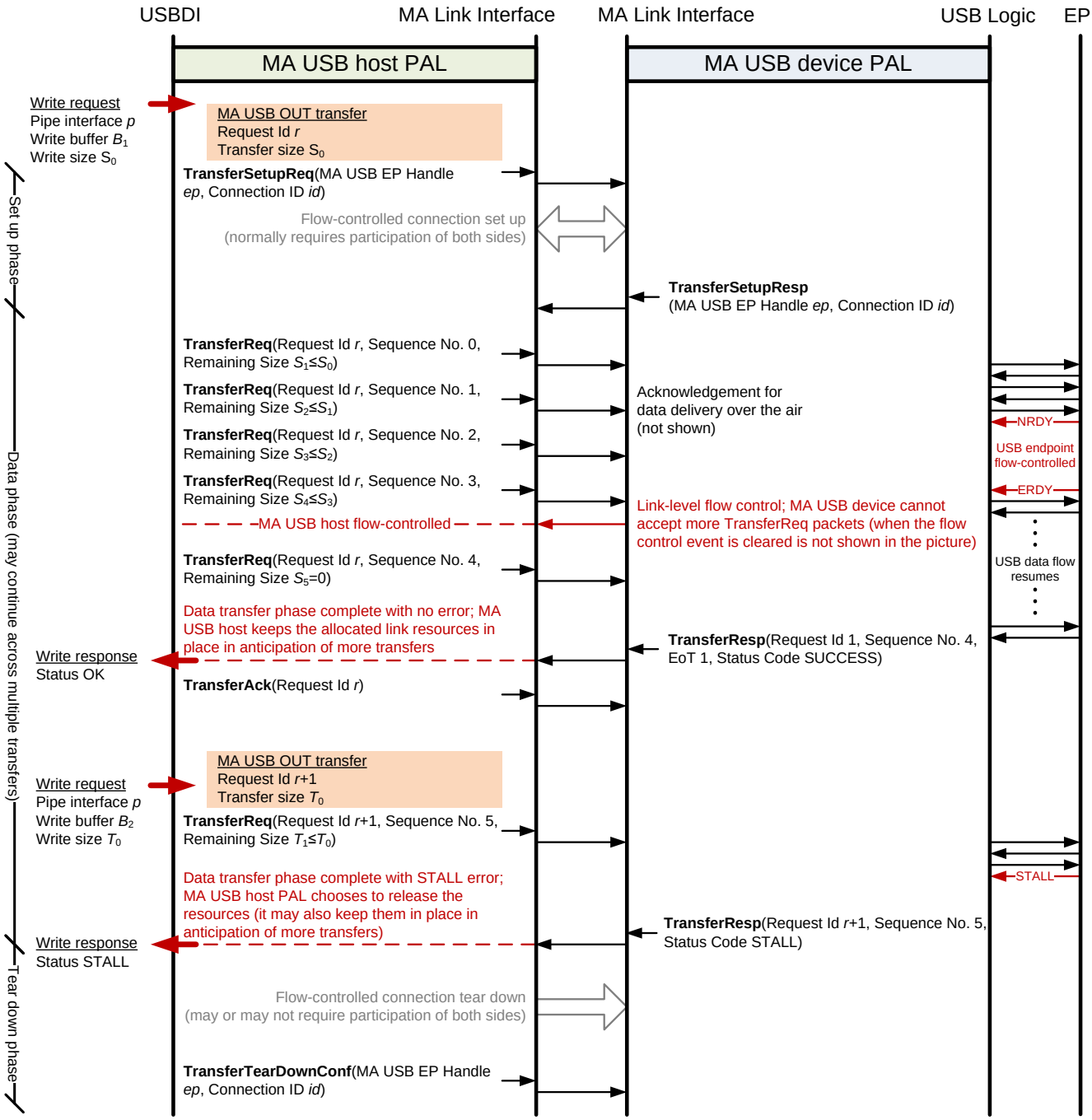


Figure 19—Link-managed OUT transfer

5.6.2 Transfer mode selection

When targeting a non-isochronous OUT endpoint, the exercised transfer mode is decided by the MA USB host. If the target endpoint indicates support for l-managed transfer mode in its MA USB EP descriptor (Section 6.3.7), the MA USB host may choose to exercise either the p-managed or l-managed transfer mode when targeting the endpoint. The MA USB host may switch from one transfer mode to another. Specifically, successful set up of an l-managed transfer indicates switch to l-managed mode, and successful tear down of an l-managed transfer indicates switch to p-managed mode. Note an l-

managed transfer set up may be unsuccessful due to resource limitations, even if the target endpoint has indicated support for l-managed transfer mode.

5.7 Control transfers

MA USB control transfers are used for USB command and status operations. As such, MA USB control transfers are defined in relation to USB control transfers specified in [USB 2.0] and [USB 3.1].

A USB control transfer comprises three stages,

1. Setup stage, during which the USB host transmits to the USB device the USB setup data. The USB setup data is an eight-byte payload.
2. Data stage, which may be an IN or an OUT data stream or non-existent
3. Status stage, which is a separate identifiable bus event where the device acknowledges the completion of the command to the host.

MA USB control transfers (Figure 20) manage the three stages of the USB control transfer using the mechanisms defined in the following sections.

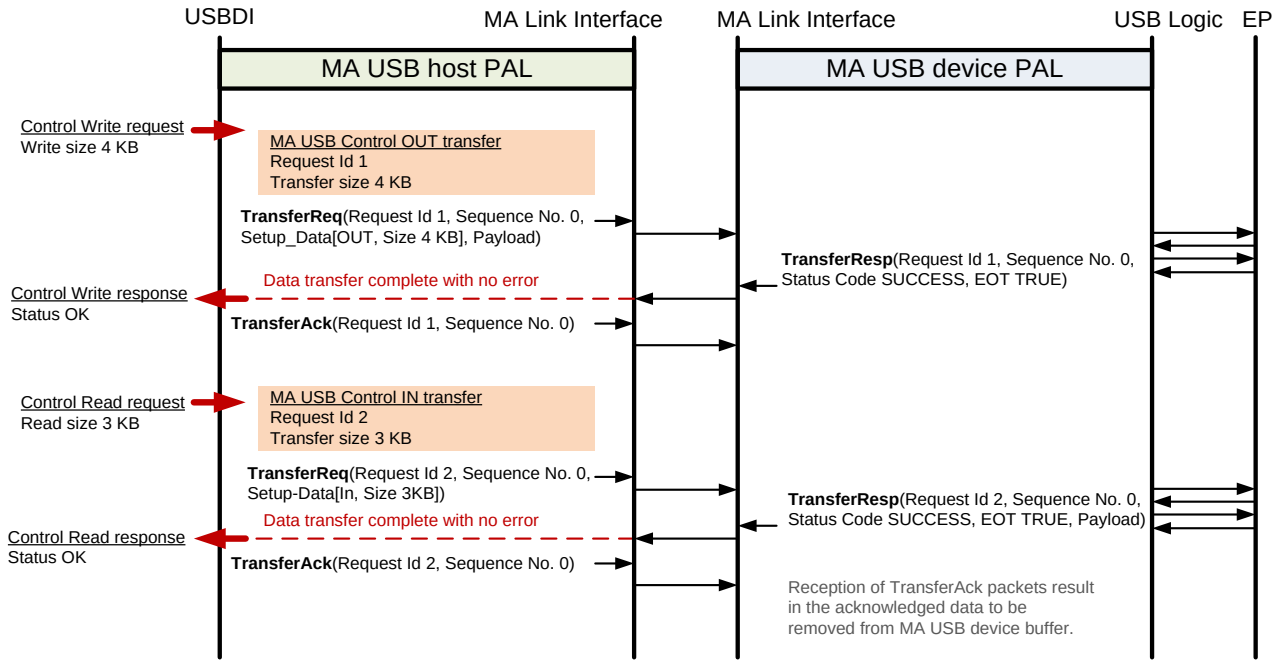


Figure 20—MA USB Control OUT and IN Transfers

5.7.1 Setup stage

An MA USB device shall have a buffer space of at least aMinControlTransferBufferSize bytes available for the first control transfer targeting the default control endpoint (endpoint 0) of each USB device behind the MA USB device. The first eight bytes of the buffer space is used for the setup data, and the remaining bytes are used for a control OUT transfer (data stage).

Each MA USB control transfer is initiated through a control TransferReq packet, which

- Carries a Request ID field with a new value.
- Carries a Sequence Number field set to zero. The sequence numbers at both the MA USB host and MA USB device re-initialize at the beginning of each control transfer request.

- (For control IN and OUT transfers) includes the 8 bytes of setup data as payload.
- (For control OUT transfers) includes as many bytes of the control write data that the target MA USB device can accept.

A device shall transmit a TransferResp packet to the start of a new control transfer. For control OUT transfers, the TransferResp packet shall include the Credit field for subsequent data TransferReq packets and initialize its expected sequence number to 1.

5.7.2 Data stage for control OUT transfers

The data stage of control OUT transfers follows the MA USB p-managed OUT (Section 5.5) or l-managed OUT (Section 5.6) data transfer model.

As described in the Setup stage, the control TransferReq packet for control OUT transfers carries as many bytes of the control write data that the target MA USB device can accept. If the size of the OUT data is larger than what the MA USB device can accept, then the host increments the sequence number counter and continues with TransferReq for OUT transfers following the transmission of the control TransferReq packet. On receipt of the Setup TransferReq packet, the target MA USB device observes from the setup data that the control transfer is a control write, and initializes its expected sequence number to 1, so that it is prepared to accept and handle the subsequent OUT TransferReq packets utilizing the bulk OUT transfer model.

NOTE — A control transfer with no data is treated as an OUT transfer with 0 bytes of control write data.

5.7.3 Data stage for control IN transfers

The data stage of control IN transfers follows the MA USB IN data transfer model (Section 5.4).

The MA USB host sets its expected sequence number to 0 after transmission of the control TransferReq packet. On receipt of the control TransferReq packet, the target MA USB device observes from the setup data that it is a control read, and initializes its sequence number to 0 and continues with the IN transfer with transmitting TransferResp packets with the IN data.

5.7.4 Status stage

The results of the USB control transfer status stage are delivered to the host encoded in the last TransferResp packet associated with the data stage of the control transfer. For a control IN transfer, the status stage result is included in the last TransferResp packet carrying payload. For a control OUT transfer, the status stage result is included in the TransferResp packet with the EoT field set to 1. Note that there is no explicit signaling required for status stage. If the control transfer completes without an error, then the SUCCESS (NO_ERROR) status in the TransferResp packet with EoT field set to 1 indicates the successful completion of the control transfer. Similarly, in case of a STALL response, the error is propagated to the application using similar mechanisms as with bulk transfers.

NOTE — Control endpoints shall not support functional STALL as defined in [USB 2.0] and [USB 3.1]. The STALL condition lasts until the receipt of the next control transfer.

5.8 Bulk transfers

Bulk transfers are used for time insensitive communication (usually large bursty data) between the host and the device. MA USB IN transfers (Section 5.4) and MA USB OUT transfers (Sections 5.5 and 5.6) are used for transport of USB IN and OUT bulk transfers, respectively.

5.9 Interrupt transfers

Interrupt transfers are generally non periodic transfers which require bounded latency. In a USB interrupt IN transfer the host regularly polls the device. In MA USB however, interrupt IN transfers use the MA USB IN transfer defined in Section 5.4, where the host issues a transfer request and waits for the response from the device. The behavior of the MA USB host and the MA USB device in case of a pending response is also described in Section 5.4.

Similarly, Interrupt OUT transfers use MA USB OUT transfers defined in Sections 5.5 and 5.6.

5.10 Isochronous transfers

MA USB isochronous transfers serve the same purpose as USB isochronous transfers: They support data streams that (1) require periodic delivery, and (2) are tolerant to occasional data losses. The USB isochronous operation is defined in [USB 2.0] and [USB 3.1] and is an inseparable companion to the MA USB isochronous operation defined in this section.

MA USB isochronous transfers do not follow the transfer models defined for other transfer types; they use a time-based delivery model that is defined in this section. The basic unit of transfer in MA USB isochronous operation is an *isochronous segment* (or segment for short), defined as the data generated or consumed by a target isochronous endpoint over a period of time equal to the Service Interval (SI) associated with that endpoint. The Service Interval associated with an FS isochronous endpoint is a power of two multiple of 1 millisecond, ranging from 1 millisecond to 32,768 milliseconds. The Service Interval associated with an HS or Enhanced SuperSpeed isochronous endpoint is a power of two multiple of 125 µs ranging from 125 µs to 4096 milliseconds. Each isochronous segment is normally (not always) associated with a presentation time, which points to the beginning of the Service Interval during which the segment was received from the target isochronous IN endpoint, or the Service Interval during which the segment is to be delivered to the target isochronous OUT endpoint.

NOTE — Some Operating System implementations refer to isochronous segments as isochronous packets.

The size of an isochronous transfer is specified by the application, and can span consecutive target Service Intervals. The size of each segment depends on the isochronous endpoint attributes and the amount of data available at the time of transfer. For example, an application-level read request for 4 segments, which targets an Enhanced SuperSpeed isochronous IN endpoint with a Service Interval of 4 milliseconds (*bInterval* = 6), maximum USB packet size of 1024 bytes (*wMaxPacketSize* =1024), up to three packet bursts in each Service Interval (*bmAttributes* = 2), and up to 16 USB packets in each burst (*bMaxBurst* =15) should secure enough buffer to receive up to 48 KB (1024 * 3 * 16 bytes) for each segment, or up to 192 KB in total. The target endpoint in this example may not generate the maximum of 48 KB over each target SI, resulting in isochronous segments that are smaller than 48 KB, and an MA USB isochronous transfer size of less than 192 KB.

NOTE — The *bmAttributes* and *bMaxBurst* parameters belong to the Enhanced SuperSpeed Endpoint Companion descriptor in this example.

The MA USB isochronous transfer model has been designed with the following observations,

- Isochronous segments can see a wide variation in size (zero bytes to 48 KB for an Enhanced SuperSpeed endpoint)
- Different links to different MA USB devices can have different MTU values, even if they employ the same technology (native Wi-Fi, native WiGig, IP, etc.)
- Acknowledging isochronous data packets at the MA USB protocol level is generally difficult or inefficient over half-duplex links such as native Wi-Fi and WiGig

- Isochronous data packets may be delivered out-of-order with some network technologies and link protocol choices
- Application-level data units are generally not aligned with USB isochronous packets, nor are they fully contained within an isochronous segment (the collective isochronous payload delivered over a Service Interval); a single application-level data unit may span multiple USB isochronous packets or even multiple isochronous segments

There is no protocol-level retransmission defined for MA USB isochronous data packets. The underlying link is expected to provide a reasonable reliability with the understanding that isochronous applications are tolerant to data loss. It is recommended to implement some level of reliability in the form of limited retransmissions at the network layer to support isochronous applications.

5.10.1 Packetization

MA USB isochronous transfers make use of two data packet subtypes: *Isochronous Transfer Request* (IsochTransferReq) packet, which is used to initiate an IN or OUT transfer, and also to carry isochronous payload for OUT transfers, and *Isochronous Transfer Response* (IsochTransferResp), which is used to provide transfer status updates and also to carry isochronous payload for IN transfers.

Each MA USB isochronous transfer may use multiple isochronous data packets to carry one or more isochronous segments. Some isochronous segments may experience fragmentation as a result of packetization. Conversely, small isochronous segments belonging to the same transfer may be carried inside the same isochronous data packet for transport efficiency. A complete or partial isochronous segment carried inside an isochronous data packet is referred to as an *isochronous data block*, or data block for short. To be able to identify and reassemble the fragments of an isochronous segment, and also to be able to pack multiple isochronous segments in each isochronous data packet, each isochronous data packet with payload also carries a set of isochronous headers associated with the data blocks in the packet. The format of isochronous data blocks and associated isochronous headers is defined in Section 5.10.1.1. For isochronous IN transfers, the MA USB host needs to specify the isochronous transfer size in the form of a series of maximum segment length values, where each value in the series indicates the size of the largest isochronous segment that the MA USB host can accept during one or more consecutive target Service Intervals. Maximum segment length values are defined in the form of a series of *isochronous read size* (IRS) blocks, where each block defines the maximum acceptable segment length for a number of consecutive target Service Intervals. The format of IRS block is defined in Section 5.10.1.2.

Figure 21 illustrates the isochronous data packet formats for isochronous IN and OUT transfers.

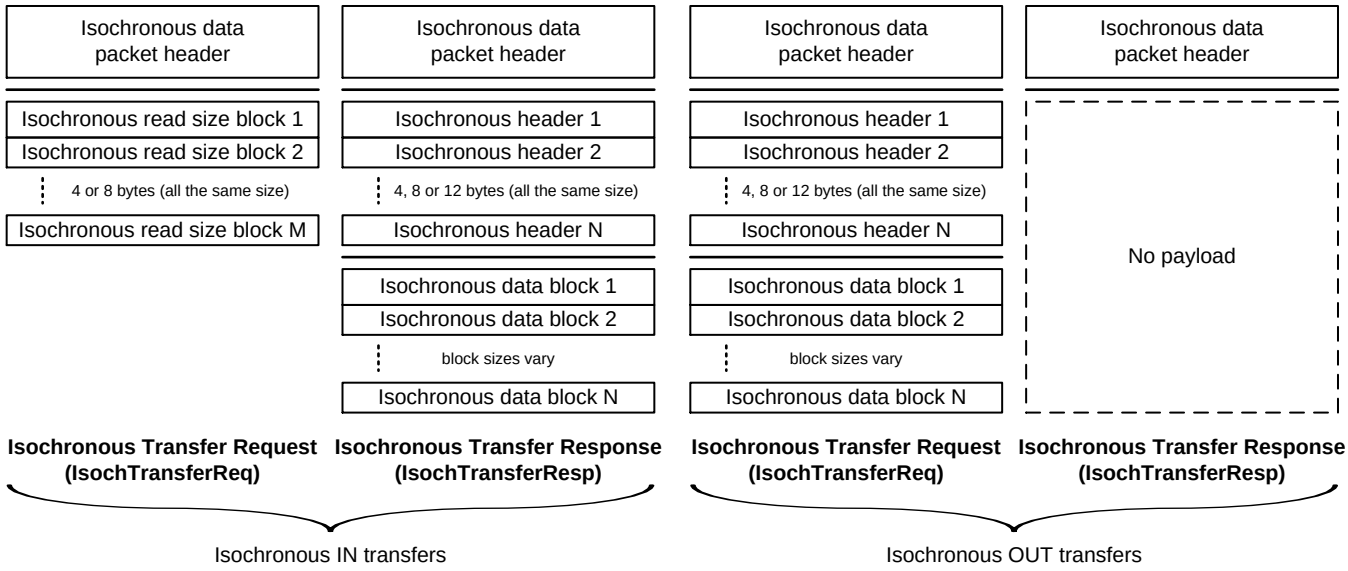


Figure 21—MA USB isochronous data packet formats

5.10.1.1 Isochronous data blocks

Figure 22 illustrates the format of isochronous data packets with isochronous payload. At the beginning of the packet are isochronous headers, placed one after another with the Segment Number field (defined below) strictly increasing. Isochronous data blocks follow the isochronous headers, and in the exact same order of the headers. If the MA USB device that sends or receives the isochronous data packet has required DWORD alignment for isochronous payload (by setting the Isochronous payload alignment field in the CapResp packet to 1), each isochronous data block is padded with 0 to 3 zero bytes to make the block length a multiple of 4 bytes; otherwise, isochronous data blocks are packed after each other with no padding.

NOTE — The format shown in Figure 22 applies only to isochronous data packets with isochronous payload, i.e., IsochTransferReq packets for isochronous OUT transfers and IsochTransferResp packets for isochronous IN transfers. See Sections 5.10.2 and 5.10.3 for details.

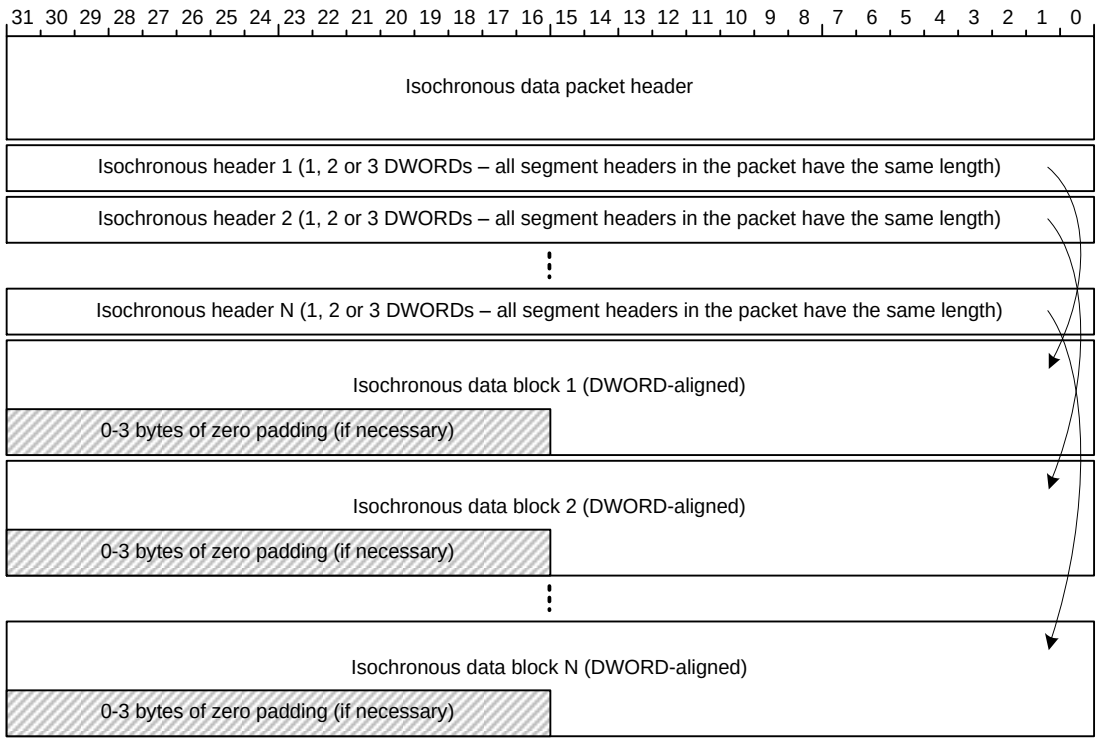


Figure 22—Format of isochronous data packets with isochronous payload

All isochronous headers in a packet have the same length, which can be 4, 8 or 12 bytes. The format (and length) of these isochronous headers is defined by the Isochronous Header Type field (Section 6.5.1.8) in the isochronous data packet header. Specifically, these isochronous headers take one of the three forms shown in Figure 23,

- The short format (Isochronous Header Format = 0) is 4 bytes in length, and does not support fragmentation; this format is best for packing multiple small isochronous segments inside a larger isochronous data packet.
- The standard format (Isochronous Header Format = 1) is 8 bytes in length, and supports fragmentation of segments as large as 64 KB; this format allows carrying a mix of complete and partial segments (subject to the rules defined in this section), and the segment size available through this format is adequate for FS, HS and Enhanced SuperSpeed USB devices.
- The long format (Isochronous Header Format = 2) is 12 bytes in length, and supports fragmentation of segments as large as 4 GB; this format is the most general and is defined to be able to support future revisions of the USB protocol.

NOTE — Each isochronous header is transmitted starting with Bit 0 of the lowest-order byte of the Block Length field and ending with Bit 4 of the S-Flags field for short format, or Bit 7 of the highest-order byte of the Fragment Offset field for long and extended formats.

NOTE — The common format for all isochronous headers in a packet is to enable efficient packet processing. Isochronous headers in different packets (including packets that target the same endpoint) may assume different formats.

NOTE — Implementations are free to choose the short or standard isochronous header format based on their packetization and design preferences. For example, an implementation that supports FS, HS and Enhanced SuperSpeed devices may choose to always use the standard format and carry a mix of complete and partial segments (subject to the rules defined in this section) in each packet. An alternative implementation may use a mix of short and standard formats depending on the segment lengths and need for fragmentation inside each packet.

NOTE — The long format for isochronous headers shall not be used unless the packet is carrying a fragment of an isochronous segment that is larger than 64 KB. This does not happen with FS, HS and Enhanced SuperSpeed isochronous endpoints.

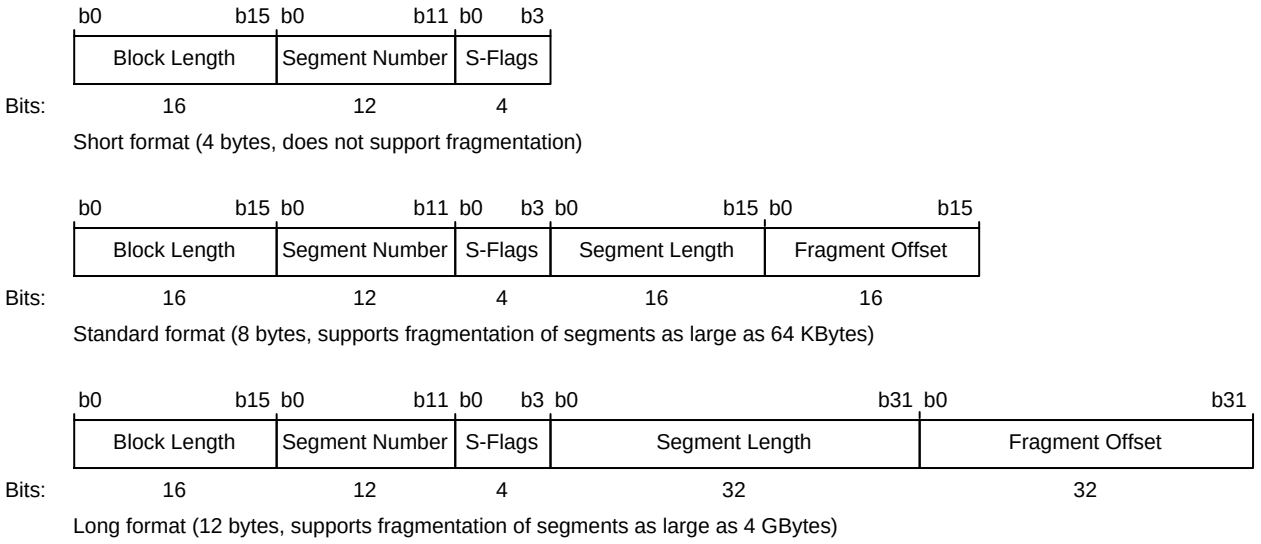


Figure 23—Isochronous header formats

The 16-bit Block Length field carries the length of the associated data block in bytes.

The 12-bit Segment Number identifies the segment the data block belongs to. Segment Number 0 refers to the first segment belonging to an isochronous transfer, Segment Number 1 refers to the second segment, and so on. Isochronous segments are sent in order, and segments with no data may be skipped, i.e., the Segment Number fields observed in any isochronous data packet are strictly increasing but are allowed to have gaps.

NOTE — MA USB architecture does not assume in-order delivery for isochronous data packets. Therefore, the Segment Number fields across sequentially received isochronous data packets belonging to the same MA USB transfer may not be strictly increasing.

NOTE — Any segment, including the first segment and the last segment of an isochronous transfer, may be skipped if it carries no data. Alternatively, an isochronous header with the Block Length field set to 0 and the Fragment bit (see below) set to 0 may be used to indicate a zero-payload (null) data block. No null data block shall be sent with the Fragment bit set to 1.

The 4-bit S-Flags field defines the contents of the data block, and also carries information related to the fragment transport and reassembly in case the associated data block is carrying a fragment. The field is shown in Figure 24, with the subfields defined in Table 1.

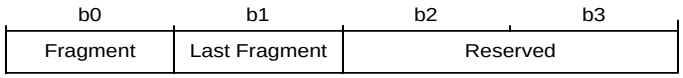


Figure 24—S-Flags field

Table 1—S-Flags subfields

Bit(s)	Description
b0	**Fragment.** Set to 1 if the data block associated with the isochronous header is carrying an incomplete isochronous segment (a fragment), and 0 otherwise. This subfield is set to zero in short isochronous headers with no fragmentation support.

b1	**Last Fragment.** Set to 1 if the carried fragment is the last fragment of an isochronous segment, and 0 otherwise. This subfield is defined only when the Fragment subfield is set to 1, and is reserved otherwise.
b3, b2	**Reserved.**

The long and extended isochronous header formats include two more fields:

- The Segment Length field indicates the length of the isochronous segment in bytes, i.e., the sum of the lengths of all data blocks carrying the segment. The field is 2 bytes long in the long format and 4 bytes long in the extended format.
- The Fragment Offset field indicates the byte offset of the carried fragment relative to the segment beginning. The field is 2 bytes long in the long format and 4 bytes long in the extended format.

NOTE — The short, long and extended isochronous header formats are defined to support very large isochronous segments as USB technology evolves while staying bit-efficient when carrying small segments.

NOTE — An isochronous header with long or extended format that corresponds to a full isochronous segment (i.e., no fragment) has the Segment Length and Block Length fields set to the segment size, and the Fragment Offset field set to 0.

The Last Fragment bit of the S-Flags field is set to 1 to indicate that the fragment carried by the isochronous data block is the last fragment of an isochronous segment.

NOTE — The Last Fragment bit has been defined to accelerate the logic to detect the last fragment of an isochronous segment; the Last Fragment bit is logically equivalent to the following Boolean function,

Segment Length = Fragment Offset + Block Length

NOTE — The Last Fragment bit (or the equivalent Boolean function above) assists the packet receiver in reassembly and delivery of fragmented segments. However, implementations must be prepared for the loss of the isochronous data packet that contains the last fragment of an isochronous segment. See Sections 5.10.2 and 5.10.3 for details.

No isochronous data packet shall include more than two fragments. The first fragment, if present, shall satisfy exactly one of the following conditions. The second fragment, if present, shall satisfy the third condition.

(1) The fragment is the last fragment of an isochronous segment (Last Fragment = 1), and is carried by the first data block in the isochronous data packet.
(2) The fragment is a fragment in the middle of an isochronous segment (Last Fragment = 0, Fragment Offset $\neq$ 0), and is carried by the only data block present in the isochronous data packet.
(3) The fragment is the first fragment of an isochronous segment (Fragment Offset field = 0), and is carried by the last data block in the isochronous data packet.

Isochronous transfers are expected to operate correctly with packet loss. Loss of an isochronous data packet translates to partial or complete loss of some isochronous segments. Segments experiencing partial loss may be discarded, or may be delivered to the USB application or the target endpoint up to anywhere before the lost bytes in the segment. No incomplete segments shall be delivered with lost bytes (gap) in the middle. Device and host implementations should balance their use of fragmentation (packetization efficiency) against increased likelihood of dropped or incomplete segments. For example, all packetization schemes in Figure 25 send the same number of isochronous segments using the same number of isochronous data packets, but they are different in the number of isochronous segments they subject to fragmentation and potential partial loss.

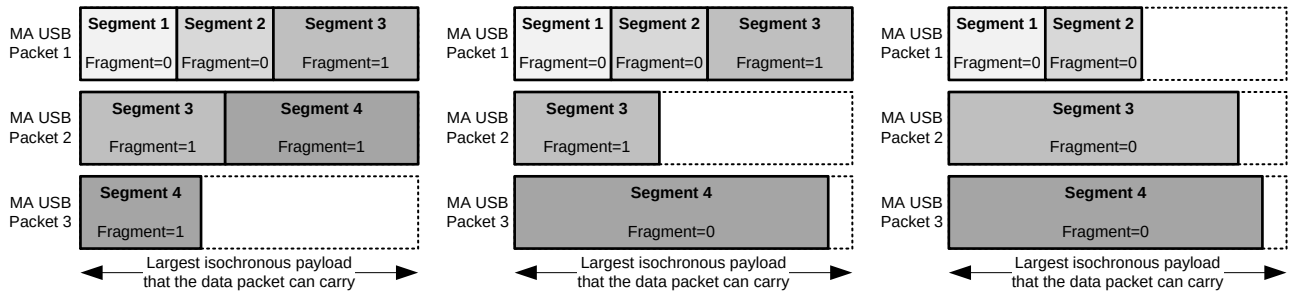


Figure 25—Examples of packetizing isochronous segments

Isochronous data blocks do not include a presentation time. The presentation time for the first isochronous data block in an isochronous data packet is carried in the Presentation Time field of the packet header (Section 6.5.1.9), and the presentation time for each subsequent data block in the packet is determined based on the Segment Number field in the isochronous header associated with the data block, and the Service Interval associated with the target isochronous endpoint.

All isochronous data packets that begin with the fragments of the same isochronous segment carry the same value for the Presentation Time field, except for request packets corresponding to an isochronous OUT transfer with ASAP delivery, where the Presentation Time field in each of the request packets is reserved. In this case, the actual presentation time of the first segment is decided by the target MA USB device, and the presentation time of each subsequent segment is implicitly determined based on the Segment Number field in each request packet and the Service Interval associated with the target isochronous endpoint. Regardless of the delivery mode of an isochronous transfer, the target MA USB device returns the presentation time of the first segment of the transfer in the Presentation Time field of each of the response packets corresponding to the transfer.

5.10.1.2 Isochronous read size blocks

Figure 26 illustrates the format of IsochTransferReq packets when used to initiate an isochronous IN transfer. Isochronous read size (IRS) blocks are carried immediately after the IsochTransferReq packet header.

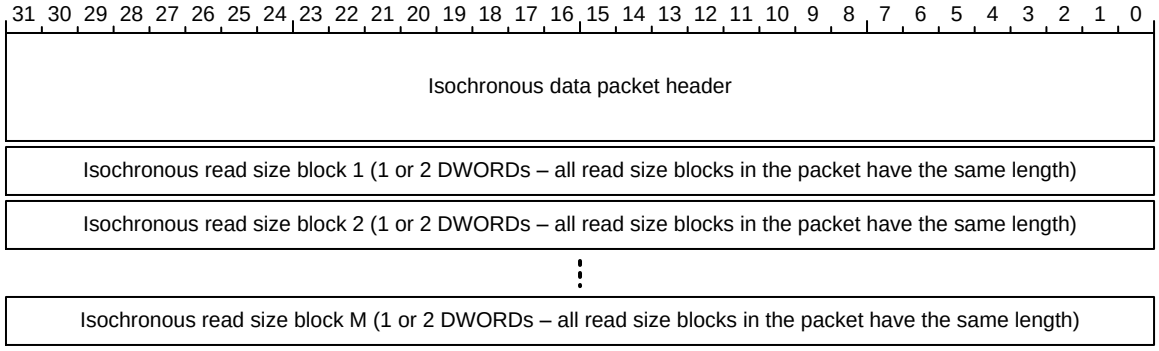


Figure 26—Format of Isochronous Transfer Request packets for IN transfers

All IRS blocks in a packet have the same length, which can be 4 or 8 bytes. The format (and length) of these blocks is defined by the Isochronous Header Type field (Section 6.5.1.8) in the IsochTransferReq packet header. Specifically, all IRS blocks in a packet take one of the two forms shown in Figure 27,

- The standard format (Isochronous Header Format = 0) is 4 bytes in length and allows maximum segment length values of up to 1 MB; the segment size available through this format is sufficient to support FS, HS and Enhanced SuperSpeed USB devices.
- The long format (Isochronous Header Format = 1) is 8 bytes in length, and allows maximum segment length values of up to 4 GB; this format is defined to be able to support future revisions of the USB protocol.

NOTE — Each IRS block is transmitted staring with Bit 0 of the Service Intervals field and ending with Bit 19 (standard format) or bit 31 (extended format) of the Maximum Segment Length field.

NOTE — The common format for all IRS blocks in a packet is to enable efficient packet processing. IRS blocks in different packets (including packets that target the same endpoint) may assume different formats.

NOTE — The long format for IRS blocks shall not be used unless the IsochTransferReq packet needs to specify at least one Maximum Segment Length value that exceeds 1 MB. This does not happen with FS, HS and Enhanced SuperSpeed isochronous endpoints.

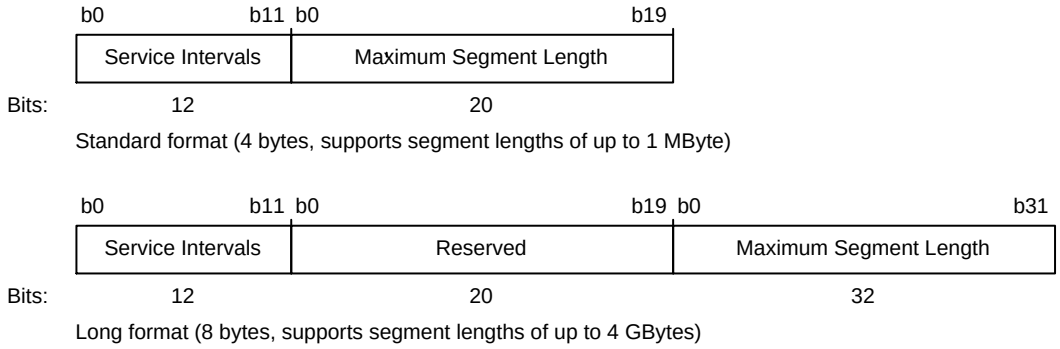


Figure 27—Isochronous read size block formats

The 12-bit Service Intervals field defines the number of consecutive target Service Intervals to which the IRS block applies.

The Maximum Segment Length field indicates the size of the largest isochronous segment (in bytes) that the MA USB host can receive during any of the target Service Intervals the IRS block applies to.

The IRS blocks in an IsochTransferReq packet apply to the target Service Intervals in strict order. For example, an IsochTransferReq packet that includes exactly two IRS blocks, with Service Intervals fields respectively set to S_1 and S_2 intervals, and Maximum Segment Length fields respectively set to M_1 and M_2, specifies a maximum segment length of M_1 for the first S_1 target service intervals, and a maximum segment length of M_2 for the next S_2 target service intervals. IsochTransferReq packets shall not include an IRS block with the Service Intervals field set to 0. The sum of the values in the Service Intervals fields of all IRS blocks in an IsochTransferReq packet shall be equal to the value of the Number of Segments field in the IsochTransferReq packet header (Section 6.5.5).

NOTE — Application-level isochronous IN requests that target a large number of Service Intervals with disparate maximum segment length values may result in a large number of IRS blocks. The MA USB host PAL may service such requests through multiple MA USB isochronous transfers to be able to meet the network MTU size for the generated IsochTransferReq packets.

5.10.2 Isochronous IN transfers

Isochronous IN transfers enable periodic data transfer from a target isochronous IN endpoint to the MA USB host, normally with preferential treatment from the network. The starting point for an isochronous IN transfer is an application-level read request targeting an isochronous IN endpoint over one or more successive Service Intervals. The first target Service Interval may begin at a precise time in the future

specified by the application-level request, or may start "as soon as possible", in which case its beginning time is decided by the target MA USB device. The MA USB host fulfills the application-level request through one or more MA USB isochronous IN transfers, which have the same general semantics as the application-level request; in particular, each MA USB isochronous IN transfer targets an isochronous IN endpoint over one or more Service Intervals, which may start at a specified time in the future, or at the convenience of the target MA USB device.

The MA USB host initiates an isochronous IN transfer by sending an Isochronous Transfer Request (IsochTransferReq) packet (Section 6.5.5) to the target MA USB device, indicating the target isochronous endpoint on the MA USB device, the Request ID assigned to the transfer request by the MA USB host, the number of isochronous segments that are being requested (one segment for each Service Interval), the beginning time of the first target Service Interval (in the Presentation Time field) or "as soon as possible" delivery option (using the ASAP subfield in the I-Flags field), and a set of isochronous read size (IRS) blocks as defined in Section 5.10.1.2.

The Request ID assigned to each transfer shall be unique within the scope of the target endpoint. The MA USB host may initiate other isochronous IN transfers targeting the same endpoint before an ongoing isochronous IN transfer has been completed. The rules for assigning Request IDs to successive isochronous transfers are the same as other MA USB IN transfers; in particular, (1) Request ID starts at zero for the first isochronous transfer after any configuration event intended to return the state of an endpoint flow to the initial state, (2) Request ID is incremented by 1 for each successive request, and (3) the MA USB host shall not have more than one pending isochronous transfer using the same Request ID and targeting the same endpoint. The rules for assigning Sequence Numbers to successive IsochTransferResp packets belonging to the same transfer (same request ID) are also the same as other MA USB IN transfers, except that the Sequence Number field is reset to zero for every new MA USB isochronous transfer. The Presentation Time field in successive IsochTransferReq packets that target the same endpoint and do not indicate ASAP delivery shall be strictly increasing.

In response to an IsochTransferReq packet, the target MA USB device programs its local resources to read from the target endpoint during the target Service Intervals. The read operation for each target Service Interval may happen anytime during that Service Interval. The target MA USB device shall packetize the isochronous payload as defined in Section 5.10.1, and shall transmit the payload back to the MA USB host in the strict order it was received from the target endpoint, using one or more Isochronous Transfer Response (IsochTransferResp) packets (Section 6.5.6). The target MA USB device may choose to deliver the isochronous segments that suffer a partial loss over the local connection up to any point before the first lost byte, including dropping them altogether. To reduce latency, the target MA USB device should packetize and transmit isochronous segments as they become available, avoiding excessive aggregation. Also, as discussed in Section 5.10.1, the target MA USB device should avoid excessive fragmentation in the interest of higher likelihood of delivering error-free segments.

Each IsochTransferResp packet belonging to an isochronous IN transfer carries the Request ID that was present in the IsochTransferReq packet that initiated the transfer. To enable the MA USB host to identify missing isochronous data packets, each IsochTransferResp packet carries a sequence number, which is set to zero in the first IsochTransferResp packet for a given transfer, and is incremented by one in every subsequent IsochTransferResp packet belonging to that transfer. In addition, each IsochTransferResp packet carries in its Presentation Time field the Global Time (MGT) pointing to the beginning of the Service Interval corresponding to the first isochronous segment or fragment carried in the packet. The EoT field shall be set to 0 in all IsochTransferResp packets belonging to an isochronous IN transfer, except in the last packet, where it shall be set to 1. The EPS and Status Code fields in each IsochTransferResp packet carry the status of the endpoint and the status code related to the transfer. The actual transfer size (i.e., the total size of the payload generated by the target isochronous endpoint over

target Service Intervals) may be smaller than the requested transfer size, including zero. The target MA USB device shall transmit an IsochTransferResp packet (with Number of Segments field set to 0, and EoT field set to 1) when the MA USB isochronous IN transfer has not generated any data.

NOTE — While isochronous data packets are allowed to be transmitted unreliably, it is recommended to exercise a high reliability at lower layers when sending any IsochTransferReq packet that initiates an isochronous IN transfer, the final IsochTransferResp packet belonging to an IN transfer, and also any IsochTransferResp packet that indicates an error.

Figure 28 illustrates the lifecycle of an MA USB isochronous IN transfer with specified delivery time: An IsochTransferReq packet initiates the transfer, targeting a remote isochronous IN endpoint over S successive Service Intervals, with the first Service Interval starting at the MA USB Global Time of F:M (Frame F, Microframe M). The target MA USB device, synchronized with the MA USB host, schedules one or more isochronous IN transfers on the local bus, and packetizes and transmits isochronous segments through one or more IsochTransferResp packets as segments become available.

NOTE — An MA USB isochronous IN transfer may also indicate "as soon as possible" (ASAP) delivery, in which case the start time of the isochronous transfer on the local bus is decided by the MA USB device.

NOTE — A single application-level isochronous read request may be served by multiple MA USB transfers.

The Presentation Time field in all IsochTransferResp packets belonging to an MA USB transfer carries the actual delivery time of the first isochronous segment (more precisely, the MGT value at the time the first byte of the first isochronous segment was released to the local bus), except when the target MA USB device experiences an error during the transfer (as indicated by the Status Code field in at least one of the returned IsochTransferResp packets), in which case, the Presentation Time field is reserved.

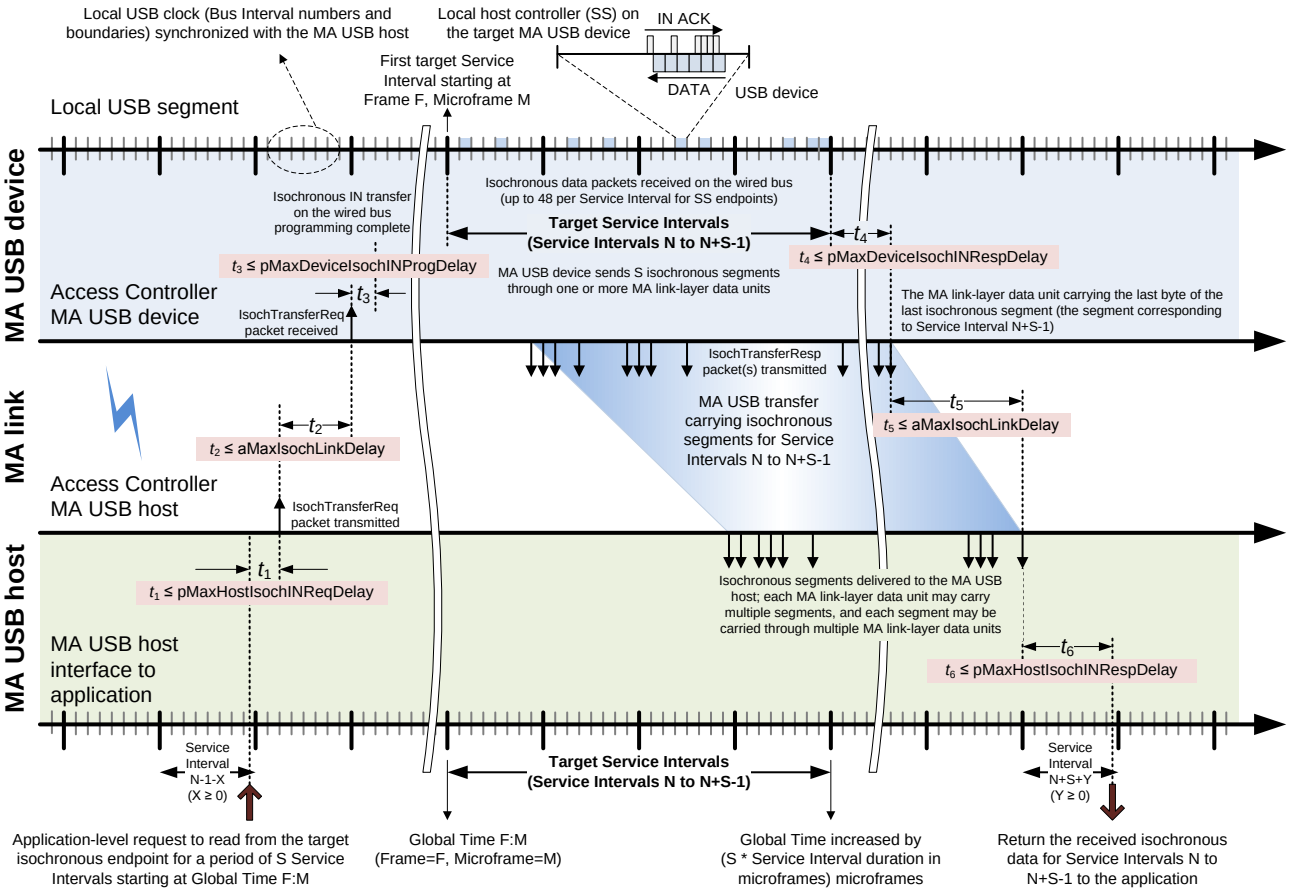


Figure 28—MA USB isochronous IN transfer

MA USB isochronous IN transfers generally follow a time-based delivery model, and their correct operation is guaranteed only if (1) certain system timings, related to the operation of the MA USB host, the network, and the target MA USB device, are bounded under normal operating conditions, and (2) the MA USB transfer follows certain rules formulated in terms of these upper bounds.

The most important timing parameter in the MA USB isochronous delivery model is the network delay: The delivery model requires isochronous data packets to be delivered under a bounded delay contract with the network. Part of the delay is intrinsic to the network, and the other depends on local decisions such as retransmission policy and packet lifetime. The overall delay for those isochronous data packets that are received (some packets may be lost) is expected not to exceed the protocol constant aMaxIsochLinkDelay, which is defined by this specification for each network technology.

Referring to Figure 28, the rules for isochronous IN transfers are defined in terms of upper bounds on the following timing parameters,

- t_1: The time from the moment an application-level isochronous read request is presented to the MA USB host PAL, to the moment the first bit of the first IsochTransferReq packet serving the read request is released to the network; this time is (MA USB host) implementation-dependent, although the application is expected to understand its impact when making isochronous read requests. The implementation guidelines in this section assume an upper bound of pMaxHostIsochINReqDelay for this timing parameter when discussing the impact.
- t_2: The time from the moment the first bit of an IsochTransferReq packet corresponding to an MA USB isochronous IN transfer is released to the network, to the moment that bit is received (if not dropped) at the target MA USB device network interface; this time is expected not to exceed the protocol constant aMaxIsochLinkDelay.
- t_3: The time from the moment the first bit of an IsochTransferReq packet corresponding to an MA USB isochronous IN transfer is received by a target MA USB device, to the moment the programming of the first isochronous IN transfer corresponding to the IsochTransferReq packet on the local bus is complete; this time is assumed to have an (MA USB device) implementation-dependent upper bound of pMaxDeviceIsochINProgDelay, which is made available to the MA USB host by the target MA USB device, together with the endpoint handle (Section 6.3.7).
- t_4: The time from the end of the last Service Interval an MA USB IN transfer targets, to the moment the first bit of the last IsochTransferResp packet corresponding to the transfer is released to the network; this time is assumed to have an (MA USB device) implementation-dependent upper bound of pMaxDeviceIsochINRespDelay, which is made available to the MA USB host by the target MA USB device, together with the endpoint handle (Section 6.3.7).
- t_5: The time from the moment the first bit of the last IsochTransferReq packet corresponding to an MA USB isochronous IN transfer request is released to the network, to the moment that bit is received (if not dropped) at the MA USB host network interface; this time is expected not to exceed the protocol constant aMaxIsochLinkDelay.
- t_6: The time from the moment the first bit of the last IsochTransferResp packet corresponding to an application-level isochronous read request is received at the MA USB host network interface, to the moment the MA USB host PAL returns to the application with the isochronous payload; this time is (MA USB host) implementation-dependent, although the application is expected to understand its impact when making isochronous read requests. The implementation guidelines in this section assume an upper bound of pMaxHostIsochINRespDelay for this timing parameter when discussing the impact.

Table 2 summarizes the timing parameters specific to the MA USB isochronous IN transfer model.

Table 2—Timing parameters specific to the MA USB isochronous IN transfer model

Timing parameter	Description	Source
pMaxHostIsochINReqDelay	Maximum time from the moment an application-level isochronous read request is presented to the MA USB host PAL, to the moment the first bit of the first IsochTransferReq packet corresponding to the request is released to the network	MA USB host implementation-dependent
pMaxDeviceIsochINProgDelay	Maximum time from the moment the first bit of an IsochTransferReq packet is received by the target MA USB device, to the moment the programming of the first isochronous IN transfer corresponding to the IsochTransferReq packet on the local bus is complete NOTE — The value of this parameter is inclusive of the minimum value of time required for posting of isochronous transfers before their scheduled execution time.	MA USB device implementation-dependent; made available to the MA USB host by the MA USB device, together with the endpoint handle (Section 7.3.2.2)
pMaxDeviceIsochINRespDelay	Maximum time from the end of the last Service Interval an MA USB isochronous IN transfer targets, to the moment the first bit of the last IsochTransferResp packet corresponding to the transfer is released to the network	MA USB device implementation-dependent; made available to the MA USB host by the MA USB device, together with the endpoint handle (Section 7.3.2.2)
pMaxHostIsochINRespDelay	Maximum time from the moment the first bit of the last IsochTransferResp packet corresponding to the last MA USB isochronous IN transfer serving an application-level isochronous read request is received at the MA USB host network interface, to the moment the MA USB host PAL returns to the application with the isochronous payload	MA USB host implementation-dependent

5.10.2.1 MA USB host requirements

MA USB host implementations should attempt to release IsochTransferReq packets to the network that either indicate as soon as possible (ASAP) delivery, or target one or more Service Intervals starting no earlier than aMaxIsochLinkDelay + pMaxDeviceIsochINProgDelay after the moment of release to the network.

NOTE — This requires an application-level isochronous read request to be presented to the MA USB host PAL no later than pMaxHostIsochINReqDelay + aMaxIsochLinkDelay + pMaxDeviceIsochINProgDelay before the earliest Service Interval it targets. MA USB host PAL implementations may reject or partially admit any application-level isochronous read request that does not meet this timing (e.g., an implementation may skip any target Service Interval that (the implementation assumes) will be received by the target MA USB device after the isochronous programming deadline); implementations may also choose to skew the MA USB Global Time (MGT) they present

to the application, to hide or alter the above timing requirement. It should be understood that skewing the Global Time presentation may not be suitable for applications or architectures that require synchronization across multiple USB systems, including multiple MA USB Service Sets.

The MA USB host PAL shall not release an IsochTransferReq to the network that indicates a start time (Presentation Time) more than aMaxFrameDistance USB frames before or after the MA USB Global Time (MGT) at the moment the first bit of the packet is released to the network.

The MA USB host PAL shall conclude a pending MA USB isochronous IN transfer (1) when it receives an IsochTransferResp packet that belongs to the transfer and has the EoT subfield set to 1, or (2) when it receives an IsochTransferResp packet that belongs to the transfer and indicates a non-recoverable error such as ISOCH_TIME_EXPIRED, or (3) at pMaxDeviceIsochINRespDelay + aMaxIsochLinkDelay time units after the end of the last Service Interval targeted by the transfer.

NOTE — The last condition guarantees the conclusion of the transfer under packet loss.

Once all pending MA USB isochronous IN transfers corresponding to an application-level isochronous read request are concluded, the MA USB host PAL shall return to the application with all isochronous segments received in entirety during the target Service Intervals. An exception is when the MA USB host PAL receives an IsochTransferResp packet that indicates an error, in which case the MA USB host PAL may return the appropriate USBDI error code to the application without further delay. The MA USB host PAL may also return partially received isochronous segments, up to any point before the first lost byte of each segment.

5.10.2.2 MA USB device requirements

In response to an IsochTransferReq packet that indicates any Presentation Time more than aMaxFrameDistance USB frames before or after the current MA USB Global Time (MGT), the target MA USB device shall send an IsochTransferResp packet with the same Request ID as the IsochTransferReq packet, and with the Status Code field set to ISOCH_TIME_INVALID. In response to an IsochTransferReq packet that indicates a valid Presentation Time, but (considering all the requested Service Intervals) the presentation time points to a time before the current MGT, or the presentation time falls within pMaxDeviceIsochINProgDelay of the current MGT, the target MA USB device shall send an IsochTransferResp packet with the same Request ID as the IsochTransferReq packet, and with the Status Code field set to ISOCH_TIME_EXPIRED.

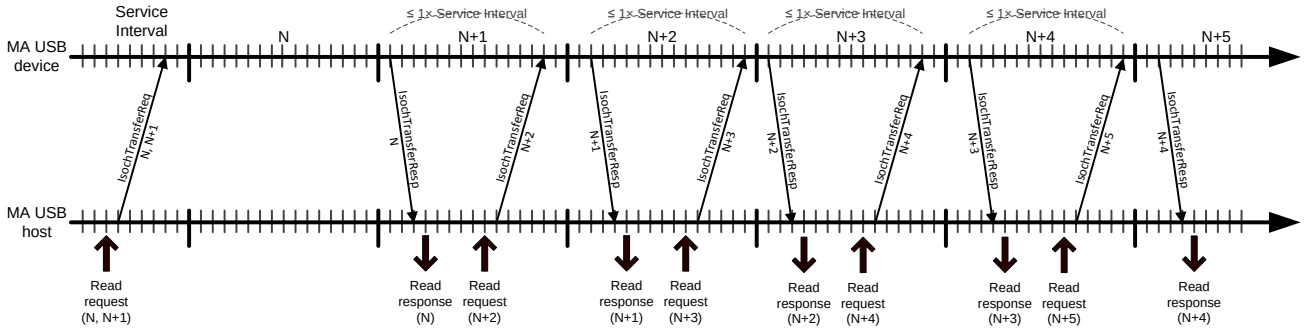
NOTE — MA USB device PAL implementations may reject, or partially admit any IsochTransferReq packet that does not meet the recommended timing (e.g., they may skip any target Service Interval that is too near to the time of programming).

5.10.2.3 Application design guidelines

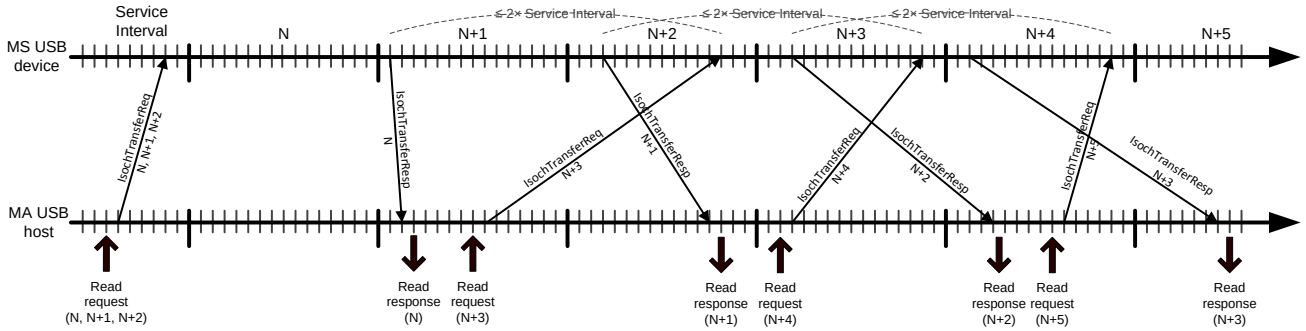
To ensure timely delivery of isochronous payloads, applications should issue each isochronous read request well ahead of the Service Interval the request is targeting. For continuous isochronous IN streaming, applications may need to issue multiple outstanding read requests. The number of outstanding requests and the associated required buffering increase with longer roundtrip times, where each roundtrip time captures a basic cycle of data movement.

NOTE — Roundtrip times can be measured from the application (MA USB host) point of view, or from the target MA USB device viewpoint, with no difference in measured values. From the application point of view, each roundtrip time represents the time between when a buffer unit is allocated to a new target Service Interval, and when a buffer unit allocated to another target Service Interval is made available again. From the target MA USB device viewpoint, each roundtrip time represents the time between when a buffer unit allocated to a target Service Interval is made available (MA USB device transmits the last IsochTransferResp packet corresponding to the target Service Interval), and when a buffer unit is allocated to a new target Service Interval (MA USB device receives an IsochTransferReq packet).

Figure 29 shows two examples of continuous IN streaming using double buffering and triple buffering. It is generally possible to achieve uninterrupted streaming using k+1 buffer units (with each buffer unit sized for one Service Interval) when roundtrip times do not exceed k Service Intervals. Implementations may choose a different level of buffering to adapt to roundtrip variations or to be able to target multiple Service Intervals through a single request.



(a) Continuous IN streaming using double buffering when the roundtrip time does not exceed one Service Interval



(b) Continuous IN streaming using triple buffering when roundtrip time does not exceed two Service Intervals

Figure 29—Continuous isochronous IN streaming using multiple levels of buffering

5.10.3 Isochronous OUT transfers

Isochronous OUT transfers enable periodic data transfer from the MA USB host to a target isochronous OUT endpoint, normally with preferential treatment from the network. The starting point for an isochronous OUT transfer is an application-level write request targeting an isochronous OUT endpoint over one or more successive Service Intervals. The first target Service Interval may begin at a precise time in the future specified by the application-level request, or may start "as soon as possible", in which case its beginning time is decided by the target MA USB device. The MA USB host fulfills the application-level request through one or more MA USB isochronous OUT transfers, which have the same general semantics as the application-level request; in particular, each MA USB isochronous OUT transfer targets an isochronous OUT endpoint over one or more Service Intervals, which may start at a specified time in the future, or at the convenience of the target MA USB device.

The MA USB host executes an isochronous OUT transfer by sending one or more Isochronous Transfer Request (IsochTransferReq) packets (Section 6.5.5) to the target MA USB device. Each IsochTransferReq packet indicates the target isochronous endpoint on the MA USB device, the Request ID assigned to the transfer request by the MA USB host, the number of isochronous segments that are being transmitted (one segment for each Service Interval), and the beginning time of the first target Service Interval (in the Presentation Time field) or "as soon as possible" (ASAP) delivery option (using the ASAP subfield in the I-Flags field). The Request ID assigned to the transfer shall be unique within the scope of the target endpoint. The MA USB host may initiate other isochronous OUT transfers

targeting the same endpoint before an ongoing isochronous OUT transfer is completed. The rules for assigning Request IDs to successive isochronous transfers are the same as other MA USB OUT transfers; in particular, (1) Request ID starts at zero for the first isochronous transfer after any configuration event intended to return the state of an endpoint flow to the initial state, (2) Request ID is incremented by 1 for each successive request, and (3) the MA USB host shall not have more than one pending isochronous transfer using the same Request ID and targeting the same endpoint. The rules for assigning Sequence Numbers to successive IsochTransferReq packets belonging to the same transfer (same request ID) are also the same as other MA USB OUT transfers, except that the Sequence Number field is reset to zero for every new MA USB isochronous transfer. The Presentation Time field in successive IsochTransferReq packets that target the same endpoint and do not indicate ASAP delivery shall be strictly increasing.

Each IsochTransferReq packet carries one or more partial or complete isochronous segments subject to the packetization rules defined in Section 5.10.1.

In response to each IsochTransferReq packet, the target MA USB device programs its local resources to write to the target endpoint during the target Service Intervals. The write operation for each target Service Interval may happen anytime during that Service Interval. The target MA USB device may choose to deliver the isochronous segments that suffer a partial loss over the local connection up to any point before the first lost byte, including dropping them altogether. To reduce latency, the MA USB host should packetize and transmit isochronous segments as they become available, avoiding excessive aggregation. Also, as discussed in Section 5.10.1, the MA USB host should avoid excessive fragmentation in the interest of higher likelihood of delivering error-free segments.

NOTE — MA USB architecture allows out-of-order delivery of isochronous data packets (transmission is always in-order). The target MA USB device may choose to discard out-of-order isochronous payload, or perform a re-order function if the presentation time of the received isochronous payload provides enough time for the function.

MEDIA DEPENDENT NOTE — While out-of-order delivery does not happen with direct operation over 802.11 radios, it is possible to experience out-of-order delivery when operating over IP (e.g., isochronous data packets carried inside UDP datagrams).

Figure 30 illustrates the lifecycle of an MA USB isochronous OUT transfer with specified delivery time: One or more IsochTransferReq packets carry the isochronous payload targeting a remote isochronous OUT endpoint over S successive Service Intervals, with the first Service Interval starting at the Global Time of F:M (Frame F, Microframe M). The target MA USB device, synchronized with the MA USB host, schedules one or more isochronous OUT transfers on the local bus, and packetizes and transmits the isochronous segments as they are received from the host. During delivery of isochronous segments on the local bus, or upon detecting errors, the target MA USB device transmits one or more IsochTransferResp packets to the MA USB host to indicate the status of the transfer. The Number of Segments field in each IsochTransferResp packet indicates the number of isochronous segments that have been delivered in entirety to the target endpoint.

NOTE — The frequency of IsochTransferResp packets is decided by the target MA USB device. For example, the target MA USB device may choose to transmit a single IsochTransferResp packet once all isochronous segments belonging to an MA USB transfer have been delivered to the target endpoint, or alternatively, transmit multiple IsochTransferResp packets to provide incremental updates to the MA USB host. For large isochronous transfers that include several segments, MA USB devices are recommended to provide incremental updates through multiple IsochTransferResp packets, to help the MA USB host better estimate the buffer available to the target endpoint.

NOTE — A single application-level isochronous write request may be served by multiple MA USB transfers.

NOTE — An MA USB isochronous OUT transfer may also indicate “as soon as possible” (ASAP) delivery, in which case the start time of the isochronous transfer on the local bus is decided by the MA USB device.

The Presentation Time field in the IsochTransferResp packet carries the actual delivery time of the first isochronous segment (more precisely, the MGT value at the time the first byte of the first isochronous segment belonging to the MA USB transfer was released to the local bus), except when the target MA USB device experiences an error during the transfer (as indicated by the Status Code field in the returned IsochTransferResp packet), in which case, the Presentation Time field is reserved.

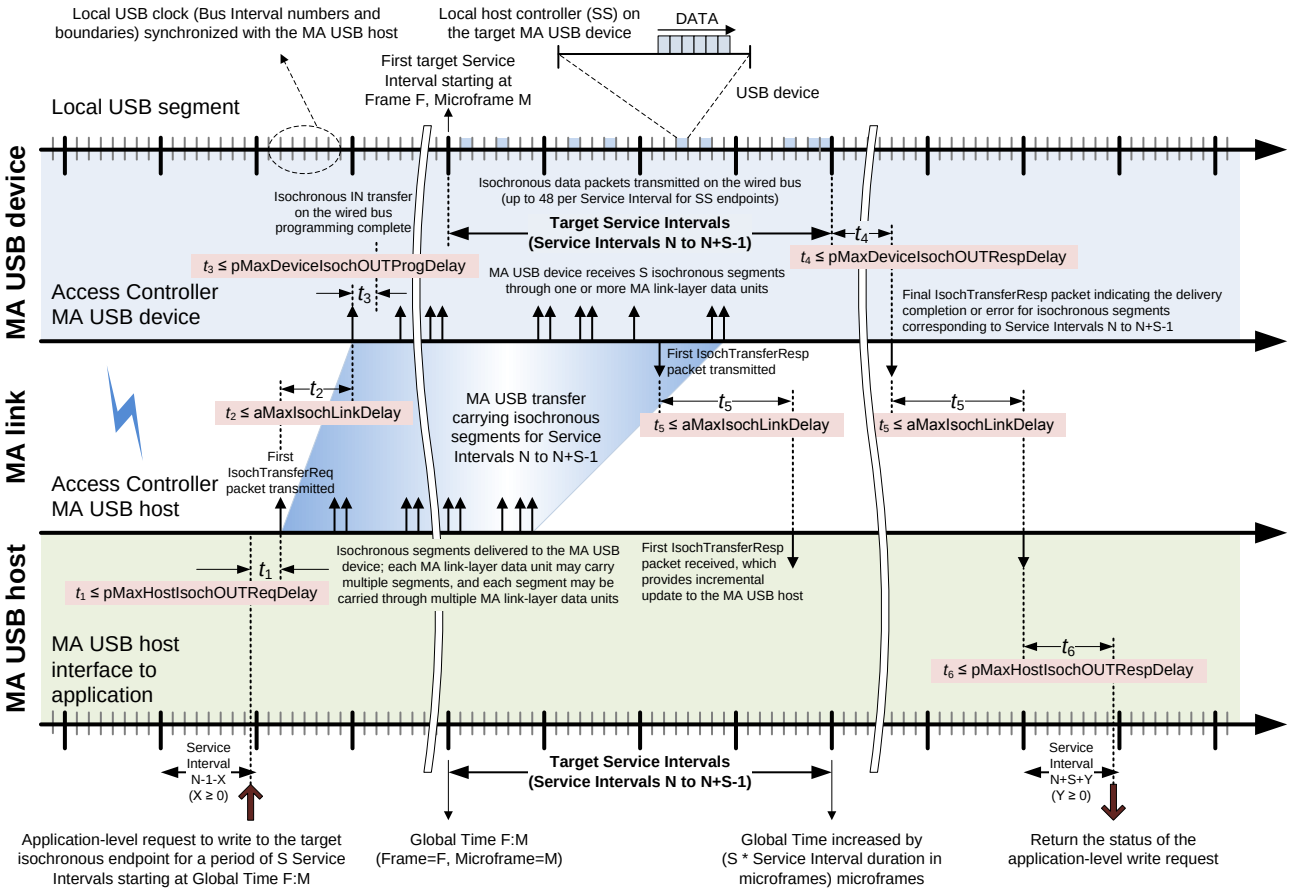


Figure 30—MA USB isochronous OUT transfer

Similar to MA USB isochronous IN transfers, MA USB isochronous OUT transfers generally follow a time-based delivery model and their correct operation is guaranteed only if (1) certain system timings, related to the operation of the MA USB host, the network, and the target MA USB device, are bounded under normal operating conditions, and (2) the MA USB transfer follows certain rules formulated in terms of these upper bounds. In particular, the upper bound on network-induced delay for isochronous data packets (the protocol constant aMaxIsochLinkDelay defined in Section 5.10.2) is again the key timing parameter for MA USB isochronous OUT transfers.

Referring to Figure 30, the rules for isochronous OUT transfers are defined in terms of upper bounds on the following timing parameters,

- t_1: The time from the moment an application-level isochronous write request is presented to the MA USB host PAL, to the moment the first bit of the first IsochTransferReq packet serving the write request is released to the network; this time is (MA USB host) implementation-dependent, although the application is expected to understand its impact when making isochronous write

requests. The implementation guidelines in this section assume an upper bound of pMaxHostIsochOUTReqDelay for this timing parameter when discussing the impact.

- t_2: The time from the moment the first bit of the first IsochTransferReq packet corresponding to an MA USB isochronous OUT transfer is released to the network, to the moment that bit is received (if not dropped) at the target MA USB device network interface; this time is expected not to exceed the protocol constant aMaxIsochLinkDelay.
- t_3: The time from the moment the first bit of the first IsochTransferReq packet corresponding to an MA USB isochronous OUT transfer is received by a target MA USB device, to the moment the programming of the first isochronous OUT transfer corresponding to the IsochTransferReq packet on the local bus is complete; this time is assumed to have an (MA USB device) implementation-dependent upper bound of pMaxDeviceIsochOUTProgDelay, which is made available to the MA USB host by the target MA USB device, together with the endpoint handle (Section 6.3.7).
- t_4: The time from the end of the last Service Interval an MA USB OUT transfer targets, to the moment the first bit of the IsochTransferResp packet corresponding to the transfer is released to the network; this time is assumed to have an (MA USB device) implementation-dependent upper bound of pMaxDeviceIsochOUTRespDelay, which is made available to the MA USB host by the target MA USB device, together with the endpoint handle (Section 6.3.7).
- t_5: The time from the moment the first bit of an IsochTransferReq packet corresponding to an MA USB isochronous OUT transfer request is released to the network, to the moment that bit is received (if not dropped) at the MA USB host network interface; this time is expected not to exceed the protocol constant aMaxIsochLinkDelay.
- t_6: The time from the moment the first bit of the IsochTransferResp packet corresponding to the last MA USB isochronous OUT transfer serving an application-level isochronous write request is received at the MA USB host network interface, to the moment the MA USB host PAL returns to the application the status of the isochronous write operation; this time is (MA USB host) implementation-dependent, although the application is expected to understand its impact when making isochronous read requests. The implementation guidelines in this section assume an upper bound of pMaxHostIsochOUTRespDelay for this timing parameter when discussing the impact.

Table 3 summarizes the timing parameters specific to the MA USB isochronous OUT transfer model.

Table 3—Timing parameters specific to the MA USB isochronous OUT transfer model

Timing parameter	Description	Source
pMaxHostIsochOUTReqDelay	Maximum time from the moment an application-level isochronous write request is presented to the MA USB host PAL, to the moment the first bit of the first IsochTransferReq packet corresponding to the request is released to the network	MA USB host implementation-dependent

pMaxDeviceIsochOUTProgDelay	Maximum time from the moment the first bit of the first IsochTransferReq packet corresponding to an isochronous OUT transfer is received by the target MA USB device, to the moment the programming of the first isochronous OUT transfer corresponding to the IsochTransferReq packet on the local bus is complete NOTE — The value of this parameter is inclusive of the minimum value of time required for posting of isochronous transfers before their scheduled execution time.	MA USB device implementation-dependent; made available to the MA USB host by the MA USB device, together with the endpoint handle (Section 7.3.2.2)
pMaxDeviceIsochOUTRespDelay	Maximum time from the end of the last Service Interval an MA USB isochronous OUT transfer targets, to the moment the first bit of the IsochTransferResp packet corresponding to the transfer is released to the network	MA USB device implementation-dependent; made available to the MA USB host by the MA USB device, together with the endpoint handle (Section 7.3.2.2)
pMaxHostIsochOUTRespDelay	Maximum time from the moment the first bit of the IsochTransferResp packet corresponding to the last MA USB isochronous OUT transfer serving an application-level isochronous write request is received at the MA USB host network interface, to the moment the MA USB host PAL returns to the application the status of the isochronous write operation	MA USB host implementation-dependent

5.10.3.1 MA USB host requirements

MA USB host implementations should attempt to release IsochTransferReq packets to the network that either indicate as soon as possible (ASAP) delivery, or target one or more Service Intervals starting no earlier than aMaxIsochLinkDelay + pMaxDeviceIsochOUTProgDelay after the moment of release to the network.

NOTE — This requires an application-level isochronous write request to be presented to the MA USB host PAL no later than pMaxHostIsochOUTReqDelay + aMaxIsochLinkDelay + pMaxDeviceIsochOUTProgDelay before the earliest Service Interval it targets. MA USB host PAL implementations may reject or partially admit any application-level isochronous write request that does not meet this timing (e.g., an implementation may skip any target Service Interval that (the implementation assumes) will be received by the target MA USB device after the isochronous programming deadline); implementations may also choose to skew the Global Time (MGT) they present to the application, to hide or alter the above timing requirement. It should be understood that skewing the MGT presentation may not be suitable for applications or architectures that require synchronization across multiple USB systems, including multiple Service Sets.

The MA USB host PAL shall not release an IsochTransferReq to the network that indicates a start time (Presentation Time) more than aMaxFrameDistance USB frames before or after the Global Time (MGT) at the moment the first bit of the packet is released to the network.

In addition, the MA USB host PAL should maintain a model of the device buffer available to the target isochronous OUT endpoint and follow a transmission schedule that does not result in buffer overflow.

The buffer model takes into account the transmitted isochronous payload, targeted Service Intervals, and the response packets received from the MA USB device.

NOTE — Insufficient buffer space allocated to an isochronous OUT endpoint on the MA USB device may result in scheduling complexity for the MA USB host, increased network activity and power consumption for the MA USB host and device, and possibly interruption of isochronous traffic stream. MA USB devices are recommended to allocate a buffer space to each isochronous OUT endpoint larger than the product of the expected data transmission rate to the endpoint multiplied by the endpoint access latency aMaxIsochLinkDelay + pMaxHostIsochOUTReqDelay + pMaxDeviceIsochOUTProgDelay. See Section 5.10.3.3 for more discussion.

For timed delivery (i.e., when isochronous presentation times are specified by the MA USB host), a model of the buffer available to a target isochronous OUT endpoint at time *t* can be defined as following

$$ABE(t) = TB - [IP(t) - \max(EIP(t), AIP(t))]$$

ABE(*t*): (Available Buffer Estimate) Host estimate of the available buffer at time *t*

TB: (Total Buffer) Total buffer allocated to the target isochronous OUT endpoint, indicated by the Buffer Size field in the MA USB EP descriptor (Section 6.3.7)

IP(*t*): (Isochronous Payload) Total isochronous payload transmitted before time *t*

DPT(*t*): (Delivered Presentation Time) Upper bound on presentation times that are expected to have been delivered to the target endpoint at time *t*; defined as the largest multiple of the endpoint service interval that does not exceed the current time (mathematically, $\lfloor MF/Interval \rfloor \times Interval$, where *MF* denotes the current absolute microframe number and *Interval* denotes the endpoint service interval in microframes); at any point in time *t*, all received isochronous payload with presentation time strictly earlier than *DPT*(*t*) is expected to have left the MA USB device buffer allocated to the endpoint

EIP(*t*): (Expired Isochronous Payload) Transmitted isochronous payload with presentation time earlier than *DPT*(t); this is the isochronous payload that is expected to have left the MA USB device buffer allocated to the target isochronous endpoint in the absence of any indication by the MA USB device

AIP(*t*): (Acknowledged Isochronous Payload) Isochronous payload transmitted before time *t* that is known to have left the MA USB device buffer through received IsochTransferResp packets

NOTE — For example, for an isochronous OUT endpoint with a service interval of 4 milliseconds (*Interval* = 32), at Global time *t* = (100, 5) (Frame number = 100, Microframe number = 5), the upper bound on delivered presentation times is given by $DPT(t) = \lfloor (100 \times 8 + 5)/32 \rfloor \times 32 = 800$ microframes = (100, 0) (Frame number = 100, Microframe number = 0), meaning no isochronous payload with presentation time strictly earlier than (100, 0) is expected to reside in the MA USB device buffer allocated to the endpoint.

NOTE — The accumulative format of the buffer model (defining buffer and payload quantities in terms of absolute numbers since the system start up) is mainly to illustrate the model; implementations likely use a differential format to track changes in the buffer size.

NOTE — Additional implementation details are needed to handle rollover in MGT and presentation times.

Figure 31 illustrates an example of how the MA USB host may estimate the buffer available to an isochronous OUT endpoint with timed delivery. Available buffer estimate can be updated as soon as an isochronous segment is guaranteed to have been delivered to the target endpoint, or an IsochTransferResp packet is received that confirms the delivery of an isochronous segment.

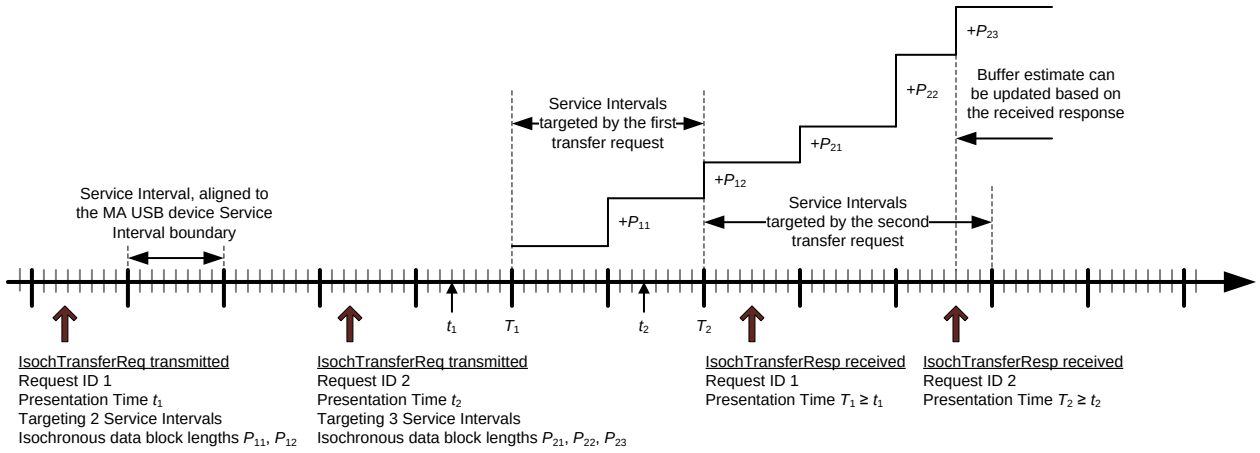


Figure 31—Example of buffer estimate for isochronous OUT timed delivery

For ASAP delivery (i.e., when isochronous presentation times are not specified), the above buffer model is simplified as

$$ABE(t) = TB - [IP(t) - AIP(t)]$$

Figure 32 illustrates an example of how the MA USB host may estimate the buffer available to an isochronous OUT endpoint with ASAP delivery. In this case, the available buffer estimate can be updated upon receiving an IsochTransferResp packet that confirms delivery of an isochronous segment. In particular, losing an IsochTransferResp packet can delay the buffer update.

NOTE — An IsochTransferResp packet that indicates delivery (successful or otherwise) of any segment belonging to an MA USB isochronous transfer also confirms the departure of all prior segments belonging to the same MA USB isochronous transfer, as well as all segments belonging to previous MA USB isochronous transfers (as identified by smaller Request ID values, subject to rollover considerations) from the MA USB device buffer.

NOTE — For large MA USB isochronous OUT transfers that include several segments, MA USB devices are recommended to provide incremental updates through multiple IsochTransferResp packets, to help the MA USB host better estimate the buffer available to the target endpoint.

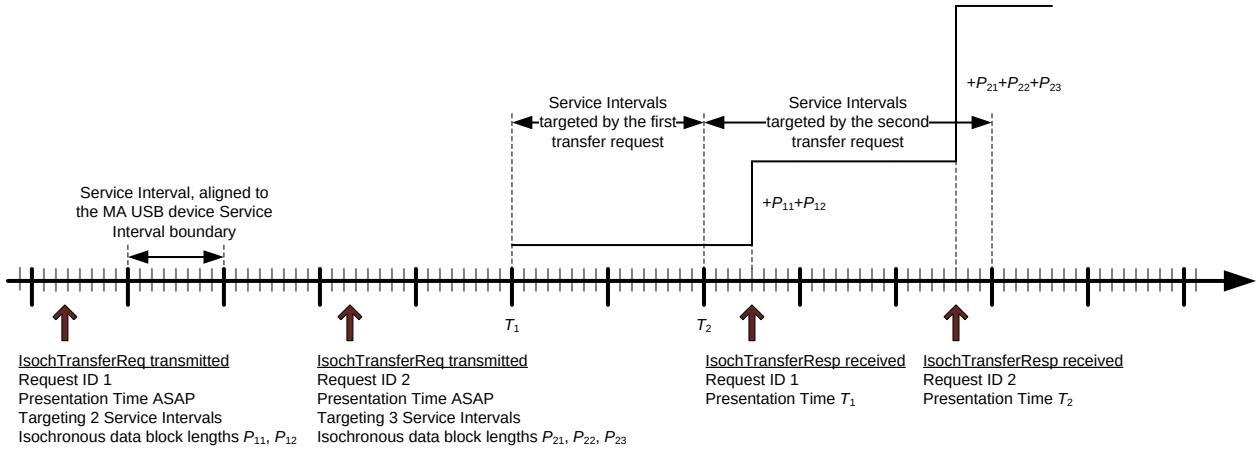


Figure 32—Example of buffer estimate for isochronous OUT ASAP delivery

The MA USB host PAL shall conclude a pending MA USB isochronous OUT transfer (1) when it receives an IsochTransferResp packet that belongs to the transfer and has the EoT subfield set to 1, or (2) when it receives an IsochTransferResp packet that belongs to the transfer and indicates a non-recoverable error such as ISOCH_TIME_EXPIRED, or (3) at pMaxDeviceIsochOUTRespDelay + aMaxIsochLinkDelay time units after the end of the last Service Interval targeted by the transfer.

NOTE — The last condition guarantees the conclusion of the transfer under packet loss.

Once all pending MA USB isochronous OUT transfers corresponding to an application-level isochronous write request are concluded, the MA USB host PAL shall return to the application the status of the isochronous write operation. An exception is when the MA USB host PAL receives an IsochTransferResp packet that indicates an error, in which case the MA USB host PAL may return the appropriate USBDI error to the application without further delay.

5.10.3.2 MA USB device requirements

In response to an IsochTransferReq packet that indicates any Presentation Time more than aMaxFrameDistance USB frames before or after the current Global Time (MGT), the target MA USB device shall send an IsochTransferResp packet with the same Request ID as the IsochTransferReq packet, and with the Status Code field set to ISOCH_TIME_INVALID. In response to an IsochTransferReq packet that indicates a valid Presentation Time, but (considering all the requested Service Intervals) the presentation time points to a time before the current MGT, or the presentation time falls within pMaxDeviceIsochOUTProgDelay of the current MGT, the target MA USB device shall send an IsochTransferResp packet with the same Request ID as the IsochTransferReq packet, and with the Status Code field set to ISOCH_TIME_EXPIRED.

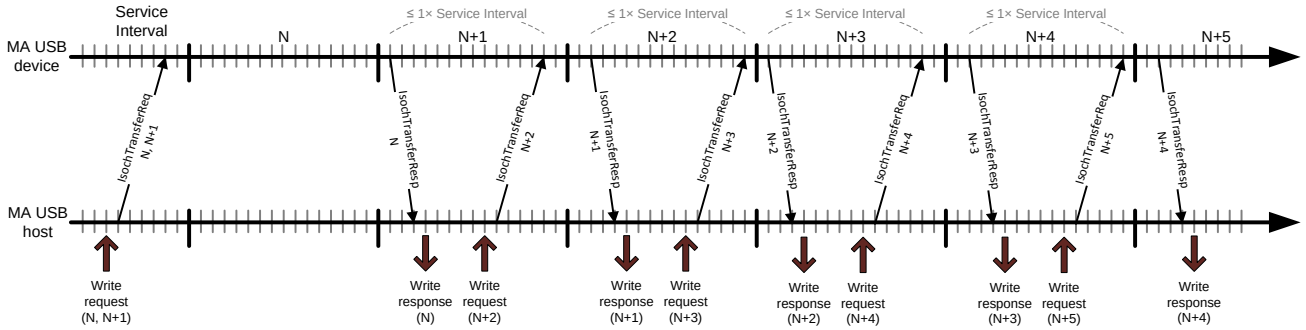
NOTE — MA USB device PAL implementations may reject, or partially admit any IsochTransferReq packet that does not meet the recommended timing (e.g., they may skip any target Service Interval that is too near to the time of programming).

5.10.3.3 Application design guidelines

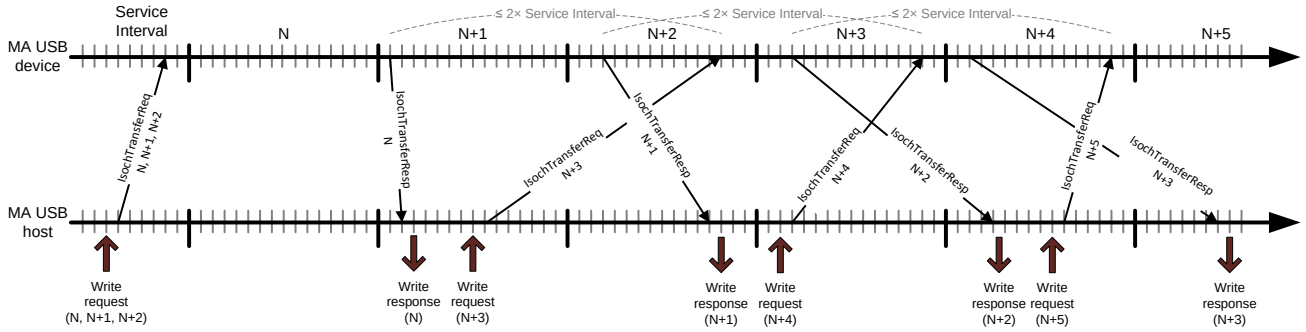
To ensure timely delivery of isochronous payload, applications should issue each isochronous write request well ahead of the Service Interval the request is targeting. For continuous isochronous OUT streaming, applications may need to issue multiple outstanding write requests. The number of outstanding requests and the associated required buffering increase with longer roundtrip times, where each roundtrip time captures a basic cycle of data movement.

NOTE — Roundtrip times can be measured from the application (MA USB host) point of view, or from the target MA USB device viewpoint, with no difference in measured values. From the application point of view, each roundtrip time represents the time between when a buffer unit is allocated to a new target Service Interval, and when a buffer unit allocated to another target Service Interval is made available again (MA USB host transmits the last IsochTransferReq packet corresponding to the target Service Interval). From the target MA USB device viewpoint, each roundtrip time represents the time between when a buffer unit allocated to a target Service Interval is made available (the last byte of isochronous payload for the target Service Interval is delivered to the USB endpoint), and when a buffer unit is allocated to a new target Service Interval (MA USB device receives the first IsochTransferReq packet corresponding to the target Service Interval).

Figure 33 shows two examples of continuous OUT streaming using double buffering and triple buffering. It is generally possible to achieve uninterrupted streaming using k+1 buffer units (with each buffer unit sized for one Service Interval) when roundtrip times do not exceed k Service Intervals. Implementations may choose a different level of buffering to adapt to roundtrip variations or to be able to target multiple Service Intervals through a single request.



(a) Continuous OUT streaming using double buffering when the roundtrip time does not exceed one Service Interval



(b) Continuous OUT streaming using triple buffering when roundtrip time does not exceed two Service Intervals

Figure 33—Continuous isochronous OUT streaming using multiple levels of buffering

5.11 Device notifications

Device notifications are a standard method for a USB device to communicate asynchronous device-level and bus-level event information to the host [USB 3.1]. Device notifications are always initiated by a USB device and the flow of data information in MA USB is always from an MA USB device to the MA USB host. The MA USB devices use the Device Notification Request (DevNotificationReq) packet (Section 6.3.50) to carry the device notifications to the MA USB host. The MA USB host shall respond to a DevNotificationReq packet with a Device Notification Response (DevNotificationResp) packet (Section 6.3.51) to inform the device whether the Device Notification Request was successfully received.

An MA USB device can send a Device Notification Request at any time.

5.12 Reliability

The MA USB protocol does not have specific reliability requirements for the physical layer. Reliable delivery is required at the link layer (802.11 mode) and network layer (IP mode) for all MA USB packets except isochronous data packets. The MA USB protocol defines mechanisms such as protocol-level retransmission to ensure data integrity in the event of network transients.

5.13 Efficiency

The MA USB protocol achieves transfer efficiency by sizing its data packets according to the MTU of the underlying link or network layer.

6 Protocol layer

6.1 Packet types

MA USB packets are broadly categorized into three types, each with one or more subtypes,

- **Management packets** transfer MA USB-specific information to manage entities such as endpoint handles. The information contained in these packets is strictly for MA USB PAL management; no USB payload of any type is carried in these packets. MA USB Capability Request (*CapReq*) and Endpoint Handle Request (*EPHandleReq*) are examples of a management packet.
- **Control packets** contain MA USB-specific information to control the flow of data packets carrying USB payload. Transfer Setup Request (TransferSetupReq) is an example of a control packet.
- **Data packets** transfer USB payload between MA USB host and MA USB devices. USB payload for all USB transfers, including control transfers, is carried in data packets.

Each packet carries a protocol header that contains the protocol version, the packet type and subtype, the MSS, and the MA USB device the packet is targeting. These are the key information a packet receiver needs to accept or reject an incoming packet, and to route the accepted packet to the correct protocol instance when multiple PAL instances (e.g., an MA USB host and an MA USB device PAL) are operating in parallel. MA USB packets can be as large as 64 KB; in practice, they are sized in accordance with the network MTU to minimize fragmentation.

MA USB does not define any header or payload protection mechanism such as Cyclic Redundancy Check (CRC) codes; all protocol content (header and payload) is assumed to be delivered error-free by the network. Similarly, MA USB does not define any data confidentiality mechanism; confidentiality, if required, is assumed to be provided by the network.

6.2 Packet formats

All packet headers have a length (in bytes) that is a multiple of 4, and are defined as a series of 4-byte data units known as double words (DWORDs). DWORDs are numbered starting from zero, and are displayed with the least significant bit (Bit 0) on the right and the most significant bit (Bit 31) on the left. Header bits are sent over the air starting with Bit 0 of DWORD 0 and ending with Bit 31 of the last DWORD in the header.

No header field is larger than a single DWORD in size, and no header field crosses DWORD boundaries. The location of each field or subfield in the header is defined pictorially or using the shorthand notation *DW*:*b*, where *DW* is the DWORD and *b* is the bit number for the least significant bit of the field or subfield. For example, a header field that occupies the lower 4 bits of the sixth byte in the header has an offset of 1:8.

Unless stated otherwise, a reserved field shall be sent with all the field bits set to zero.

The payload of the packet, if present, starts at bit number 0 of the first DWORD following the header fields. The payload bits are sent over the air starting with Bit 0 of the first DWORD following the packet header and ending with the most significant bit of the last DWORD of the payload.

6.2.1 Common header fields

All MA USB packets share the common header shown in Figure 34.

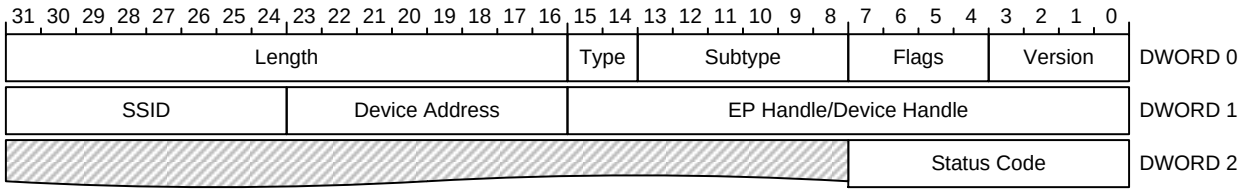


Figure 34— Common header for MA USB packets

6.2.1.1 Version

The 4-bit Version field (offset 0:0) identifies the MA USB protocol version. It is defined as 0000b for this revision of the specification. MA USB 1.0 devices and hosts shall set the Version field to 0000b at transmit.

6.2.1.2 Flags

The 4-bit Flags field (offset 0:4), shown in Figure 35, brings together the bit fields in Table 4.

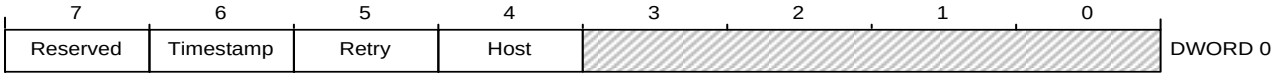


Figure 35—Flags field

Table 4—Flags subfields

Width (bits)	Offset (DW:bit)	Description
1	0:4	**Host**. Set to 1 if the packet is sent by the host, and set to 0 otherwise.
1	0:5	**Retry**. Set to 1 if the packet is a retry, and set to 0 otherwise.
1	0:6	**Timestamp**. Set to 1 if the packet header includes an MA USB Timestamp and Media Time/Transmission Delay fields, and 0 otherwise. NOTE — Certain packet subtypes are required to carry MA USB timestamps; all these packets shall set the Timestamp bit to 1. NOTE — For isochronous data packets, a valid Media Time/Transmission Delay field is separately marked by the MTD bit in the I-Flags (Section 6.5.1.8) field.
1	0:7	**Reserved**.

NOTE — The Host bit helps uniquely identify any MA USB packet (type/subtype) that may be transmitted by both an MA USB host and an MA USB device, and with potentially the same payload.

6.2.1.3 Type and Subtype

The 2-bit Type field (offset 0:14) and the 6-bit Subtype field (offset 0:8) together identify the packet variant, which in turn defines the expected packet structure and function. Valid Type and Subtype combinations are listed in Table 5.

Table 5—Type and Subtype values for MA USB packet variants

Type	Description	Subtype	Packet full name	Packet short name
00b	Management	000000b	MA USB Capability Request	CapReq
00b	Management	000001b	MA USB Capability Response	CapResp
00b	Management	000010b	USB Device Handle Request	USBDevHandleReq
00b	Management	000011b	USB Device Handle Response	USBDevHandleResp

Type	Description	Subtype	Packet full name	Packet short name
00b	Management	000100b	Endpoint Handle Request	EPHandleReq
00b	Management	000101b	Endpoint Handle Response	EPHandleResp
00b	Management	000110b	Endpoint Activate Request	EPActivateReq
00b	Management	000111b	Endpoint Activate Response	EPActivateResp
00b	Management	001000b	Endpoint Inactivate Request	EPInactivateReq
00b	Management	001001b	Endpoint Inactivate Response	EPInactivateResp
00b	Management	001010b	Endpoint Reset Request	EPResetReq
00b	Management	001011b	Endpoint Reset Response	EPResetResp
00b	Management	001100b	Clear Transfers Request	ClearTransfersReq
00b	Management	001101b	Clear Transfers Response	ClearTransfersResp
00b	Management	001110b	Endpoint Handle Delete Request	EPHandleDeleteReq
00b	Management	001111b	Endpoint Handle Delete Response	EPHandleDeleteResp
00b	Management	010000b	MA USB Device Reset Request	DevResetReq
00b	Management	010001b	MA USB Device Reset Response	DevResetResp
00b	Management	010010b	Modify EP0 Request	ModifyEP0Req
00b	Management	010011b	Modify EP0 Response	ModifyEP0Resp
00b	Management	010100b	Set USB Device Address Request	SetUSBDevAddrReq
00b	Management	010101b	Set USB Device Address Response	SetUSBDevAddrResp
00b	Management	010110b	Update Device Request	UpdateDevReq
00b	Management	010111b	Update Device Response	UpdateDevResp
00b	Management	011000b	USB Device Disconnect Request	USBDevDisconnectReq
00b	Management	011001b	USB Device Disconnect Response	USBDevDisconnectResp
00b	Management	011010b	USB Suspend Request	USBSuspendReq
00b	Management	011011b	USB Suspend Response	USBSuspendResp
00b	Management	011100b	USB Resume Request	USBResumeReq
00b	Management	011101b	USB Resume Response	USBResumeResp
00b	Management	011110b	Remote Wake Request	RemoteWakeReq
00b	Management	011111b	Remote Wake Response	RemoteWakeResp
00b	Management	100000b	Ping Request	PingReq
00b	Management	100001b	Ping Response	PingResp
00b	Management	100010b	MA USB Device Disconnect Request	DevDisconnectReq
00b	Management	100011b	MA USB Device Disconnect Response	DevDisconnectResp

Type	Description	Subtype	Packet full name	Packet short name
00b	Management	100100b	MA USB Device Initiated Disconnect Request	DevInitDisconnectReq
00b	Management	100101b	MA USB Device Initiated Disconnect Response	DevInitDisconnectResp
00b	Management	100110b	Synchronization Request	SynchReq
00b	Management	100111b	Synchronization Response	SynchResp
00b	Management	101000b	Cancel Transfer Request	CancelTransferReq
00b	Management	101001b	Cancel Transfer Response	CancelTransferResp
00b	Management	101010b	Endpoint Open Stream Request	EPOpenStreamReq
00b	Management	101011b	Endpoint Open Stream Response	EPOpenStreamResp
00b	Management	101100b	Endpoint Close Stream Request	EPCloseStreamReq
00b	Management	101101b	Endpoint Close Stream Response	EPCloseStreamResp
00b	Management	101110b	USB Device Reset Request	USBDevResetReq
00b	Management	101111b	USB Device Reset Response	USBDevResetResp
00b	Management	110000b	Device Notification Request	DevNotificationReq
00b	Management	110001b	Device Notification Response	DevNotificationResp
00b	Management	110010b	Endpoint Set Keep-Alive Request	EPSetKeepAliveReq
00b	Management	110011b	Endpoint Set Keep-Alive Response	EPSetKeepAliveResp
00b	Management	110100b	Get Port Bandwidth Request	GetPortBWReq
00b	Management	110101b	Get Port Bandwidth Response	GetPortBWResp
00b	Management	110110b	Sleep Request	SleepReq
00b	Management	110111b	Sleep Response	SleepResp
00b	Management	111000b	Wake Request	WakeReq
00b	Management	111001b	Wake Response	WakeResp
00b	Management	111110b	Vendor Specific Request	VendorSpecificReq
00b	Management	111111b	Vendor Specific Response	VendorSpecificResp
01b	Control	000000b	Transfer Setup Request	TransferSetupReq
01b	Control	000001b	Transfer Setup Response	TransferSetupResp
01b	Control	000010b	Transfer Tear Down Confirmation	TransferTearDownConf
10b	Data	000000b	Transfer Request	TransferReq
10b	Data	000001b	Transfer Response	TransferResp
10b	Data	000010b	Transfer Acknowledgement	TransferAck
10b	Data	000011b	Isochronous Transfer Request	IsochTransferReq
10b	Data	000100b	Isochronous Transfer Response	IsochTransferResp

6.2.1.4 Length

The 16-bit Length field (offset 0:16) carries the length of the MA USB packet, including the packet header, in bytes.

6.2.1.5 EP Handle/Device Handle

This 16-bit field (offset 1:0) carries a handle for a USB device or a USB endpoint. When pointing to a USB device (in MA USB packets of type management), the field is referred to as Device Handle, and is unstructured. When pointing to a USB endpoint (EP) (in MA USB packets of type control and data), the field is referred to as EP Handle, and is structured as shown in Figure 36.

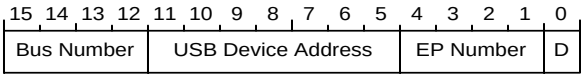


Figure 36—EP Handle field

The Bus Number subfield carries the USB bus number for the EP assigned by the MA USB device PAL. For virtual USB devices the Bus Number subfield is set to the reserved value of 15. For physical USB devices it is set to the value of the USB bus number allocated by the MA USB device PAL. This subfield is used to ensure uniqueness of the EP handle value when there are multiple USB devices with the same USB address present.

> NOTE — A USB host controller implementation can choose to treat each downstream port as a different bus instance (Enhanced SuperSpeed and USB 2.0 speeds) while assigning addresses to the devices connected to the downstream ports (in the case of a host controller with 2 Enhanced SuperSpeed ports and 2 USB 2.0 ports there can be up to 4 separate bus instances). Additionally, a host controller implementation may choose to assign USB addresses independently on each bus instance (starting at device address 1, for example).

The USB Device Address subfield carries the USB address of the USB device to which the EP belongs. The USB address is allocated by the MA USB device. For virtual USB devices the USB Device Address subfield is set by the MA USB device PAL. For physical USB devices it is set to the USB address of the device. Prior to allocation of the USB address to the device, this field is set to the default value of 0.

The EP Number subfield carries the EP number as defined in [USB 2.0]. Prior to allocating an address, the only valid EP Number is zero.

The Direction (D) subfield carries the direction of the EP as defined in [USB 2.0].

6.2.1.6 Device Address

The 8-bit Device Address field (offset 1:16) carries the address assigned by an MA USB host to an MA USB device, or a value of 0xFF to indicate any MA USB device within the MSS.

6.2.1.7 SSID

The 8-bit SSID field (offset 1:24) identifies the MA USB Service Set (MSS) to which the target MA USB device belongs.

6.2.1.8 Status Code

The 8-bit Status Code field (offset 2:0) is used to communicate a status code, e.g., the success or failure of a requested operation. If an operation is successful, or involves no error, the Status Code is set to 0 (SUCCESS, or NO_ERROR). Otherwise, depending on the operation, one of the values in Table 6 is returned in a response packet. Management packets with Status Code field set to values other than 0, shall still carry all the fields defined for the packet subtype.

1

Table 6—Status Code values

Value	Description
0	**SUCCESS (NO_ERROR)**. Used when no error condition is being reported.
1-127	Reserved values
128	**UNSUCCESSFUL.** The operation did not succeed due an unidentified error.
129	**INVALID_MA_USB_SESSION_STATE.** The MA USB session is in an invalid state for the requested operation.
130	**INVALID_DEVICE_HANDLE.** Provided device handle is invalid.
131	**INVALID_EP_HANDLE**. Provided EP handle is invalid.
132	**INVALID_EP_HANDLE_STATE.** Provided EP handle is an invalid state for the requested operation.
133	**INVALID_REQUEST**. The received Transfer Request was invalid or did not match the endpoint capabilities.
134	**MISSING_SEQUENCE_NUMBER**. A sequence number gap was detected.
135	**TRANSFER_PENDING**. Used in response to a duplicate IN Transfer Request to indicate that the Transfer Request has been received but the data is not being received from the target USB endpoint.
136	**TRANSFER_EP_STALL.** Transfer request ended with the USB endpoint returning STALL handshake.
137	**TRANSFER_SIZE_ERROR.** The TransferResp packet includes an unexpected number of bytes in its payload.
138	**TRANSFER_DATA_BUFFER_ERROR.** Overrun of incoming data or under-run of outgoing data has occurred.
139	**TRANSFER_BABBLE_DETECTED.** Babble error returned by the WSB device.
140	**TRANSFER_TRANSACTION_ERROR.** Transaction error returned by the target USB device.
141	**TRANSFER_SHORT_TRANSFER.** Transfer request ended with the short packet.
142	**TRANSFER_CANCELLED.** The received Transfer Request corresponds to a cancelled transfer.
143	**INSUFFICIENT_RESOURCES.** There are not enough resources on the MA USB device to perform the operation.
144	**NOT_SUFFICIENT_BANDWIDTH.** There is not enough bandwidth to support the endpoint(s).
145	**INTERNAL_ERROR.** MA USB device encountered an internal error.
146	**DATA_OVERRUN.** The isochronous transfer exceeded the bandwidth allocated to the endpoint.
147	**DEVICE_NOT_ACCESSED.** The USB device is not accessible.
148	**BUFFER_OVERRUN.** The USB device attempted transfer of more data than the scheduled amount per service interval for an isochronous IN transfer.
149	**BUSY.** An attempt was made to close an endpoint with outstanding transfers.
150	**DROPPED_PACKET.** MA USB device dropped a TransferReq packet.
151	**ISOCH_TIME_EXPIRED.** Isochronous payload was delivered later than its intended presentation time.
152	**ISOCH_TIME_INVALID.** The presentation time is invalid.

153	**NO_USB_PING_RESPONSE.** Completion of data transfer to an Enhanced SuperSpeed isochronous endpoint within the service interval is not possible because the USB device has not responded to a PING with a PING_RESPONSE in the required time [USB 3.1].
154	**NOT_SUPPORTED.** The value of the Subtype field is not recognized or the requested operation is not supported.
155	**REQUEST_DENIED.** A protocol operation request was denied.
156	**MISSING_REQUEST_ID.** A Request ID gap was detected.
157-255	Reserved values

6.3 Management packets

MA USB management packets share the common management header shown in Figure 37. All management header fields except for the Dialog Token field are defined in Section 6.2.

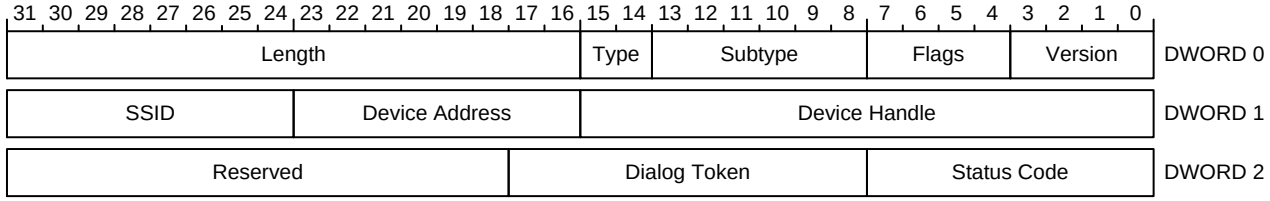


Figure 37—Common header for MA USB management packets

6.3.1 Common header fields

6.3.1.1 Dialog Token

The 10-bit Dialog Token field (offset 2:8) is used to match request and response management packets between an MA USB host and a target MA USB device PAL. The field is set to 1 at session initialization or upon any configuration event that returns the state of the MA USB PAL to the initial state. The Dialog Token value is incremented by 1 after transmitting each new (i.e., not retried) management packet carrying a request, with wraparound to 1 after reaching the maximum value of aMaxDialogToken. To avoid ambiguity in tracking the request management packets waiting for response, the MA USB PAL shall have no more than $\lfloor aMaxDialogToken/2 \rfloor$ (half the size of the Dialog Token space) outstanding management requests. Also the total management requests targeted for an MA USB PAL shall not exceed the number returned by the target MA USB PAL in the Number of Outstanding Management Requests field in the CapReq or CapResp packets. A management packet carrying an out-of-order Dialog Token value shall be ignored by the recipient.

NOTE — The value of the Dialog Token field is allocated by the originator of the request. Hence, there may be two active requests at the MA USB device or the MA USB host with the same Dialog Token value, one originated from the MA USB device and one from the MA USB host.

The Dialog Token field is set to 0 for management packets with broadcast Device Address, and other management packets that do not require a response correlated with the request.

6.3.2 MA USB Capability Request (CapReq)

The MA USB Capability Request (CapReq) packet is transmitted by the MA USB host to a target MA USB device to inquire about the MA USB device's capabilities. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 0 (CapReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The CapReq packet carries the fields listed in Table 7 after the management header.

Table 7—MA USB Capability Request fields

Width (bits)	Offset (DW:bit)	Description
12	3:0	**Number of Outstanding Management Requests.** Indicates the maximum number of device initiated outstanding management requests that MA USB host can track. NOTE—The MA USB host is recommended to be able to track at least 127 outstanding requests per each MA USB device it supports.
20	3:12	**Reserved.**

In addition, the CapReq packet may carry one or multiple MA Host Capability descriptors. The format of the MA Host Capability descriptors is defined in Table 8. The MA Host Capability descriptors are byte aligned.

Table 8—Format of MA Host Capability descriptors

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes.
8	N:n+8	**MA Host Capability Type.** Indicates the type of the descriptor. Descriptor types are listed in Table 9.
Variable	N:n+16	**Descriptor specific format.**

The valid MA Host Capability Types to be carried in the CapReq packet are listed in Table 9.

Table 9—MA Host Capability Type values

MA Host Capability Type	Value	Description
Synchronization Capabilities	3	Indicates the synchronization related capabilities of the MA USB host. Shall be present if the MA USB host has access to Media Time.
Link Sleep Capability	5	Indicates the capability of the MA USB host related to the session state transition to Session Inactive. Shall be present if the MA USB host supports transition of the session state to Session Inactive without the integrated USB device being suspended.

6.3.2.1 Synchronization Capabilities descriptor

The Synchronization Capabilities descriptor reports the synchronization related capabilities of the MA USB host and shall be present if the MA USB host has access to the Media Time. Absence of the Synchronization Capabilities descriptor is equivalent of Media Time Available field set to 0. Table 10 illustrates the format of the Synchronization Capabilities descriptor.

Table 10—Synchronization Capabilities descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.

8	N:n+8	**MA Host Capability Type.** Indicates the type of the descriptor. Set to the value of Synchronization Capabilities in Table 9.
1	N:n+16	**Media Time Available.** Indicates whether the MA USB host has access to a synchronized Media Time. Value — Meaning 0 — MA USB host does not have access to Media Time 1 — MA USB host has access to Media Time
7	N:n+17	**Reserved.**

6.3.2.2 Link Sleep Capability descriptor

The Link Sleep Capability descriptor indicates whether the MA USB host can receive a Sleep Request packet (Section 6.3.56) from a target MA USB device to take the session state to Session Inactive without the MA USB host suspending the USB device integrated into the target MA USB device first. Table 11 lists the descriptor fields. The absence of the descriptor is equivalent to the Link Sleep Capable field set to 0.

Table 11—Link Sleep Capability descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.
8	N:n+8	**MA Host Capability Type.** Indicates the type of the descriptor. Set to the value of Link Sleep Capability in Table 9.
1	N:n+16	**Link Sleep Capable.** Indicates whether the MA USB host can receive a Sleep Request packet (Section 6.3.56) from a target MA USB device to take the session state to Session Inactive without suspending the integrated USB device. Value — Meaning 0 — Cannot receive a Sleep Request packet without USB suspend 1 — Can receive a Sleep Request packet without USB suspend
7	N:n+17	**Reserved.**

6.3.3 MA USB Capability Response (CapResp)

The MA USB Capability Response (CapResp) packet is transmitted by the target MA USB device in response to an MA USB Capability Request (CapReq) packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 1 (CapResp), respectively. The Status Code field indicates whether the request was successfully completed.

The CapResp packet carries the fields listed in Table 12 after the management header.

Table 12—MA USB Capability Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Number of Endpoints**. Indicates the maximum number of endpoints for which the MA USB device can track state. For MA USB hubs, the minimum value of this field is 16.
8	3:16	**Number of Devices.** Indicates the maximum number of USB devices the MA USB device can manage; only MA USB hubs can manage multiple (two or more) USB devices.

5	3:24	**Number of Streams.** 2 to the power of the value of this field is the maximum number of streams supported by the MA USB device for any of its Enhanced SuperSpeed bulk endpoints.
3	3:29	**Device Type.** Indicates whether the MA USB device is an MA USB hub or not. Value — Meaning 0 — Not an MA USB hub 1 — MA USB hub with an integrated USB 2.0 hub 2 — MA USB hub with an integrated USB 3.1 hub 3-8 — Reserved
8	4:0	**Descriptors Count.** Indicates the total number of MA Device Capabilities descriptors present.
24	4:8	**Descriptors Length.** Indicates the total size of MA Device Capabilities descriptors in bytes.
16	5:0	**Number of Outstanding Transfer Requests.** Indicates the maximum number of total outstanding transfer requests that the MA USB device can track.
12	5:16	**Number of Outstanding Management Requests.** Indicates the maximum number of host initiated outstanding management requests that MA USB device can track. NOTE—It is recommended that the number of outstanding management requests the MA USB device supports should be at least 2 times the maximum number of endpoints plus 2 times the total number of devices it supports.
4	5:28	**Reserved.**

In addition, the CapResp packet may carry one or multiple MA Device Capability descriptors. The format of the MA Device Capability descriptors is defined in Table 13. The MA Device Capability descriptors are byte aligned.

Table 13—Format of MA Device Capability descriptors

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes.
8	N:n+8	**MA Device Capability Type.** Indicates the type of the descriptor. Descriptor types are listed in Table 14.
Variable	N:n+16	**Descriptor specific format.**

The valid MA Device Capability Types to be carried in the CapResp packet are listed in Table 14.

Table 14—MA Device Capability Type values

MA Device Capability Type	Value	Description
Speed Capability	0	Indicates the speed capability of the USB device behind the MA USB device. If the Device Type field is set to 0 or 2, the Speed Capability descriptor shall be present.

MA Device Capability Type	Value	Description
P-managed OUT Capabilities	1	Indicates the optional capabilities related to P-managed OUT transfers that the MA USB device supports. Shall be present if the MA USB device supports any of the P-managed OUT optional capabilities.
Isochronous Capabilities	2	Indicates the isochronous related capabilities of the MA USB device. Shall be present if the Device Type field is not set to 0, or if the MA USB device supports isochronous endpoints.
Synchronization Capabilities	3	Indicates the synchronization related capabilities of the MA USB device. Shall be present if the Device Type field is not set to 0, or if the MA USB device supports isochronous endpoints.
Container ID Capability	4	Indicates the capability related to support of Container ID descriptor by the integrated USB device behind the MA USB device PAL. Shall be present if the integrated USB device supports Container ID descriptor (Section 4.4.4).
Link Sleep Capability	5	Indicates the capability of the MA USB device related to the session state transition to Session Inactive. Shall be present if the MA USB device supports transition of the session state to Session Inactive without the integrated USB device being suspended.

The following sections define the MA Device Capability descriptors.

6.3.3.1 Speed Capability descriptor

If the Device Type field is set to 0 the CapResp packet carries the Speed Capability descriptor for the single USB device behind the MA USB device. If the Device Type field is set to 2 the CapResp packet carries the Speed Capability descriptor for the integrated Enhanced SuperSpeed hub in the MA USB hub. Table 15 illustrates the format of the Speed Capability descriptor.

Table 15—Speed Capability descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 4.
8	N:n+8	**MA Capability Type.** Indicates the type of the descriptor. Set to the value of Speed Capability in Table 14.
4	N:n+16	**Reserved.**
4	N:n+20	**Speed.** Indicates the speed of the USB device behind the MA USB device. Value — Meaning 0 — Low-Speed 1 — Full-Speed 2 — High-Speed 3 — SuperSpeed 4 — SuperSpeedPlus 5-15 — Reserved

4	N:n+24	**Reserved.**
2	N:n+28	**Lane Speed Exponent (LSE).** Indicates the LSE of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.
2	N:n+30	**Reserved.**

6.3.3.2 P-managed OUT Capabilities descriptor

The P-managed OUT Capabilities descriptor shall be present if the MA USB device supports any of the optional capabilities related to the P-managed OUT transfer. Absent of the descriptor is equivalent to the P-managed OUT Capability Bitmap field set to 0. Table 16 illustrates the format of the P-managed OUT Capabilities descriptor.

Table 16—P-managed OUT Capabilities descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.
8	N:n+8	**MA Capability Type.** Indicates the type of the descriptor. Set to the value of P-managed OUT Capabilities in Table 14.
8	N:n+16	**P-managed OUT Capability Bitmap.** Indicates the optional capabilities the MA USB device supports.

Table 17 illustrates the format of the P-managed OUT Capability Bitmap field.

Table 17—P-managed OUT Capability Bitmap format

Width (bits)	Offset (DW:bit)	Description
1	N:n+16	**Elastic Buffer Capability.** Indicates whether the MA USB host is allowed to exceed the amount of credit advertised by the MA USB device during p-managed OUT transfers targeting the MA USB device. Value — Meaning 0 — MA USB host shall not transmit data if no credit is available 1 — MA USB host may transmit data if no credit is available
1	N:n+17	**Drop Notification.** Indicates whether the MA USB device can return an explicit DROPPED_PACKET status during p-managed OUT transfers. Value — Meaning 0 — MA USB device cannot return a DROPPED_PACKET status 1 — MA USB device can return a DROPPED_PACKET status
6	N:n+18	**Reserved.**

6.3.3.3 Isochronous Capabilities descriptor

The Isochronous Capabilities descriptor reports the isochronous related capabilities of the MA USB device and shall be present if the MA USB device supports isochronous endpoints or the Device Type field is not set to 0. Table 18 illustrates the format of the Isochronous Capabilities descriptor.

Table 18—Isochronous Capabilities descriptor

Width (bits)	Offset (DW:bit)	Description

8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.
8	N:n+8	**MA Capability Type.** Indicates the type of the descriptor. Set to the value of Isochronous Capabilities in Table 14.
1	N:n+16	**Isochronous payload alignment.** Indicates whether the MA USB device requires DWORD aligned payload for isochronous transfers. Value — Meaning 0 — MA USB device transmits and expects to receive byte-aligned isochronous payload 1 — MA USB device transmits and expects to receive DWORD-aligned isochronous payload
7	N:n+17	**Reserved.**

6.3.3.4 Synchronization Capabilities descriptor

The Synchronization Capabilities descriptor reports the synchronization related capabilities of the MA USB device and shall be present if the MA USB device supports isochronous endpoint(s) or the Device Type field is not set to 0. Table 19 illustrates the format of the Synchronization Capabilities descriptor.

Table 19—Synchronization Capabilities descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.
8	N:n+8	**MA Capability Type.** Indicates the type of the descriptor. Set to the value of Synchronization Capabilities in Table 14.
1	N:n+16	**Media Time Available.** Indicates whether the MA USB device has access to a synchronized Media Time. Value — Meaning 0 — MA USB device does not have access to Media Time 1 — MA USB device has access to Media Time
1	N:n+17	**Timestamp Request.** Indicates the need to receive MA USB timestamps from the MA USB host through unicast or broadcast packets. Value — Meaning 0 — Need to receive MA USB timestamps only when the MA USB device has isochronous endpoints selected 1 — Need to receive MA USB timestamps regardless of endpoint configuration NOTE — MA USB hubs shall set this field to1. NOTE — Some MA USB devices may not have any isochronous endpoints selected when the CapResp packet is transmitted.
6	N:n+18	**Reserved.**

6.3.3.5 Container ID Capability descriptor

The Container ID Capability descriptor shall be present if the integrated USB device behind the MA USB device PAL supports the Container ID descriptor and reports its value. Table 20 illustrates the format of the Container ID Capability descriptor.

Table 20—Container ID Capability descriptor

Width	Offset	Description

(bits)	(DW:bit)	
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 18.
8	N:n+8	**MA Capability Type.** Indicates the type of the descriptor. Set to the value of Container ID Capability in Table 14.
128	N:n+16	**Container ID.** Indicates the value of Container ID.

6.3.3.6 Link Sleep Capability descriptor

The Link Sleep Capability descriptor indicates whether the MA USB device can receive a Sleep Request packet (Section 6.3.56) from the MA USB host to take the session state to Session Inactive without the MA USB host suspending the USB device integrated into the MA USB device first. Table 21 lists the descriptor fields. The absence of the descriptor is equivalent to the Link Sleep Capable field set to 0.

Table 21—Link Sleep Capability descriptor

Width (bits)	Offset (DW:bit)	Description
8	N:n	**Length.** Indicates the length of the descriptor in bytes. Set to 3.
8	N:n+8	**MA Device Capability Type.** Indicates the type of the descriptor. Set to the value of Remote Link Sleep in Table 14.
1	N:n+16	**Link Sleep Capable.** Indicates whether the MA USB device can receive a Sleep Request packet (Section 6.3.56) from the MA USB host to take the session state to Session Inactive without suspending the integrated USB device. Value — Meaning 0 — Cannot receive a Sleep Request packet without USB suspend 1 — Can receive a Sleep Request packet without USB suspend
7	N:n+17	**Reserved.**

6.3.4 USB Device Handle Request (USBDevHandleReq)

The USB Device Handle Request (USBDevHandleReq) packet is transmitted by the MA USB host to a target MA USB device to trigger the MA USB device to assign a handle to the USB device behind it. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 2 (USBDevHandleReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The USBDevHandleReq packet carries the fields listed in Table 22 after the management header.

Table 22—USB Device Handle Request fields

Width (bits)	Offset (DW:bit)	Description
20	3:0	**MA USB Route String**. Identifies the USB topology as defined in Section 4.2 and the corresponding route string, as defined in [USB 3.1]. NOTE — Route strings do not include the root hub port number. For USB devices behind an MA USB hub, the first entry in the route string refers to the port number on the hub integrated into the MA USB hub through which the USB device is accessed.
4	3:20	**Speed.** Identifies the speed of the USB device as following, Value — Meaning 0 — Low-Speed 1 — Full-Speed

		2 High-Speed 3 SuperSpeed 4 SuperSpeedPlus 5-15 Reserved
8	3:24	**Reserved.**
16	4:0	**Hub.** Identifies the device handle of the hub inside the MA USB hub through which the USB device is accessed (USB device may not be directly attached to this hub). Reserved for integrated USB devices.
16	4:16	**Reserved.**
16	5:0	**Parent HS Hub.** If the device is LS or FS (including an LS or FS hub) and is accessed through an HS hub, this field identifies the device handle of the "parent" HS hub that isolates LS or FS signaling on its downstream facing port from HS signaling on its upstream facing port. Reserved for other device speeds.
4	5:16	**Parent HS Hub Port.** If the device is LS or FS (including an LS or FS hub) accessed through an HS hub, this field identifies the port number on the parent HS hub (identified by the Parent HS Hub field above) to which the LS or FS device is directly attached. Reserved for other device speeds.
1	5:20	**MTT (Multiple Transaction Translators).** Set to 1 for an LS or FS USB device, if it is connected through an HS hub that has Multiple TTs support enabled by software. The value of the field is set to 0 otherwise. See [USB 2.0] for details.
2	5:21	**Lane Speed Exponent (LSE).** Indicates the LSE of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.
9	5:23	**Reserved.**

6.3.5 USB Device Handle Response (USBDevHandleResp)

The USB Device Handle Request (USBDevHandleResp) packet is transmitted by the target MA USB device to the MA USB host in response to a USBDevHandleReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 3 (USBDevHandleResp), respectively. The Status Code field indicates whether the request was successfully completed.

The USBDevHandleResp packet carries the fields listed in Table 23 after the management header.

Table 23—USB Device Handle Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**USB Device Handle.** Handle of the USB device.
16	3:16	**Reserved.**

6.3.6 Endpoint Handle Request (EPHandleReq)

The Endpoint Handle Request (EPHandleReq) packet is transmitted by the MA USB host to request a target MA USB device to assign a list of handles to the endpoints of a designated USB device behind the MA USB device. The Device Handle field carries the handle of the USB device behind the MA USB device for which the endpoint handle list is requested, i.e., the handle returned by the USBDevHandleResp packet. The Type and Subtype fields are set to 0 (Management) and 4 (EPHandleReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPHandleReq packet carries the fields listed in Table 24 after the management header.

Table 24—Endpoint Handle Request fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Descriptors.** Indicates the number of EP descriptors carried in the EPHandleReq packet.
6	3:5	**Size of EP Descriptor.** Indicates the size of each EP descriptor included in the EPHandleReq packet, in bytes. The size is inclusive of the bytes padded to make each EP descriptor DWORD-aligned.
21	3:11	**Reserved.**

In addition, each EPHandleReq packet carries one or more EP descriptors. All of the EP descriptors included in an EP Handle Request packet are of the same size, defined by the Size of EP Descriptor field.

Table 25 illustrates the format of each EP descriptor. The size of the EP descriptor depends on the speed of the USB device to which the EP belongs:

- For an LS, FS, or HS device, the EP descriptor takes 8 bytes (including 1 byte of zero padding).
- For an Enhanced SuperSpeed device the EP descriptor takes 16 bytes (including 3 bytes of zero padding).
- For an Enhanced SuperSpeed device operating at above Gen 1 speed with isochronous endpoint(s), the EP descriptor takes 24 bytes (including 3 bytes of zero padding).

Table 25—EP descriptor

Width (bytes)	Offset (DW:bit)	Description
7	N:0	Standard endpoint descriptor, as defined in [USB 2.0].
6	N+1:24	(For Enhanced SuperSpeed devices only) SuperSpeed Endpoint Companion Descriptor, as defined in [USB 3.1].
8	N+3:8	(For Enhanced SuperSpeed devices operating at above Gen 1 speed with isochronous EPs only) SuperSpeedPlus Isochronous Endpoint Companion Descriptor, as defined in [USB 3.1].
Variable	Variable	Zero padding to make the EP descriptor DWORD-aligned.

6.3.7 Endpoint Handle Response (EPHandleResp)

The Endpoint Handle Response (EPHandleResp) packet is transmitted by the target MA USB device to the MA USB host, and includes the list of endpoint handles requested by the MA USB host. The Device Handle field carries the handle of the USB device behind the MA USB device for which the endpoint handle list is returned. The Type and Subtype fields are set to 0 (Management) and 5 (EPHandleResp), respectively. The Status Code field indicates whether the request was successfully completed. The request is considered successful if there is at least one endpoint for which EP handle is successfully allocated.

The EPHandleResp packet carries the fields listed in

Table 26 after the management header.

Table 26—EP Handle Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of MA USB EP Descriptors.** Indicates the number of MA USB EP descriptors carried in the EPHandleResp packet. This field is set to the same value as the Number of EP Descriptors field in the corresponding EPHandleReq packet.
27	3:5	**Reserved.**

In addition, the EPHandleResp packet carries one or more MA USB EP descriptors. MA USB EP descriptors are inserted in the same order as the corresponding EP descriptors in the EPHandleReq packet.

Table 27 illustrates the format of each MA USB EP descriptor. Each descriptor takes 16 bytes.

Table 27—MA USB EP descriptor format

Width (bits)	Offset (DW:bit)	Description
16	N:0	**EP Handle.** The handle of the endpoint requested in the EPHandleReq packet.
1	N:16	**Direction.** Endpoint direction. Value — Meaning 0 — Control or OUT endpoint 1 — IN endpoint
1	N:17	**Isochronous.** Indicates an isochronous endpoint. Value — Meaning 0 — Non-isochronous endpoint 1 — Isochronous endpoint
1	N:18	**L-managed.** For a control or non-isochronous OUT endpoint, indicates whether the endpoint can be accessed through an l-managed transfer (in addition to the mandatory p-managed transfer). The field is reserved for all other endpoint types. Value — Meaning 0 — L-managed transfers not supported 1 — L-managed transfers supported
1	N:19	**Valid.** Indicates whether the handle for the endpoint was successfully allocated and the handle in the EP Handle field is valid. Value — Meaning 0 — The handle is valid. 1 — The handle is not valid. NOTE — If the handle is not valid the EP handle remains in Unassigned state.
12	N:20	**Reserved.**
16	N+1:0	**Credit Consumption Unit (CCU).** For a control or non-isochronous OUT endpoint, the field indicates the buffer size (in bytes) that the MA USB host must assume as the unit of credit consumption for all p-managed transfers targeting the endpoint. The field is reserved for all other endpoints.
16	N+1:16	**Reserved.**

32	N+2:0	**Buffer Size.** For control and OUT endpoints, the field indicates the buffer size (in bytes) available to transfers targeting the endpoint. The field is reserved for IN endpoints.
16	N+3:0	**Isochronous Programming Delay.** For isochronous endpoints, the field indicates the maximum time, in microseconds, from the moment the first bit of an IsochTransferReq packet is received by the MA USB device to the moment the programming of the first isochronous transfer corresponding to that IsochTransferReq packet on the local bus is complete. The field is reserved for non-isochronous endpoints. NOTE — The value of this field is denoted by pMaxDeviceIsochINProgDelay for isochronous IN endpoints (Section 5.10.2), and by pMaxDeviceIsochOUTProgDelay for isochronous OUT endpoints (Section 5.10.3).
16	N+3:16	**Isochronous Response Delay.** For isochronous endpoints, the field indicates the maximum time, in microseconds, from the moment the end of the last Service Interval an isochronous MA USB transfer targets to the moment the first bit of the last IsochTransferResp packet corresponding to the transfer is released to the network. The field is reserved for non-isochronous endpoints. NOTE — The value of this field is denoted by pMaxDeviceIsochINRespDelay for isochronous IN endpoints (Section 5.10.2), and by pMaxDeviceIsochOUTRespDelay for isochronous OUT endpoints (Section 5.10.3).

6.3.8 Endpoint Activate Request (EPActivateReq)

The Endpoint Activate Request (EPActivateReq) packet is transmitted by the MA USB host to a target MA USB device to change the status of a set of EP handles behind the MA USB device from inactive to active. The Device Handle field carries the handle of the USB device behind the MA USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 6 (EPActivateReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPActivateReq packet carries the fields listed in Table 28 after the management header.

Table 28—Endpoint Activate Request fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles.** Number of EP handles included in the packet.
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles the MA USB host is requesting to activate, concatenated in 16-bit increments.

6.3.9 Endpoint Activate Response (EPActivateResp)

The Endpoint Activate Response (EPActivateResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPActivateReq packet. The Device Handle field carries the handle of the USB device behind the MA USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 7 (EPActivateResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPActivateResp packet carries the fields listed in Table 29 after the management header.

Table 29—Endpoint Activate Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles with Error.** Number of EP handles whose activation did not complete. This field is nonzero only if the Status Code field in the packet carries a value other than SUCCESS (NO_ERROR).
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles whose activation failed, concatenated in 16-bit increments. The list is empty when all EP handles in the corresponding EPActivateReq packet were successfully activated. If the Endpoint Activate Request is executed as an atomic operation then the list may be empty when the Status Code field is set to Failure.

6.3.10 Endpoint Inactivate Request (EPInactivateReq)

The Endpoint Inactivate Request (EPInactivateReq) packet is transmitted by the MA USB host to a target MA USB device to inactivate a set of EP handles under the MA USB device management. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 8 (EPInactivateReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPInactivateReq packet carries the fields listed in Table 30 after the management header.

Table 30—Endpoint Inactivate Request fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles.** Number of EP handles included in the packet.
1	3:5	**Suspend (SP) Flag.** Indicates whether the EPInactivateReq is issued due to suspension of the endpoints for which EP Handles are included. Value — Meaning 0 — Endpoints not suspended 1 — Endpoints suspended
26	3:6	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles the MA USB host is requesting to inactivate, concatenated in 16-bit increments.

NOTE — Endpoint handles may be inactivated for a number of reasons. Examples are inactivating an EP handle prior to deleting the handle, and inactivating an EP handle prior to cancelling a transfer targeting the corresponding endpoint.

6.3.11 Endpoint Inactivate Response (EPInactivateResp)

The Endpoint Inactivate Response (EPInactivateResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPInactivateReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 9 (EPInactivateResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPInactivateResp packet carries the fields listed in Table 31 after the management header.

Table 31—Endpoint Inactivate Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles with Error.** Number of EP handles whose inactivation did not successfully complete. This field is nonzero only if the Status Code field in the packet carries a value other than SUCCESS (NO_ERROR).
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles whose inactivation failed, concatenated in 16-bit increments. The list is empty when all endpoints specified in the corresponding EPInactivateReq packet have been successfully inactivated. If the Endpoint Inactivate Request is executed as an atomic operation then the list may be empty when the Status Code field is set to Failure. The Endpoint Inactivate Request is expected to be successfully completed unless the command is not valid.

6.3.12 Endpoint Reset Request (EPResetReq)

The Endpoint Reset Request (EPResetReq) packet is transmitted by the MA USB host to a target MA USB device to change the status of a set of endpoint handles behind the MA USB device from halted to inactive. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 10 (EPResetReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPResetReq packet carries the fields listed in Table 32 after the management header.

Table 32—Endpoint Reset Request fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP reset information blocks.** Number of the EP reset information blocks included in the packet (Table 33).
27	3:5	**Reserved.**

In addition, each EPResetReq packet carries a set of EP reset information blocks, where the format of each block is shown in Table 33.

Table 33—EP reset information block

Width (bits)	Offset (DW:bit)	Description
16	N:0	**EP Handle.** The EP handle the MA USB host is requesting to reset.
1	N: 16	**Transfer State Preserve (TSP) Flag.** Indicates whether the value of the data toggle bit [USB 2.0] or the sequence number [USB 3.1] shall be preserved for the endpoint. Value — Meaning 0 — Value of the data toggle bit [USB 2.0] or sequence number [USB 3.1] is reset. 1 — Value of the data toggle bit [USB 2.0] or sequence number [USB 3.1] is preserved.

		NOTE — Resetting data toggle [USB 2.0] or sequence number [USB 3.1] of an endpoint not in a halted state, requires the MA USB host to first inactivate the endpoint if the endpoint handle is in Active State (using EPInactivateReq packet), then delete the handle of the endpoint (using EPHandleDeleteReq packet), and then request a new endpoint handle with the original EP descriptor (using EPHandleReq packet). IMPLEMENTATION NOTE — If the value of this field is 0, the USB controller hardware should invalidate any cached Transfer Descriptors and fetch the next Transfer Request Block when endpoint transits from stopped to running state. If the value of this field is 1, the USB controller hardware should retry last transaction once the endpoint is transitioned from stopped to running state, providing that no other commands were performed on the endpoint.
15	N:17	**Reserved.**

6.3.13 Endpoint Reset Response (EPResetResp)

The Endpoint Reset Response (EPResetResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPResetResp packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 11 (EPResetResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPResetResp packet carries the fields listed in Table 34 after the management header.

Table 34—Endpoint Reset Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles with Error.** Number of EP handles whose reset did not successfully complete. This field is nonzero only if the Status Code field in the packet carries a value other than SUCCESS (NO_ERROR).
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** EP handles whose reset failed, concatenated in 16-bit increments. The list is empty when all EP handles in the corresponding EPResetReq packet were successfully reset. If the Endpoint Reset Request is executed as an atomic operation then the list may be empty when the Status Code field is set to Failure.

6.3.14 Clear Transfers Request (ClearTransfersReq)

The Clear Transfers Request (ClearTransfersReq) packet is transmitted by the MA USB host to a target MA USB device to clear all outstanding transfers targeted at a set of endpoints behind the MA USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 12 (ClearTransfersReq), respectively. The Status Code field is set to 0 (NO_ERROR). ClearTransfersReq packet carries the fields listed in Table 35 after the management header.

Table 35—Clear Transfers Request fields

Width (bits)	Offset (DW:bit)	Description
8	3:0	**Number of clear transfers information blocks.** Number of clear transfers information blocks (Table 36) included in the packet.

		NOTE — The MA USB host may be required to limit the number of clear transfers information blocks included in the packet streams to meet the MA link MTU size.
24	3:8	**Reserved.**

In addition, each ClearTransfersReq packet carries a set of clear transfer Information blocks; the format of each block is shown in Table 36.

Table 36—Clear transfers information block

Width (bits)	Offset (DW:bit)	Description
16	N:0	**EP Handle.** The Endpoint Handle the request is targeting.
16	N:16	**Stream ID.** Indicates the target stream when the target endpoint is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol; reserved otherwise.
8	N+1:0	**Start Request ID.** Indicates the value of the Request ID that MA USB host uses following the ClearTransfersReq packet, which is not the target of this request; i.e., The ClearTransfersReq targets all the transfers carrying a preceding request ID values. The scope of cancellation are transfers carrying Request IDs less than Start Request ID. A transfer with Request ID set to Start Request ID or larger is not in the scope of cancellation. .
24	N+1:8	**Reserved.**

6.3.15 Clear Transfers Response (ClearTransfersResp)

The Clear Transfers Response (ClearTransfersResp) packet is transmitted by the target MA USB device to the MA USB host in response to a ClearTransfersReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 13 (ClearTransfersResp), respectively. The Status Code field indicates whether the request was successfully completed.

ClearTransfersResp packet carries the fields listed in Table 37 after the management header.

Table 37—Clear Transfers Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of cancel transfers status blocks.** Number of cancel transfers status blocks included in the packet (Table 38). The number of cancel transfers status blocks shall be equal to the value of Number of cancel transfers information blocks field in the corresponding ClearTransfersReq packet.
27	3:5	**Reserved.**

In addition, each ClearTransfersResp packet carries a set of cancel transfers status blocks; the format of each block is shown in Table 38.

Table 38— Cancel transfers status block

Width (bits)	Offset (DW:bit)	Description
16	N:0	**EP Handle.** Indicates the endpoint for which the status is being returned.

16	N:16	**Stream ID.** Indicates the stream to which the response is related when the endpoint identified in the EP Handle field is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol; reserved otherwise.
1	N+1:0	**Cancellation Status.** Set to 1 if the request was successfully completed for the transfers identified by the EP Handle, Stream ID, and Start Request ID of the corresponding request or the Status Code field is set to SUCCESS. Set to 0 otherwise.
1	N+1:1	**Partial Delivery.** Set to 1 if the last cancelled transfer was not completed, i.e., some (but not all) data related to the transfer was moved to/from the device before its cancellation.
30	N+1:2	**Reserved.**
8	N+2:0	**Last Request ID.** Indicates the value of the Request ID field in the last TransferResp packet created by the MA USB Device.
24	N+2:8	**Delivered Sequence Number.** For an OUT transfer, indicates the last sequence number delivered to the device. Valid if Partial Delivery field is set to 1 and reserved otherwise. Reserved for IN transfers.
32	N+3:0	**Delivered Byte Offset.** Indicates the total amount of data related to the transfer delivered to the device identified by the sequence number value in the Delivered Sequence Number field. Valid if partial delivery field is set to 1 and reserved otherwise. Reserved for IN transfers.

6.3.16 Endpoint Handle Delete Request (EPHandleDeleteReq)

The Endpoint Handle Delete Request (EPHandleDeleteReq) packet is transmitted by the MA USB host to a target MA USB device to delete the handles of a set of endpoints behind the MA USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 14 (EPHandleDeleteReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPHandleDeleteReq packet carries the fields listed in Table 39 after the management header.

Table 39—Endpoint Handle Delete Request fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles.** Number of EP handles included in the packet.
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles the MA USB host is requesting to delete, concatenated in 16-bit increments.

6.3.17 Endpoint Handle Delete Response (EPHandleDeleteResp)

The Endpoint Handle Delete Response (EPHandleDeleteResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPHandleDeleteReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 15 (EPHandleDeleteResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPHandleDeleteResp packet carries the fields listed in Table 40 after the management header.

Table 40—Endpoint Handle Delete Response fields

Width (bits)	Offset (DW:bit)	Description
5	3:0	**Number of EP Handles with Error.** Number of EP handles that could not be deleted. This field is nonzero only if the Status Code field in the packet carries a value other than SUCCESS (NO_ERROR).
27	3:5	**Reserved.**
Variable	4:0	**EP Handle List.** List of EP handles that could not be deleted, concatenated in 16-bit increments. The list is empty when all EP handles in the corresponding EPHandleDeleteReq packet were successfully deleted. If the Endpoint Handle Delete Request is executed as an atomic operation then the list may be empty when the Status Code field is set to Failure.

6.3.18 MA USB Device Reset Request (DevResetReq)

The MA USB Device Reset Request (DevResetReq) packet is transmitted by the MA USB host to a target MA USB device to clear all of the MA USB device state. The Device Handle and Dialog Token fields are reserved. The Type and Subtype fields are set to 0 (Management) and 16 (DevResetReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.19 MA USB Device Reset Response (DevResetResp)

The MA USB Device Reset Response (DevResetResp) packet is transmitted by a target MA USB device to the MA USB host in response to a DevResetReq packet. The Device Handle and Dialog Token fields are reserved. The Type and Subtype fields are set to 0 (Management) and 17 (DevResetResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.20 Modify EP0 Request (ModifyEP0Req)

The Modify EP0 Request (ModifyEp0Req) packet is transmitted by the MA USB host to a target MA USB device to modify the default endpoint parameters and/or request an updated EP0 handle. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 18 (ModifyEP0Req), respectively. The Status Code field is set to 0 (NO_ERROR).

The ModifyEP0Req packet carries the fields listed in Table 41 after the management header.

Table 41—Modify EP0 Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP0 Handle.** EP0 handle of the USB device identified by the Device Handle field.
16	3:16	**Max Packet Size.** Updated value of the maximum packet size for EP0 handle, in bytes.

6.3.21 Modify EP0 Response (ModifyEP0Resp)

The Modify EP0 Response (ModifyEP0Resp) packet is transmitted by the target MA USB device to the MA USB host in response to a ModifyEP0Req packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0

(Management) and 19 (ModifyEP0Resp), respectively. The Status Code field indicates whether the request was successfully completed.

The ModifyEP0Resp packet carries the fields listed in Table 42 after the management header.

Table 42—Modify EP0 Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP0 Handle.** Present only in the following cases: a) The USB Address subfield of the EP0 Handle field in the corresponding ModifyEP0Req packet is set to 0, and the USB device identified by the Device Handle field is in the addressed state [USB 2.0]. The EP0 Handle field contains the EP0 handle of the USB device with a nonzero USB Address subfield. b) The USB Address subfield of the EP0 Handle field in the corresponding ModifyEP0Req packet is not 0, and the USB device identified by the Device Handle field is in Default state [USB 2.0]. The EP0 Handle field contains the EP0 handle of the USB device with a USB Address subfield set to 0.
16	3:16	**Reserved.**

6.3.22 Set USB Device Address Request (SetUSBDevAddrReq)

The Set USB Device Address Request (SetUSBDevAddrReq) packet is transmitted by the MA USB host to a target MA USB device to set the USB address of a USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 20 (SetUSBDevAddrReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The SetUSBDevAddrReq packet carries the fields listed in Table 43 after the management header.

Table 43—Set USB Device Address Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Response Timeout.** The duration in milliseconds by which the USB device is expected to respond to the USB Address Device command. If no response is received from the USB device in the expected response time the MA USB device shall respond to the SetUSBDevAddrReq with status code set to UNSUCCESSFUL.
16	3:16	**Reserved.**

6.3.23 Set USB Device Address Response (SetUSBDevAddrResp)

The Set Device USB Address Response (SetUSBDevAddrResp) packet is transmitted by the target MA USB device to the MA USB host in response to a SetUSBDevAddrReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 21 (SetUSBDevAddrResp), respectively. The Status Code field indicates whether the request was successfully completed.

The SetUSBDevAddrResp packet carries the fields listed in Table 44 after the management header.

Table 44—Set USB Device Address Response fields

Width (bits)	Offset (DW:bit)	Description
7	3:0	**USB Device Address.** The USB address of the USB device requested in SetUSBDevAddrReq packet.
25	3:7	**Reserved.**

6.3.24 Update Device Request (UpdateDevReq)

The Update Device Request (UpdateDevReq) packet is transmitted by the MA USB host to a target MA USB device to modify the device parameters of a USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 22 (UpdateDevReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The UpdateDevReq packet carries the fields listed in Table 45 after the management header.

Table 45—Update Device Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Max Exit Latency.** The worst case exit latency for the links and hubs between the target USB device and the integrated USB hub on the MA USB hub. The latency associated with the integrated USB hub is included in the value of the Max Exit Latency field if the Integrated Hub Latency field is set to 1, and is excluded otherwise. See [USB 3.1] for a description of how to calculate the Maximum Exit Latency. Set to 0 if the integrated USB device is not a hub.
1	3:16	**Hub.** Indicates whether the USB device is a hub. Value — Description 0 — Not a hub 1 — Hub
4	3:17	**Number of Ports.** Number of downstream facing ports of the USB device if the USB device is a hub. Reserved otherwise.
1	3:21	**MTT (Multiple Transaction Translators).** Set to 1 for an HS hub that has Multiple Transaction Translators (TTs) enabled by software and set to 0 otherwise. See [USB 2.0] for details.
2	3:22	**TTT (Transaction Translator Think Time).** Time required by an HS hub to proceed to the next FS/LS transaction. Reserved if the target USB device is not an HS hub. Value — Description 0 — TT requires at most 8 FS bit times of inter-transaction gap on an LS or FS downstream bus. 1 — TT requires at most 16 FS bit times. 2 — TT requires at most 24 FS bit times. 3 — TT requires at most 32 FS bit times.
1	3:24	**Integrated Hub Latency.** If the USB device is a hub indicates whether the latency associated with the integrated hub is included in the value of Max Exit Latency field. Reserved if the USB device is not a hub. Value — Description 0 — Integrated hub latency excluded. 1 — Integrated hub latency included.

		NOTE — The MA USB device is expected to be aware of the value of the integrated hub latency.
7	3:25	**Reserved.**

In addition, the UpdateDevReq packet carries the device descriptor.

Table 46 illustrates the format of the device descriptor of the target USB device. The descriptor takes 18 bytes.

Table 46—Device descriptor

Width (bytes)	Offset (DW:bit)	Description
18	4:0	Standard device descriptor, as defined in [USB 2.0] and [USB 3.1].
2	8:16	Reserved.

6.3.25 Update Device Response (UpdateDevResp)

The Update Device Response (UpdateDevResp) packet is transmitted by the target MA USB device to the MA USB host in response to an UpdateDevReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 23 (UpdateDevResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.26 USB Device Disconnect Request (USBDevDisconnectReq)

The USB Device Disconnect Request (USBDevDisconnectReq) packet is transmitted by the MA USB host to a target MA USB device to clear the MA USB state of a USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 24 (USBDevDisconnectReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.27 USB Device Disconnect Response (USBDevDisconnectResp)

The USB Device Disconnect Response (USBDevDisconnectResp) packet is transmitted by the target MA USB device to the MA USB host in response to a USBDevDisconnectReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 25 (USBDevDisconnectResp), respectively. The Status Code field indicates whether the request was successfully completed. Successful completion of the request results in the device handle of the target USB device becoming invalid.

6.3.28 USB Suspend Request (USBSuspendReq)

The USB Suspend Request (USBSuspendReq) packet is transmitted by the MA USB host to a target MA USB device to request the integrated USB device behind the MA USB device PAL to move to the Suspend state. The Device Handle field identifies the integrated USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 26 (USBSuspendReq), respectively. The Status Code value is set to 0 (NO_ERROR).

6.3.29 USB Suspend Response (USBSuspendResp)

The USB Suspend Response (USBSuspendResp) packet is transmitted by the target MA USB device to the MA USB host in response to a USBSuspendReq packet. The Device Handle field carries the same value as the Device Handle field in the corresponding USBSuspendReq packet. The Type and Subtype

fields are set to 0 (Management) and 27 (USBSuspendResp), respectively. The Status Code value indicates whether the request was successfully completed.

6.3.30 USB Resume Request (USBResumeReq)

The USB Resume Request (USBResumeReq) packet is transmitted by the MA USB host to a target MA USB device to request the integrated USB device behind the MA USB device PAL to move out of the Suspend state. The Device Handle field identifies the integrated USB device behind the MA USB device PAL the request is targeting. The Type and Subtype fields are set to 0 (Management) and 28 (USBResumeReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.31 USB Resume Response (USBResumeResp)

The USB Resume Response (USBResumeResp) packet is transmitted by the target MA USB device to the MA USB host in response to a USBResumeReq packet. The Device Handle field carries the same value as the Device Handle field in the corresponding USBResumeReq packet. The Type and Subtype fields are set to 0 (Management) and 29 (USBResumeResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.32 Remote Wake Request (RemoteWakeReq)

If the remote wake mechanism is supported, the MA USB Remote Wake Request (RemoteWakeReq) packet is transmitted by the MA USB device to the MA USB host to request the MA USB host to resume communication with the integrated USB device. The Device Handle field carries the handle of the integrated USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 30 (RemoteWakeReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The RemoteWakeReq packet carries the fields listed in Table 47 after the management header.

Table 47—Remote Wake Request fields

Width (bits)	Offset (DW:bit)	Description
1	3:0	**USB Device Resumed.** Indicates whether the USB device initiating the remote wake request is already resumed and ready to receive USB control packets or the MA USB device PAL expects a USB Device Resume Request packet to resume the USB device. Value — Meaning 0 — The USB device is in Suspend state 1 — The USB device is not in Suspend state
31	3:1	**Reserved.**

6.3.33 Remote Wake Response (RemoteWakeResp)

The Remote Wake Response (RemoteWakeResp) packet is transmitted by the MA USB host to the target MA USB device in response to a RemoteWakeReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 31 (RemoteWakeResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.34 Ping Request (PingReq)

The Ping Request (PingReq) packet is transmitted by the MA USB host or the MA USB device to verify or possibly re-establish network connectivity with a target MA USB device. The Device Address field is set to the address of the target MA USB device, or to 0xFF (any device). The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 32 (PingReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.35 Ping Response (PingResp)

The Ping Response (PingResp) packet is transmitted by the MA USB device or the MA USB host in response to a PingReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 33 (PingResp), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.36 MA USB Device Disconnect Request (DevDisconnectReq)

The MA USB Device Disconnect Request (DevDisconnectReq) packet is transmitted by the MA USB host to a target MA USB device to initiate transition to Session Down state. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 34 (DevDisconnectReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.37 MA USB Device Disconnect Response (DevDisconnectResp)

The MA USB Device Disconnect Response (DevDisconnectResp) packet is transmitted by the target MA USB device to the MA USB host in response to a DevDisconnectReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 35 (DevDisconnectResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.38 MA USB device Initiated Disconnect Request (DevInitDisconnectReq)

The MA USB device Initiated Disconnect Request (DevInitDisconnectReq) packet is transmitted by the MA USB device to initiate the MA USB host to transition to Session Down state. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 36 (DevInitDisconnectReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.39 MA USB device Initiated Disconnect Response (DevInitDisconnectResp)

The MA USB device Initiated Disconnect Response (DevInitDisconnectResp) packet is transmitted by the MA USB host to the target MA USB device in response to a DevInitDisconnectReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 37 (DevInitDisconnectResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.40 Synchronization Request (SynchReq)

The Synchronization Request (SynchReq) packet is transmitted by the MA USB host to a target MA USB device or to all MA USB devices in an MSS to deliver the MGT. The Device Handle and Dialog Token fields are reserved. The Type and Subtype fields are set to 0 (Management) and 38 (SynchReq), respectively. The Status Code field is set to 0 (NO_ERROR). The SynchReq packet carries the fields listed in Table 48 after the management header.

Table 48—Synchronization Request fields

Width (bits)	Offset (DW:bit)	Description

1	3:0	**MTD Valid.** Defined in Section 6.5.1.8 (Table 66).
1	3:1	**Response Required.** Set to 1 if a SynchResp packet is required. NOTE — For broadcast SynchReq packets that have this field set to1, only MA USB devices that have requested to receive MA USB timestamps are required to respond.
30	3:2	**Reserved.**
32	4:0	**MA USB Timestamp.** Defined in Section 6.5.1.11.
32	5:0	**Media Time/Transmission Delay.** Defined in Section 6.5.1.12.

6.3.41 Synchronization Response (SynchResp)

The Synchronization Response (SynchResp) packet is transmitted by the target MA USB device to the MA USB host in response to a SynchReq packet with unicast Device Address. An MA USB device may also transmit a SynchResp packet to the MA USB host in response to a SynchReq packet with broadcast Device Address. The Device Handle and Dialog Token fields are reserved. The Type and Subtype fields are set to 0 (Management) and 39 (SynchResp), respectively. The Status Code field is set to SUCCESS. The SynchResp packet carries the same fields as SynchReq packet (Table 48) after the management header, except that the Response Required field is reserved and the MA USB Timestamp field carries the MA USB Global Time at the target MA USB device at the moment the field is initialized.

NOTE — After transmitting a SynchReq packet with broadcast Device Address, the MA USB host may receive a number of SynchResp packets ranging from 0 to the number of MA USB devices in the MA USB Service Set.

NOTE — The latency compensation method for the MGT reading in a SynchResp packet is the same as the method used for the SynchReq packet that triggered the response.

6.3.42 Cancel Transfer Request (CancelTransferReq)

The Cancel Transfer Request (CancelTransferReq) packet is transmitted by the MA USB host to request cancellation of a transfer. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 40 (CancelTransferReq), respectively. The Status Code field is set to 0 (NO_ERROR). The Dialog Token field is reserved.

The CancelTransferReq packet carries the fields listed in Table 49 after the management header.

Table 49—Cancel Transfer Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP Handle.** Indicates the endpoint the request is targeting.
16	3:16	**Stream ID.** Indicates the target stream when the target endpoint is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol; reserved otherwise.
8	4:0	**Request ID.** Indicates the target request.
24	4:8	**Reserved.**

6.3.43 Cancel Transfer Response (CancelTransferResp)

The Cancel Transfer Response (CancelTransferResp) packet is transmitted by the MA USB device in response to a CancelTransferReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 41

(CancelTransferResp), respectively. The Status Code field indicates whether the request was successfully completed. The Dialog Token field is reserved.

The CancelTransferResp packet carries the fields listed in Table 50 after the management header.

Table 50—Cancel Transfer Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP Handle.** Indicates the endpoint for which the response is being returned.
16	3:16	**Stream ID.** Indicates the stream to which the response is related when the endpoint identified in the EP Handle field is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol; reserved otherwise.
8	4:0	**Request ID.** Indicates the request for which the response is being returned.
3	4:8	**Cancellation Status.** Set to 0 if the Status Code field is not set to SUCCESS. Set to 1 if the transfer was cancelled before any data was moved to/from the USB device. Set to 2 if the transfer was cancelled after some data was moved to/from the USB device. Set to 3 if the transfer was completed. Set to 4 if the transfer was not yet received. Set to 5 if the transfer was cleared without any data moved during ClearTransfersReq processing.
21	4:11	**Reserved.**
24	5:0	**Delivered Sequence Number.** For an OUT transfer, indicates the last sequence number delivered to the device. Valid if Cancellation Status field is set to 2 and reserved otherwise. Reserved for IN transfers.
8	5:24	**Reserved.**
32	6:0	**Delivered Byte Offset.** Indicates the amount of data identified by the sequence number value in the Delivered Sequence Number field delivered to the device. Valid if Cancellation Status field is set to 2 and reserved otherwise. Reserved for IN transfers.

6.3.44 Endpoint Open Stream Request (EPOpenStreamReq)

The Endpoint Open Stream Request (EPOpenStreamReq) packet is transmitted by the MA USB host to a target MA USB device to open streams on an Enhanced SuperSpeed bulk endpoint on a USB device behind the MA USB device. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 42 (EPOpenStreamReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPOpenStreamReq packet carries the fields listed in Table 51 after the management header.

Table 51—Endpoint Open Streams Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP Handle.** Indicates the Enhanced SuperSpeed bulk endpoint the request is targeting.
16	3:16	**Number of Streams.** Indicates the number of streams to be opened for the target endpoint.

16	4:0	**Stream ID Index.** Indicates the lower bound for the stream IDs the MA USB host is retrieving, if the value of Open Stream field is set to 0. Reserved otherwise.
1	4:16	**Allocation mode.** Indicates the rules associated with allocation of stream IDs; may be set to 1 only if the number of streams is less than 256. Value Meaning 0 The allocation of stream IDs is free of rules. 1 The allocation of stream IDs shall be sequential and start at 1.
1	4:17	**Open Stream.** Indicates whether the request is to open new streams or retrieve previously opened streams. Value Meaning 0 The request is to retrieve stream IDs of previously opened streams 1 The request is to open new streams
14	4:18	**Reserved.**

6.3.45 Endpoint Open Stream Response (EPOpenStreamResp)

The Endpoint Open Stream Response (EPOpenStreamResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPOpenStreamReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) 43 (EPOpenStreamResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPOpenStreamResp packet carries the fields listed in Table 52 after the management header.

Table 52—Endpoint Open Stream Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Number of Streams.** Indicates the number of streams that are included in the packet. If the Status Code field in the packet carries a value indicating SUCCESS (NO_ERROR) this field shall carry a value less than or equal to the value of the Number of Streams field in the corresponding EPOpenStreamReq packet. NOTE — The number of streams that are included in the packet may be less than the number requested in the EPOpenStreamReq packet to meet the MA link MTU size. An MA USB host that receives a smaller number of Stream IDs than it asked for may transmit additional EPOpenStreamReq packets with updated Number of Streams field to receive additional Stream IDs.
16	3:16	**Number of stream ID blocks.** Indicates the number of stream ID blocks included in this packet.

In addition, each EPOpenStreamResp packet carries a set of stream ID interval blocks. Each block identifies a range of consecutive stream IDs. The format of each block is shown in Table 53. The stream ID interval blocks shall be included in the increasing order of the stream ID values.

Table 53—Stream ID interval block

Width (bits)	Offset (DW:bit)	Description

16	N:0	**First Stream ID.** Indicates the value of the first stream ID in the interval block.
16	N:16	**Last Stream ID.** Indicates the value of the last stream ID in the interval block. If the values of the Last Stream ID and First Stream ID fields are equal there is only one Stream ID identified in the block.

6.3.46 Endpoint Close Stream Request (EPCloseStreamReq)

The Endpoint Close Stream Request (EPCloseStreamReq) packet is transmitted by the MA USB host to a target MA USB device to close streams on an Enhanced SuperSpeed bulk endpoint on a USB device under the MA USB device management. The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 44 (EPCloseStreamReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPCloseStreamReq packet carries the fields listed in Table 54 after the management header.

Table 54—Endpoint Close Stream Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**EP Handle**. Indicates the Enhanced SuperSpeed bulk endpoint the request is targeting.
1	3:16	**Close All.** Indicates whether the request targets all the open streams. Value — Meaning 0 — The request is for a subset of the open streams. 1 — The request is for all of the open streams.
15	3:17	**Reserved.**
16	4:0	**Number of stream ID blocks.** Indicates the number of stream ID blocks included in this packet. Set to 0 if the Close All field is set to 1.
16	4:16	**Reserved.**

In addition, each EPCloseStreamResp packet may carry a set of stream ID interval blocks. Each block identifies a range of consecutive stream IDs. The format of each block is shown in Table 53.

6.3.47 Endpoint Close Stream Response (EPCloseStreamResp)

The Endpoint Close Stream Response (EPCloseStreamResp) packet is transmitted by the target MA USB device to the MA USB host in response to an EPCloseStreamReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 45 (EPCloseStreamResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.48 USB Device Reset Request (USBDevResetReq)

The USB Device Reset Request (USBDevResetReq) packet is transmitted by the MA USB host to a target MA USB device to request reset of the integrated USB device, or in case of an MA USB hub, to inform the MA USB hub of the reset of a downstream USB device (as defined in Section 7.3.5). The Device Handle field carries the handle of the USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 46 (USBDevResetReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.49 USB Device Reset Response (USBDevResetResp)

The USB Device Reset Response (USBDevResetResp) packet is transmitted by the target MA USB device to the MA USB host in response to a USBDevResetReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) 47 (USBDevResetResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.50 Device Notification Request (DevNotificationReq)

The Device Notification Request (DevNotificationReq) packet is transmitted by the MA USB device to the MA USB host to carry USB defined device notifications [USB 3.1]. The Device Handle field carries the handle of the USB device that is issuing the notification. The Type and Subtype fields are set to 0 (Management) and 48 (DevNotificationReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The DevNotificationReq packet carries the fields listed in Table 55 after the management header.

Table 55—Device Notification Request fields

Width (bits)	Offset (DW:bit)	Description
4	3:0	**Reserved**
4	3:4	**Notification Type.** The field identifies the type of the device notification. Refer to Section 8.5.6 of [USB3.1].
56	3:8	**Notification Type Specific Data.** The field carries payload dependent on Notification Type. Refer to section 8.5.6 of [USB3.1].

6.3.51 Device Notification Response (DevNotificationResp)

The Device Notification Response (DevNotificationResp) packet is transmitted by the MA USB host to the target MA USB device in response to a DevNotificationReq packet. The Device Handle field carries the handle of the USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 49 (DevNotificationResp), respectively. The Status Code field indicates whether the request was successfully completed.

6.3.52 Endpoint Set Keep-Alive Request (EPSetKeepAliveReq)

The Endpoint Set Keep-Alive Request (EPSetKeepAliveReq) packet is transmitted by an MA USB device to the MA USB host to set the keep-alive duration for an endpoint in units of aTransferKeepAlive. This keep-alive determines the time that the host shall hold-off resending a TransferReq for which the device has already sent a TransferResp with status code TRANSFER_PENDING. The Device Handle field carries the handle of the USB device behind the MA USB device the request is targeting. The Type and Subtype fields are set to 0 (Management) and 50 (EPSetKeepAliveReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The EPSetKeepAliveReq packet carries the fields listed in Table 56 after the management header.

Table 56—Endpoint Set Keep-Alive Request fields

Width (bits)	Offset (DW:bit)	Description

16	3:0	**Keep-alive duration.** The duration of the keep-alive measured in units of aTransferKeepAlive. If this value is 0 the keep-alive is reset to the default value aDefaultKeepAliveDuration.
16	3:16	**EP Handle.** Indicates the endpoint the request is targeting.

6.3.53 Endpoint Set Keep-Alive Response (EPSetKeepAliveResp)

The Endpoint Set Keep-Alive Response (EPSetKeepAliveResp) packet is transmitted by the MA USB host to an MA USB device in response to an EPSetKeepAliveReq packet. The Device Handle field carries the handle of the USB device behind the MA USB device for which the response is being returned. The Type and Subtype fields are set to 0 (Management) and 51 (EPSetKeepAliveResp), respectively. The Status Code field indicates whether the request was successfully completed.

The EPSetKeepAliveResp packet carries the fields listed in Table 57 after the management header.

Table 57—Endpoint Set Keep-Alive Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Old Keep-alive Duration.** The previous value of the keep-alive measured in units of aTransferKeepAlive.
8	3:16	**Start Request ID.** The Request ID at which the new keep-alive duration will take effect.
8	3:24	**Reserved.**
16	4:0	**Stream ID.** Indicates the stream to which the response is related when the endpoint identified in the corresponding EPSetKeepAliveReq packet is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol; reserved otherwise.
16	4:16	**EPHandle.** Indicates the endpoint to which the response is related.

6.3.54 Get Port Bandwidth Request (GetPortBWReq)

The Get Port Bandwidth Request (GetPortBWReq) packet is transmitted by the MA USB host to a target MA USB hub to retrieve the percentage of the periodic bandwidth available on each port of a USB hub integrated into or downstream of the MA USB hub. The Device Handle field carries the handle of the USB hub the request is targeting.

The Type and Subtype fields are set to 0 (Management) and 52 (GetPortBWReq), respectively. The Status Code field is set to 0 (NO_ERROR).

Support of GetPortBWReq packet by the MA USB host and by the MA USB hub is optional. MA USB host shall not transmit a GetPortBWReq packet to MA USB devices. MA USB host shall not transmit another GetPortBWReq packet to an MA USB hub after it has received a GetPortBWResp packet with Status Code field set to NOT_SUPPORTED.

The GetPortBWReq packet carries the fields listed in Table 58 after the management header.

Table 58— Get USB Port Bandwidth Request fields

Width (bits)	Offset (DW:bit)	Description
4	3:0	**Speed.** Indicates the speed of the port(s) for which the available bandwidth shall be returned.

		Value Meaning 0 Low-Speed 1 Full-Speed 2 High-Speed 3 SuperSpeed 4 SuperSpeedPlus 5-15 Reserved
2	3:4	**Lane Speed Exponent (LSE).** Indicates the LSE of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.
4	3:6	**Lane Count.** Indicates the lane count of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.
2	3:10	**Link Protocol.** Indicates the link protocol of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.
4	3:12	**Reserved.**
16	3:16	**Lane Speed Mantissa (LSM).** Indicates the LSM of the USB device as defined in [USB 3.1] if the Speed field is set to SuperSpeed or SuperSpeedPlus. Reserved otherwise.

6.3.55 Get Port Bandwidth Response (GetPortBWResp)

The Get Port Bandwidth Response (GetPortBWResp) packet is transmitted by the target MA USB hub to the MA USB host in response to a GetPortBWReq packet to report the percentage of the periodic bandwidth available on each port of the USB hub the request is targeting. The Type and Subtype fields are set to 0 (Management) and 53 (GetPortBWResp), respectively. The Status Code field indicates whether the MA USB hub supports the feature, and if it does whether the request was successfully completed. If the MA USB hub does not support the GetPortBWReq packet it shall respond with a GetPortBWResp packet with Status Code field set to NOT_SUPPORTED. If the MA USB hub does not support the speed indicated by the value of the Sublink Speed Attribute ID field it shall respond with a GetPortBWResp packet with Status Code field set to UNSUCCESSFUL.

If the status code field is set to SUCCESS, then the GetPortBWResp packet carries the fields listed in Table 59 after the management header.

Table 59— Get USB Port Bandwidth Response fields

Width (bits)	Offset (DW:bit)	Description
8	3:0	**Number of Ports.** Indicates the number of ports on the USB hub for which bandwidth information is reported.
24	3:8	**Reserved**
Variable	4:0	**Port Available Bandwidth List.** List of port available bandwidth reports, concatenated in 8-bit increments, starting with the report of port number 1 in increasing order. Each 8-bit entry has a range of 0 to 100 and indicates the percentage of the periodic bandwidth available to the corresponding port.

6.3.56 Sleep Request (SleepReq)

The Sleep Request (SleepReq) packet is transmitted by an MA USB PAL to indicate the desire to move the session state to Session Inactive (Section 8.1.1.4). The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 54 (SleepReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The SleepReq packet carries the fields listed in Table 60 after the management header.

Table 60—SleepReq fields

Width (bits)	Offset (DW:bit)	Description
32	3:0	**Management Request Timeout.** Indicates the maximum time (in milliseconds) between the moment the remote MA USB PAL releases a management packet requiring a response to the management channel, and the moment it can expect the corresponding response packet through the management channel, while its session is in Session Inactive state.
32	4:0	**Data Request Timeout.** Indicates the maximum time (in milliseconds) between the moment the remote MA USB PAL releases a control or data packet requiring a response to the assigned data channel, and the moment it can expect the corresponding response packet through the assigned data channel, while its session is in Session Inactive state.

NOTE — Management Request Timeout and Data Request Timeout values are respectively equivalent to aManagementRequestTimeout and aTransferTimeout protocol constants when session is in Session Active state.

6.3.57 Sleep Response (SleepResp)

The Sleep Response (SleepResp) packet is transmitted in response to a SleepReq packet to acknowledge the desire to move the session state to Session Inactive (Section 8.1.1.4), and carries the same fields as the SleepReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 55 (SleepResp), respectively. The Status Code value indicates whether the request to move the session to Session Inactive is granted. A Status Code value of 0 (NO_ERROR) indicates that the MA USB PALs at the two ends of an MA link can coordinate activities at the MA link level to enter low-power mode, and a Status Code value of REQUEST_DENIED indicates that the MA link cannot enter low-power mode.

When the Status Code field is set to 0 (NO_ERROR), the Management Request Timeout and Data Request Timeout values are less than or equal to the corresponding values in the SleepReq packet and indicate the management and control or data packet exchange timeout values when the exchange is initiated by the recipient of the SleepResp packet. When the Status Code field is set to REQUEST_DENIED, the Management Request Timeout and Data Request Timeout values are less than or equal to the corresponding values in the SleepReq packet and indicate the maximum timeout values acceptable to the recipient of the SleepReq packet. Any of the timeout fields may be set to 0 to indicate that the request to move the session state to Session Inactive is being denied without specifying maximum acceptable values.

NOTE — For example, a SleepReq packet transmitted by an MA USB device with Management Request Timeout value of 5,000 indicates that should the MA USB host accept the request to move its session state to Session Inactive, it has to exercise a timeout value of 5 seconds when initiating a management packet exchange that targets the MA USB device. A SleepResp packet transmitted by the MA USB host with Status Code value of 0 (NO_ERROR) and Management Request Timeout value of 4,000 indicates that the MA USB host will observe the timeout value of 5 seconds when it initiates a management packet exchange that targets the MA USB device; furthermore, it indicates that the MA USB host has moved its own session state to Session Inactive, and the MA

USB device has to exercise a timeout value of 4 seconds when initiating a management packet exchange that targets the MA USB host. A SleepResp packet transmitted by the MA USB host with Status Code value of REQUEST_DENIED and a Management Request Timeout value of 4,000 indicates that the MA USB host has not moved its session state to Session Inactive, but may accept the request to move its session state to Session Inactive if the Management Request Timeout value in a future SleepReq packet does not exceed 4,000 milliseconds.

6.3.58 Wake Request (WakeReq)

The Wake Request (WakeReq) packet is transmitted by an MA USB PAL to indicate the desire to move the session state to Session Active (Section 8.1.1.3). The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 56 (WakeReq), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.59 Wake Response (WakeResp)

The Wake Response (WakeResp) packet is transmitted in response to a WakeReq packet to acknowledge the desire to move the session state to Session Active (Section 8.1.1.3). The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 57 (WakeResp), respectively. The Status Code field is set to 0 (NO_ERROR).

6.3.60 Vendor Specific Request (VendorSpecificReq)

The Vendor Specific Request (VendorSpecificReq) packet is transmitted by the MA USB host or the MA USB device to request a vendor specific operation. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 62 (VendorSpecificReq), respectively. The Status Code field is set to 0 (NO_ERROR).

The VendorSpecificReq packet carries the fields listed in Table 61 after the management header.

Table 61—Vendor Specific Request fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Vendor Identifier.** The value assigned by the USB-IF to the organization that has defined the content of the particular vendor-specific request.
16	3:16	**Reserved.**

6.3.61 Vendor Specific Response (VendorSpecificResp)

The Vendor Specific Response (VendorSpecificResp) packet is transmitted by the MA USB host or the MA USB device in response to a VendorSpecificReq packet. The Device Handle field is reserved. The Type and Subtype fields are set to 0 (Management) and 63 (VendorSpecificResp), respectively. The Status Code field is set to SUCCESS.

The VendorSpecificResp packet carries the fields listed in Table 62 after the management header.

Table 62—Vendor Specific Response fields

Width (bits)	Offset (DW:bit)	Description
16	3:0	**Vendor Identifier.** The value assigned by the USB-IF to the organization that has defined the content of the particular vendor-specific request that triggered the response.
16	3:16	**Reserved.**

6.4 Control packets

MA USB control packets share the control header shown in Figure 38. All control header fields are defined in Section 6.2.

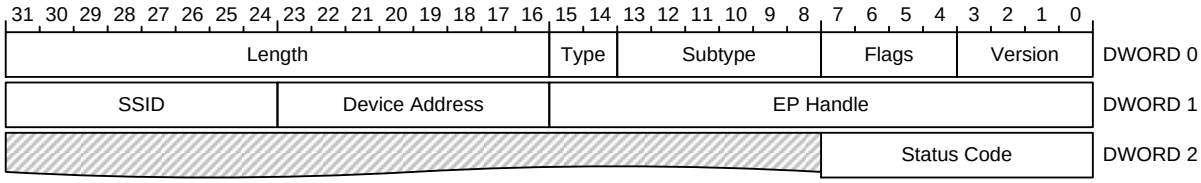


Figure 38—Common header for MA USB control packets

6.4.1 Transfer Setup Request (TransferSetupReq)

The Transfer Setup Request (TransferSetupReq) packet initiates the setup phase of an l-managed OUT transfer. The Type and Subtype fields are set to 1 (Control) and 0 (TransferSetupReq), respectively. The Status Code field is set to 0 (NO_ERROR).

TransferSetupReq packet carries the fields listed in Table 63 after the control header.

Table 63—Transfer Setup Request fields

Width (bits)	Offset (DW:bit)	Description
8	2:8	**Link Type**. Identifies the link type as following, Value — Description 0 — Reserved 1 — IEEE 802.11 link (802.11 mode) 2-255 — Reserved
Variable	2:16	**Connection ID**. Identifies the flow-controlled link-level connection associated with the l-managed OUT transfer. The size of the field depends on the link type as specified by the Link Type field. For an IEEE 802.11 link (Link Type = 1), the Connection ID field is 8 bits wide, with the lower-order 4 bits set to the Traffic Identifier (TID) on which a dedicated traffic flow is established or will be established to serve the l-managed OUT transfer. The higher-order 4 bits of the field are reserved in this case.

6.4.2 Transfer Setup Response (TransferSetupResp)

The Transfer Setup Response (TransferSetupResp) packet indicates the success or failure of the set up phase for an l-managed OUT transfer, and has the same format as the TransferSetupReq packet (Section 6.4.1). The Type and Subtype fields are set to 1 (Control) and 1 (TransferSetupResp), respectively. The Status Code field indicates the success or failure of the l-managed OUT transfer set up phase.

6.4.3 Transfer Tear Down Confirmation (TransferTearDownConf)

The Transfer Tear Down Confirmation (TransferTearDownConf) packet indicates the end of lifetime of an l-managed OUT transfer, and has the same format as the TransferSetupReq packet (Section 6.4.1). The Type and Subtype fields are set to 1 (Control) and 2 (TransferTearDownConf), respectively. The Status Code field is set to 0 (NO_ERROR).

6.5 Data packets

Non-isochronous data packets share the header shown in Figure 39. Isochronous data packets share the header shown in Figure 40. Header fields not defined in Section 6.2 are defined in Section 6.5.1. Data packet subtypes are defined in Sections 6.5.2 through 6.5.6.

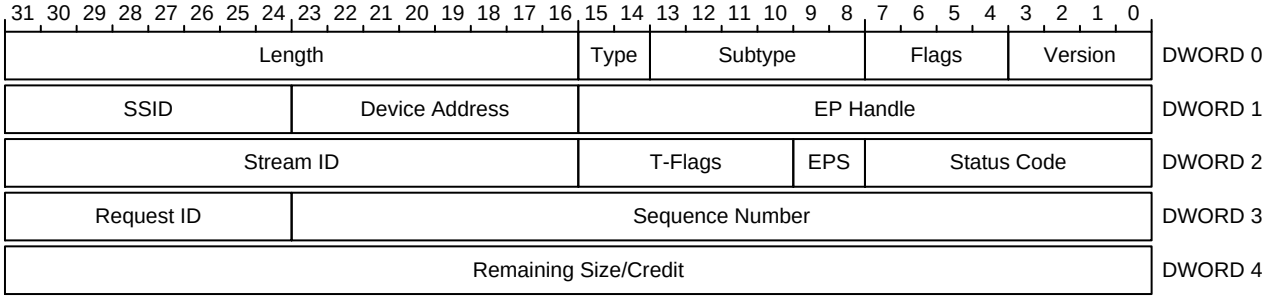


Figure 39—Common header for MA USB non-isochronus data packets

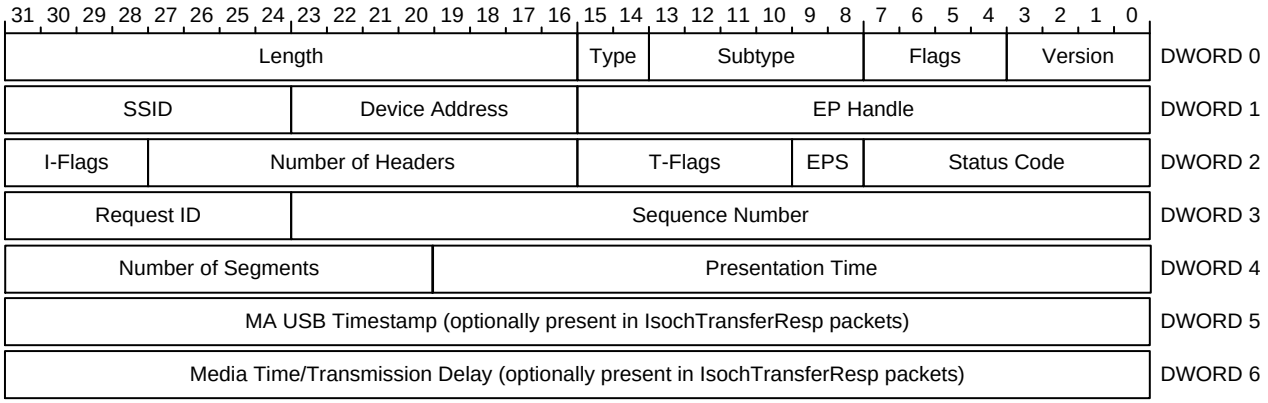


Figure 40—Common header for MA USB isochronus data packets

6.5.1 Common data header fields

6.5.1.1 EPS

The 2-bit EPS (EP Status) field (offset 2:8) indicates the status of the EP handle contained in the data packets transmitted by the MA USB device. It assumes one of the values listed in Table 64. The EPS field is reserved in data packets transmitted by the MA USB host.

Table 64—EPS field values

Value	Description
00b	**Unassigned.** The EP handle contained in the packet is not assigned to any endpoint.
01b	**Active.** The EP handle contained in the packet is assigned and is in active state, i.e., the corresponding endpoint is able to move data and complete transfer requests successfully.
10b	**Inactive.** The EP handle contained in the packet is assigned but is in an inactive state, i.e., the corresponding endpoint is not able to move data and complete transfer requests successfully.
11b	**Halted.** The EP handle contained in the packet in Halted state, i.e., the corresponding endpoint has encountered a USB halt condition.

6.5.1.2 T-Flags

The 6-bit T-Flags field (offset 2:10), shown in Figure 41, contains the bit fields listed in Table 65.

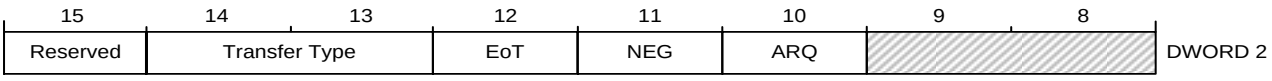


Figure 41—T-Flags field

Table 65—T-Flags subfields

Width (bits)	Offset (DW:bit)	Description
1	2:10	**ARQ (Acknowledgment Request).** Set to 1 to indicate an acknowledgment request and 0 otherwise. Reserved for all data packet subtypes other than Transfer Request and Transfer Response.
1	2:11	**NEG (Negative Credit).** A logical extension of the 32-bit Credit field to form a 33-bit signed integer using two's complement format. If set to 0, the 32-bit credit field is carrying a positive number in the range 0 to $2^{32} - 1$. If set to 1, the 33-bit number made of concatenation of this bit and the Credit field is carrying a negative number in the range -1 to -2^{32}. This field may be set to 1 only if the device indicates support for elastic buffer capability in the CapResp packet it sends to the MA USB host.
1	2:12	**EoT (End of Transfer).** In an OUT transfer, set to 1 to indicate the completion of the transfer, i.e., delivery of the data to the endpoint. In an IN transfer, set to 1 to indicate the last TransferResp packet related to the transfer.
2	2:13	**Transfer Type.** Indicates the transfer type of the data packet, Value — Meaning 00b — Control 01b — Isochronous 10b — Bulk 11b — Interrupt NOTE — In a Control TransferReq packet with the Sequence Number field set to 0, the first 2 DWORDs of the payload in the packet are control transfer setup data.
1	2:15	**Reserved.**

6.5.1.3 Stream ID (non-isochronous data packets)

The 16-bit Stream ID field (offset 2:16) identifies the target stream when the target endpoint is an Enhanced SuperSpeed bulk endpoint that supports the Enhanced SuperSpeed Stream Protocol. It is reserved in all other cases.

6.5.1.4 Sequence Number

The 24-bit Sequence Number field (offset 3:0) is normally used to identify the successive data packets that carry the payload for a given MA USB transfer, i.e., successive Transfer Request or Isochronous Transfer Request packets belonging to an OUT transfer, and successive Transfer Response or Isochronous Transfer Response packets belonging to an IN transfer. The field is set to 0 for the first data bearing packet, and is incremented by one for each subsequent data bearing packet, with rollover to 0 after aMaxSequenceNumber. Additional (subtype-specific) usages of the field are described together with each data packet subtype definition.

6.5.1.5 Request ID

The 8-bit Request ID field (offset 3:24) carries the identifier assigned by the MA USB host to an MA USB transfer to identify it among all outstanding transfers targeting the same endpoint. All data packets (request or response) belonging to the same MA USB transfer carry the same Request ID.

6.5.1.6 Remaining Size/Credit (non-isochronous data packets)

The 32-bit Remaining Size/Credit field (offset 3:0) carries the number of bytes remaining to complete a transfer (assuming the current packet is successfully received), or the number of bytes that the receiver can accept. The exact function of this field depends on the transfer direction and data packet subtype.

6.5.1.7 Number of Headers (isochronous data packets)

When sent in an MA USB isochronous transfer request packet for an OUT transfer or an MA USB isochronous transfer response for an IN transfer, the 12-bit Number of Headers field (offset 2:16) carries the number of isochronous headers that are present in the packet. When sent in an MA USB isochronous transfer request packet for an IN transfer, the Number of Headers field carries the number of IRS headers that are present in the packet. The Number of Headers field is reserved otherwise.

6.5.1.8 I-Flags (isochronous data packets)

The 4-bit I-Flags field (offset 2:28) is formatted as shown in Figure 42, with the subfields defined in Table 66.

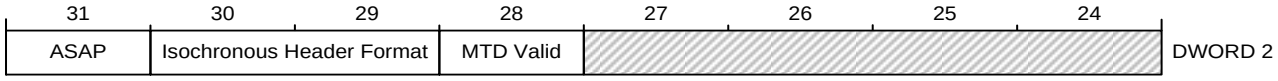


Figure 42—I-Flags field

Table 66—I-Flags subfields

Width (bits)	Offset (DW:bit)	Description
1	2:28	**MTD Valid.** Set to 1 if the Media Time/Transmission Delay field in the packet carries a valid value and set to 0 otherwise. NOTE — The MTD Valid subfield can be set to 0 both at transmission (e.g., before releasing the MA USB packet to the lower network layer to indicate that the field has not been uninitialized) and reception (e.g., after receiving the MA USB packet from the lower network layer and detecting inaccuracy in the Media Time/Transmission Delay field.).
2	2:29	**Isochronous Header Format.** Indicates the format of all isochronous headers or isochronous read size (IRS) blocks in the packet. When applied to isochronous headers, valid values for this subfield are 0 (short format), 1 (standard format) and 2 (long format). When applied to isochronous read size (IRS) blocks, valid values are 0 (standard format) and 1 (long format).
1	2:31	**ASAP**. Set to 1 to request immediate (as soon as possible) delivery to or from the target isochronous endpoint, or to 0 to indicate delivery at the time specified by the Presentation Time field in an isochronous transfer request packet. This field is reserved for isochronous transfer response packets.

6.5.1.9 Presentation Time (isochronous data packets)

The 20-bit Presentation Time field (offset 4:0) carries the intended time of delivery of the first segment of transfer, or actual time of delivery of the first isochronous segment in the packet to or from the target

isochronous endpoint. The time of delivery for each subsequent segment (if any) embedded in the packet is implicitly defined according to the Service Interval associated with the target isochronous endpoint. The field is reserved in isochronous transfer request packets that indicate as soon as possible (ASAP) delivery. The Presentation Time field has the format shown in Figure 43, with the subfields defined in Table 67.

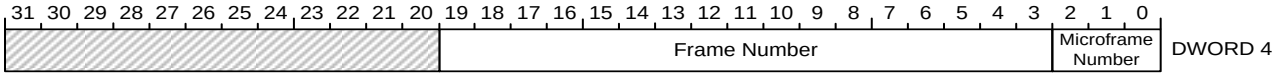


Figure 43—Presentation Time field format

Table 67—Presentation Time subfields

Width (bits)	Offset (DW:bit)	Description
3	4:0	**Microframe Number**. Indicates the modulo-8 number of the USB microframe (the 125 µs time base) at the time of interest. Set to 0 when not supported.
17	4:3	**Frame Number**. Indicates the USB frame number, i.e., the 1 millisecond time base established by the MA USB host. Frame number is maintained modulo 2^{17} or a smaller number, but not smaller than 2^{11}.

6.5.1.10 Number of Segments (isochronous data packets)

When sent in an MA USB isochronous transfer request packet, the 12-bit Number of Segments field (offset 4:20) carries the total number of isochronous segments that are being requested or transmitted for the MA USB transfer. When sent in an MA USB isochronous transfer response packet, the field carries the number of complete segments that have been read from or delivered to the target isochronous endpoint in the MA USB device at the time the response packet was generated.

NOTE — The value of this field is generally larger than the value of the Number of Headers field (Section 6.5.1.7), which carries the number of isochronous segments (or fragments thereof) present in the packet.

6.5.1.11 MA USB Timestamp (isochronous data packets)

The 32-bit MA USB timestamp field carries a sample of the MA USB Global Time (Section 6.6.1). It is present in all IsochTransferReq packets, and optionally present in IsochTransferResp packets.

NOTE — When transmitted in an IsochTransferResp packet, the MA USB Timestamp field represents a reading of the MA USB Global Time that the transmitting MA USB device is maintaining. The MA USB host can monitor the device synchronization status for error recovery and diagnostic purposes.

6.5.1.12 Media Time/Transmission Delay (isochronous data packets)

Depending on the latency compensation method applied to an MA link, the 32-bit Media Time/Transmission Delay field carries the synchronized Media Time of the MA link at the time MGT is captured into the MA USB Timestamp field, or the time elapsed (in nanoseconds) from the moment MGT is captured into the MA USB Timestamp field to the moment the first bit of the Media Time/Transmission Delay field is released to the physical medium. This field is present in all IsochTransferReq packets, and optionally present in IsochTransferResp packets.

NOTE — The Media Time format and its adaptation into the 32-bit Media Time field depend on the Media Clock and are media-specific.

NOTE — A network packet may contain more data units than a single MA USB packet, e.g., other MA USB packets, or packets belonging to other protocols. The Transmission Delay field captures an interval that ends at the moment the first bit of the field itself is released to the physical medium, and therefore finds dependency on the relative offset of the field into the network packet that carries the field.

Media Time and Transmission Delay values have accuracy requirements. For Media Time, there is an additional sampling accuracy requirement applicable at transmission. See Section 6.6 for details.

6.5.2 Transfer Request (TransferReq)

The Transfer Request (TransferReq) packet is sent by the MA USB host to initiate an MA USB non-isochronous IN or OUT transfer, and to carry the transfer payload for OUT transfers. It carries the data header shown in Figure 39. When initiating a control transfer the packet carries the control set up data. When initiating a bulk or interrupt IN transfer the packet carries no payload.

The Type and Subtype fields are set to 2 (Data) and 0 (TransferReq), respectively. The Status Code field is set to 0 (NO_ERROR). The EPS field is reserved. The Request ID and Sequence Number fields are set according to Section 5.4 for IN transfers and Section 5.5 for OUT transfers.

The Remaining Size/Credit field carries the remaining number of bytes to complete an OUT transfer or the number of bytes that the device can transmit in the case of an IN transfer.

6.5.3 Transfer Response (TransferResp)

The Transfer Response (TransferResp) packet is sent by an MA USB device to carry the payload for a non-isochronous IN transfer, or to provide feedback (in the form of receive confirmation, endpoint status and credit) for a non-isochronous OUT transfer. It carries the data header shown in Figure 39.

The Type and Subtype fields are set to 2 (Data) and 1 (TransferResp), respectively. The Status Code field is set to one of the values listed in Table 6 to indicate successful or failed reception of the payload belonging to one or more Transfer Request packets. The EPS field is set to one of the values listed in Table 64 to indicate the status of the EP handle contained in the packet.

The Sequence Number field is set according to Section 6.5.1.4 for IN transfers.

For OUT transfers, the field is used to acknowledge or provide feedback on one or more Transfer Request packets. See Section 5.5 for operation details.

The Remaining Size/Credit field is used differently depending on the transfer type. For IN transfers, the field carries the remaining number of bytes to complete the transfer. For OUT transfers, the field carries the number of bytes the MA USB device can accept throughout the OUT transfer.

A TransferResp packet of an IN transfer that does not carry any payload is called a null transfer. A null transfer may carry an EOT field set to 1 and it may have the Sequence Number field set to aInvalidSequenceNumber.

6.5.4 Transfer Acknowledgement (TransferAck)

The Transfer Acknowledgement (TransferAck) packet is used by the MA USB host to acknowledge TransferResp packets sent by an MA USB device. It carries the data header shown in Figure 39.

The Type and Subtype fields are set to 2 (Data) and 2 (TransferAck), respectively. The Status Code field is set to one of the values listed in Table 6. The Status Code field shall be set to TRANSFER_PENDING if the TransferAck is sent in response to a TransferResp packet that had the ARQ bit set to 1 and a Status Code value of TRANSFER_PENDING. This enables a device to determine that a TransferResp packet with status TRANSFER_PENDING has been received by the host. The EPS field is reserved.

The Sequence Number field is used to acknowledge or provide feedback on one or more Transfer Response packets.

The Remaining Size/Credit field is reserved.

6.5.5 Isochronous Transfer Request (IsochTransferReq)

The Isochronous Transfer Request (IsochTransferReq) packet initiates an MA USB isochronous IN or OUT transfer. It carries the data header shown in Figure 40. When initiating an isochronous IN transfer, the packet carries no payload. When initiating an isochronous OUT transfer, the packet additionally carries the first segment of the OUT transfer payload.

The Type and Subtype fields are set to 2 (Data) and 3 (IsochTransferReq), respectively. The Status Code field is set to 0 (NO_ERROR). The EPS field is reserved.

The Sequence Number field is set to aInvalidSequenceNumber for IN transfers. It is set according to Section 6.5.1.4 for OUT transfers.

6.5.6 Isochronous Transfer Response (IsochTransferResp)

The Isochronous Transfer Response (IsochTransferResp) packet is used to carry isochronous payload for IN transfers, or provides feedback for isochronous OUT transfers. It carries the data header shown in Figure 40.

The Type and Subtype fields are set to 2 (Data) and 4 (IsochTransferReq), respectively.

6.6 Clock synchronization

6.6.1 Clock model

The USB clock model is defined in Section 5.12.2 of [USB 2.0]. The MA USB clock model maintains the Sample Clock and Service Clock defined in [USB 2.0], but replaces the Bus Clock with an MA USB Global Clock that is maintained by the MA USB host and made available to all MA USB devices within an MA USB Service Set. The MA USB Global Clock has the same frequency as the Enhanced SuperSpeed USB Bus Clock (8 KHz). In addition, each MA USB hub is expected to synchronize the Bus Clock for each of its wired USB segments downstream to the MA USB Global Clock. The MA USB clock model is shown in Figure 44.

NOTE — This may require that, on full-speed and high-speed bus segments, the first bit of the SOF is issued onto the USB bus when the delta field of the MGT is zero. The frame counter in the PID must match bits 26:16 of the MGT (the part of the MGT that corresponds to the traditional "frame number"). For SuperSpeed and SuperSpeed Plus segments, bits 13:0 of the ITS in the ITP, when issued from the root port of the segment, must match bits 26:13 of the MGT (the part of the MGT that corresponds to the SuperSpeed bus interval counter), and bits 26:14 in the ITS of the ITP must be set appropriately relative to the bus boundary established by the MGT when the MGT Delta field is zero.

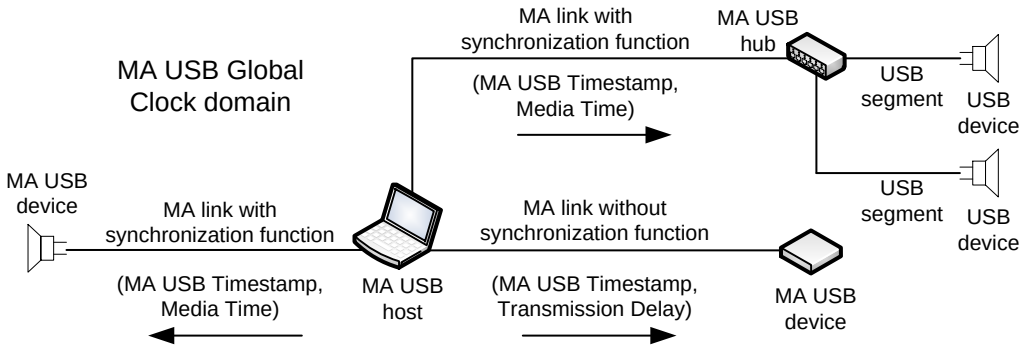


Figure 44—MA USB clock model and distribution

Each sample of the MA USB Global Clock designates the MA USB Global Time (MGT) at the sampling instant. The MGT format is shown in Figure 45, with the subfields defined in Table 68.

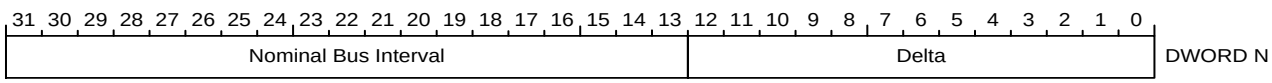


Figure 45—MA USB Global Time (MGT) format

Table 68—MA USB Global Time (MGT) subfields

Width (bits)	Offset (DW:bit)	Description
13	N:0	**Delta**. Indicates the value of Delta field in Isochronous Timestamp Packet as defined in [USB 3.1]. Delta is maintained in units of 8 nominal HS bit times, i.e., 8/480 µs (nominal).
19	N:13	**Nominal Bus Interval**. Indicates the nominal USB microframe number, i.e., the 125 µs time base. Nominal Bus Interval is maintained modulo 2^{19}.

6.6.2 Synchronization

Clock synchronization in MA USB is achieved by the MA USB host transmitting samples of the MA USB Global Clock through timestamp fields of certain management and data packets. Specifically, the MA USB host carries the MA USB Global Time in the MA USB Timestamp field of SynchReq (Section 6.3.40) and IsochTransferReq (Section 6.5.5) packets.

Because of the loose coupling between MA USB protocol and the PHY layer of each MA link, this specification defines mechanisms to enable the receiver of an MA USB timestamp to compensate for the variable latency MA USB packets experience across an MA link.

- Compensation through a synchronized "Media Time": Some MA links provide access to a synchronized "Media Clock", possibly implemented for networking purposes outside MA USB. By providing a reading of the Media Clock (referred to as Media Time) at the moment the MA USB Timestamp field in an outgoing MA USB packet is initialized, an additional point of reference is made available to the receiver of the MA USB timestamp to adjust the received timestamp for additional latencies. The Media Clock of an MA link shall have an accuracy of ±aMaxMediaTimeError, i.e., the Media Time at the two ends of an MA link shall not be different by more than 2×aMaxMediaTimeError. Media Time is carried in the Media Time field of the MA USB packet containing an MA USB timestamp. Sampling of the Media Clock shall have an accuracy of ±aMaxMediaTimeSamplingError, i.e., the value inserted into the Media Time field of an MA USB packet shall not be more than ±aMaxMediaTimeSamplingError different from the value of the Media Time at the moment of insertion.

  NOTE — The Media Time format depends on the Media Clock and is media-specific.

  NOTE — The synchronization function of a Media Clock may be required to compensate for the MA link average latency (path delay) to meet the Media Clock accuracy requirement.

  NOTE — A received MA USB timestamp carries an error no smaller than the difference between the local Media Time at the moment the MA USB timestamp is captured and the Media Time transmitted in the MA USB packet. Implementations need to consider additional complexities when applying an error term to the complex (frame, microframe) MGT format.

- Compensation through an estimate of the "Transmission Delay": Alternatively, the MA USB host may choose to augment the MA USB timestamps with an estimate of the delay the containing MA USB packet experiences from the moment the MA USB timestamp is initialized to the moment the packet is released to the physical medium. The delay estimate, expressed in nanoseconds, is carried in the Transmission Delay field of the MA USB packet containing the timestamp, and shall have an accuracy of ±aMaxTransmissionDelayError.

NOTE — As the name implies, the transmission delay estimate does not cover additional latencies incurred at the receiver; compensating for latencies local to the receiver is a matter of receiver implementation and outside the scope of this specification.

The compensation method for each MA link is decided by the MA USB host and does not change through the lifetime of an MA link. The MA USB host indicates access to a synchronized Media Time by setting the Media Time Available field in any CapReq packet it transmits to a target MA USB device. The target MA USB device also indicates access to the same synchronized Media Time by setting the Media Time Available bit in any CapResp packet it transmits to the MA USB host. The latency compensation method applied to the MA link in both directions shall be based on Media Time if both MA USB host and device have indicated Media Time availability, and shall be based on Transmission Delay otherwise.

NOTE — The type and characteristics (e.g., accuracy) of the Media Clock are assumed to be known to both MA USB host and device by virtue of the MA Link type and the discovery context, i.e., MA USB host and device refer to the same Media Clock when announcing the Clock availability.

The MA USB host distributes the MGT through a combination of broadcast and unicast SynchReq (Section 6.3.40) packets as well as IsochTransferReq (Section 6.5.5) packets. The MA USB host shall not transmit unicast SynchReq packets to an MA USB device that has not requested to receive MA USB timestamps.

For monitoring and diagnostic purposes, the MA USB host may request an MA USB device that has requested to receive MA USB timestamps to share its local instance of the MGT through an appropriate response packet. An MA USB device that has requested to receive MA USB timestamps

- shall respond to a unicast or broadcast SynchReq packet that has a Response Required bit set to 1 with a SynchResp packet that includes a local reading of the MGT;
- may insert a local reading of the MGT in any of the IsochTransferResp (Section 6.5.6) packets it transmits to the MA USB host.

The MA USB host shall distribute the MGT to each MA USB device in Session Active state (Section 8.1.1.3) that has at least one configured isochronous endpoint. MGT shall be delivered at least once every aMinSynchFrequencyActive if the target MA USB device has at least one active isochronous endpoint (i.e., a configured isochronous endpoint that has not been idle for more than aMinSynchFrequencyIdle), and at least every aMinSynchFrequencyIdle otherwise.

NOTE — The MA USB host may distribute the MGT to an MA USB hub prior to the enumeration of the integrated USB hub and following the transition of the integrated USB hub out of a low power state.

NOTE — The clock synchronization mechanism for MA USB described in this specification assumes a hardware implementation.

7 MA USB device framework

7.1 Device states

The device states match the state of the session between the MA USB host and the MA USB device as defined in Section 8.1.

7.2 EP handle states

An endpoint handle is either not assigned or assigned. An assigned endpoint handle can be in Active, Inactive, or Halt State. The state diagram of an endpoint handle is depicted in Figure 46 and the states are described below.

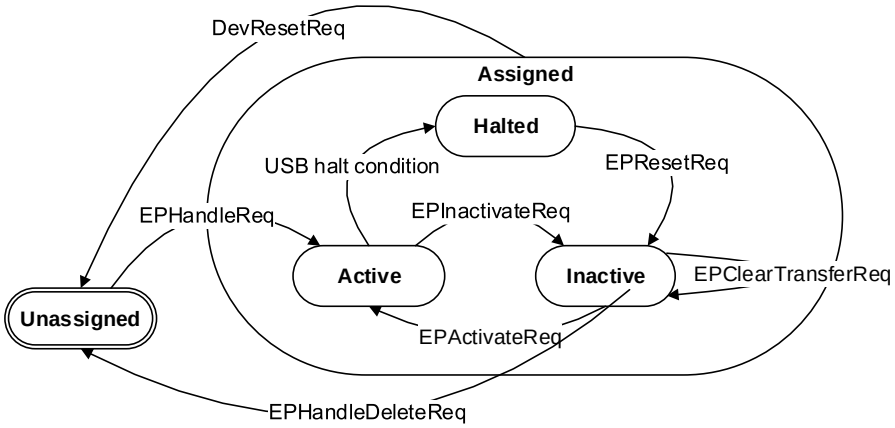


Figure 46—EP handle state diagram

7.2.1 Active state

This state is entered on any of the following events:

- An endpoint handle is first assigned by the MA USB device in response to an EPHandleReq packet (Section 6.3.6) received from the MA USB host.
- The MA USB device receives an EPActivateReq packet (Section 6.3.8) from the MA USB host for an endpoint handle in Inactive State.

While in this state the endpoint is capable of USB transactions and the MA USB host may transmit TransferReq packets targeting the endpoint.

This state exits when:

- The USB transactions with the associated endpoint encounter a USB halt condition.
- The MA USB device receives an EPInactivateReq packet (Section 6.3.10) from the MA USB host.
- The MA USB device receives a DevResetReq packet (Section 6.3.18).

7.2.2 Halted state

This state is entered when the USB transactions with an endpoint with a handle in Active State encounter a USB halt condition (i.e., transmission error occurs on the bus or STALL handshake is returned from the endpoint).

While in Halted State the endpoint is not capable of USB transactions. Any TransferReq packet received targeting the EP shall be buffered and acknowledged with a TransferResp packet with status code set to INVALID_EP_HANDLE_STATE.

This state exits when:

- The MA USB device receives an EPResetReq packet (Section 6.3.12) from the MA USB host.
- The MA USB device receives a DevResetReq packet (Section 6.3.18).

NOTE — rules for entering and exiting the Halted state for endpoints are as defined in USB specification; specifically as defined in Section 5.6.5 of [USB 2.0] for isochronous endpoints and Section 5.5.5 of [USB 2.0] for control endpoints.

7.2.3 Inactive state

This state is entered,

- From Active State, if the MA USB device receives an EPInactivateReq packet (Section 6.3.10) from the MA USB host.
- From Halted State, if the MA USB device receives an EPResetReq packet (Section 6.3.12) from the MA USB host.

While in Inactive State the endpoint is not capable of USB transactions. Any TransferReq packet received targeting the EP shall be buffered and acknowledged with a TransferResp packet with status code set to INVALID_EP_HANDLE_STATE. The transition to Inactive state does not clear outstanding transfers. The MA USB host may transmit ClearTransfersReq (Section 6.3.14) or CancelTransferReq (Section 6.3.42) packets to request clearing (cancelling) outstanding transfers on an endpoint with an endpoint handle in Inactive State. The MA USB device is not required to transmit TransferResp packets for the cleared pending requests.

The MA USB host may transmit an EPHandleDeleteReq packet (Section 6.3.16) to transition the EP Handle to Not Assigned state; the MA USB device shall cancel any pending requests for the EP Handle. The MA USB device is not required to transmit TransferResp packets for the cleared pending requests.

NOTE — Status Code value of INVALID_EP_HANDLE returned in the EPHandleDeleteResp packet indicates successful completion of the EPHandleDeleteReq packet.

This state exits when:

- The MA USB device receives an EPActivateReq packet (Section 6.3.8) from the MA USB host.
- The MA USB device receives an EPHandleDeleteReq packet (Section 6.3.16) from the MA USB host.

  NOTE — The MA USB host will issue this command when modifying established device configurations (e.g., in preparation to set up for an alternate USB interface or configuration) or de-configuring a USB device which usually occurs when a USB device is disconnected.
- The MA USB device receives a DevResetReq packet (Section 6.3.18).

7.2.4 Unassigned state

This state is entered

- On power-on-reset.
- From any state if the MA USB device receives a DevResetReq packet (Section 6.3.18) from the MA USB host.
- From Inactive State if the MA USB device receives an EPHandleDeleteReq packet (Section 6.3.16) from the MA USB host.

- If the MA USB device receives a USBDevDisconnectReq packet (Section 6.3.26) from the MA USB host targeting a device to which the EP handle belongs.
- If the MA USB device receives a DevDisconnectReq packet (Section 6.3.36) from the MA USB host.

This state exits when the MA USB device receives an EPHandleReq packet (Section 6.3.6) from the MA USB host.

If the MA USB PAL receives a TransferReq packet targeting an EP handle in the Unassigned state, then the MA USB device shall respond with a TransferResp packet with status code set to INVALID_EP_HANDLE.

7.3 Device set up

7.3.1 Discovery mechanism

MA USB host and MA USB device discover one another using device discovery mechanism provided by lower layers.

7.3.2 USB device enumeration

This specification defines the over the air MA USB management packets related to USB device enumeration. These packets are normative. The order of these management packets, however, may be different for different implementations as the sequence of the Management actions in the USB system of the MA USB host may be different for different USB host systems and implementations. These actions trigger the MA USB host to transmit MA USB management packets. Table 69 lists the USB system management actions, events, and entities with a general description as a reference.

Table 69—USB system management actions, events, and entities

Action	Description
Open USB device	In response to a USB port connect event, the USB host core system software will create a new context (sometimes called a device object) for the device that is connected downstream of the port. This device object will be populated with information extracted from the device (descriptors etc.) and is typically used by the host system to store the state of the device.
USB device removal	In response to a USB port disconnect event, the USB host core system software will initiate clearing all the states related to the USB device.
Open EP	When a USB Client device driver issues a SetConfiguration request to USB system software, the USB host core system software will create a new endpoint context (sometimes called an endpoint object) for each endpoint in the configuration referenced by the SetConfiguration request. The endpoint object is typically used by the host system software to store the state of the endpoint.

Modify EP0	USB Full-speed devices are allowed to implement a default control endpoint maximum packet size of 8, 16, 32 or 64 bytes. The device reports the size of the implementation in its device descriptor. An operational nuance is that the host system must read the device descriptor before knowing the size of the endpoint. Many host implementations make an assumption about the size of the control endpoint, read the device descriptor and then modify the endpoint object to set the new packet size.
Address device	A USB host core system manages the state of an attached device based on the canonical device state diagram in Chapter 9 of the Universal Serial Bus Specification, revision 1.0. A USB port reset drives the device attached to that port to device address zero (called the Default device state). The host system will then initiate an action to assign a nonzero address to the device and transition it to the Address state.
Evaluate device	There are situations where additional device parameters are determined by policies of the host system software or are obtained in the later phases of enumeration. This event represents the projection of host system policies that need to be recorded in the device object for proper operation on the bus.
Configure device	Before a device can be used (other than the default control endpoint), it must be explicitly configured. In typical USB host systems, the client device driver requests USB system software to set a specific configuration on the device. The USB host system software will allocate endpoint objects for the endpoints described in the selected configuration and will issue a SetConfiguration request to the device.
Port manipulation	USB core system software provides standard methods for upper level entities (like the hub driver) to manipulate fields in downstream facing port registers (e.g. read status, write control bits). Port manipulation represents the use of any of these methods activated by USB host system software.

1

Event	**Description**
Port status change	USB downstream facing ports in Hosts and Hubs have associated port registers in which are contained control, status and status change bits. Whenever a status bit in a port register changes as a result of a device interaction, the event is recorded in a corresponding status change bit. A status change bit usually results in a notification being delivered to the USB host system software that a status change bit is set in a particular port.

2

Entity	**Description**
MA USB host USB core system	This represents the USB system software on the host platform that provides enumeration, device and endpoint management, and transfer services to USB client drivers.

3

Integrated USB device enumeration follows successful establishment of MA USB device connection with the MA USB host, i.e., following completion of capability exchange.

Example generalized sequences of the integrated USB device enumeration procedure is shown in Figure 47.

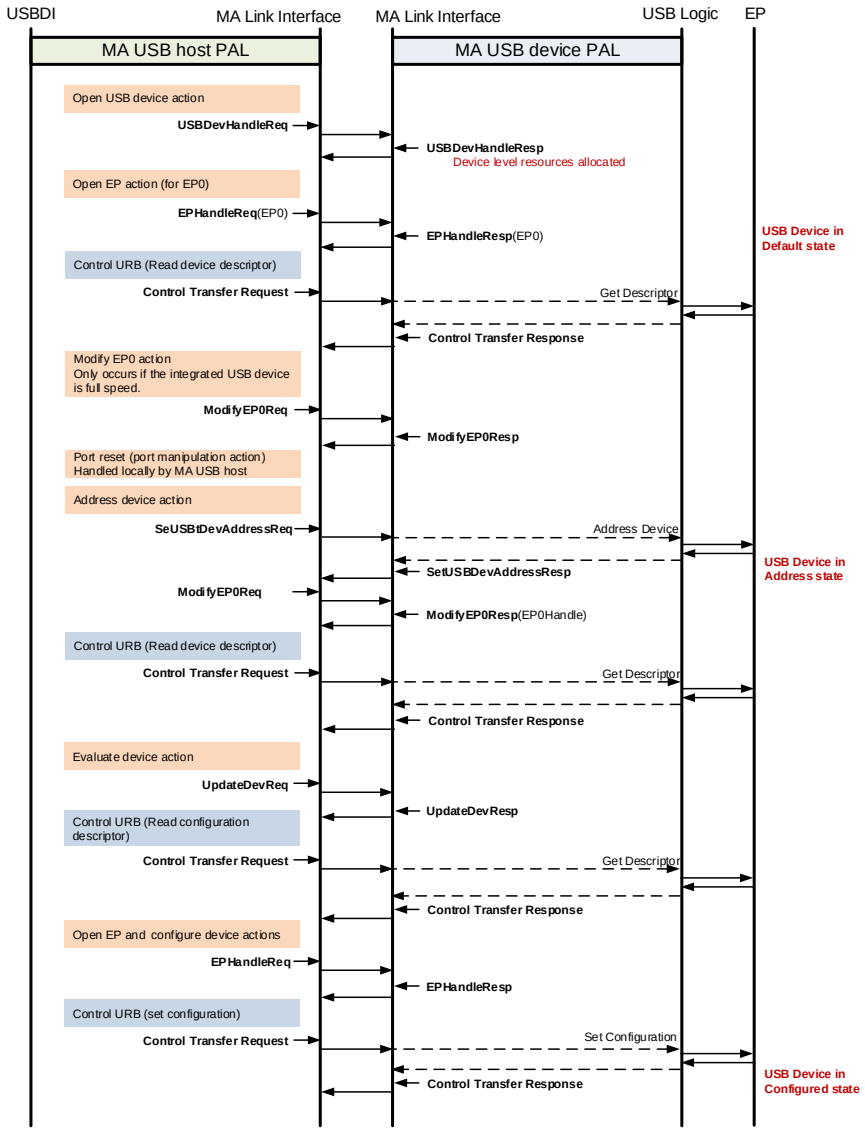


Figure 47—Enumeration of an integrated USB device

Following the MA USB session establishment, the MA USB host PAL emulates a port status change event equivalent of connecting a wired USB device to one of the root ports, which will trigger port manipulation actions by the MA USB host USB core system. These actions are handled locally by the MA USB host PAL.

7.3.2.1 USB device handle allocation

The Open Device action triggers the MA USB host to transmit a USB Device Handle Request (USBDevHandleReq) packet (Section 6.3.4) to the MA USB device managing the target USB device. The USBDevHandleReq packet carries the USB route string (as defined in [USB 3.1]) for the USB device, the port number on which the USB device is connected to its parent hub (root port for the

integrated USB device on MA USB device), and the speed of the USB device which is learnt as part of the capability exchange.

Upon receipt of the USBDevHandleReq packet, the target MA USB device shall respond with a USB Device Handle Response (USBDevHandleResp) packet (Section 6.3.5) to inform the MA USB host whether the request was successfully completed. If the target USB device is already allocated a device handle, then the MA USB device shall respond with a USBDevHandleReq packet carrying the device handle of the USB device and the value of the Status Code field set to INVALID_REQUEST.

The MA USB host shall ensure that at any given time there is only one USBDevHandleReq packet pending a response.

7.3.2.2 Endpoint handle allocation

Both Open EP and Configure device actions trigger the MA USB host to transmit an EP Handle Request (EPHandleReq) packet (Section 6.3.6) to the MA USB device managing the target USB device (identified by the Device Handle field in the EPHandleReq packet).

The EPHandleReq packet is only valid after USB device handle allocation has been successfully completed.

When triggered by an Open EP action, the EPHandleReq packet is transmitted to request an EP handle for the default control endpoint (endpoint 0) on the target USB device. In this case, the EPHandleReq packet carries a single EP descriptor, corresponding to the default control endpoint on the device. The USB descriptor fields embedded inside the EP descriptor are set as follows: *bLength*=7, *bDescriptorType*=5 (ENDPOINT), *bEndpointAddress* =00h, *bmAttributes*=0, *wMaxPacketSize*=8 for an LS or FS device, 64 for an HS device and 512 for an Enhanced SuperSpeed device, and *bInterval*=0.

When triggered by a Configure Device action, the EPHandleReq packet is transmitted to request EP handles for a number of endpoints on the target USB device. In this case, the EPHandleReq packet carries an EP descriptor for each of the endpoints activated under the selected device configuration.

Upon receipt of the EPHandleReq packet, the target MA USB device shall respond with an EP Handle Response (EPHandleResp) packet (Section 6.3.7) to inform the MA USB host whether the request was successfully completed, and also to return MA USB attributes of the EP handle such as the credit consumption unit (Section 5.5.1). Note that for EP0 the USB Address subfield in EP handle is set to the default value 0.

7.3.2.3 Modification of EP0 parameters

The Modify EP0 action has two usages,

- It may be invoked to modify the maximum packet size for the default control endpoint of an FS USB device from its initial value.
- It is also invoked to request an updated EP0 handle after the USB device has been assigned a USB address, i.e., after the MA USB host receives a Set USB Device Address Response (SetUSBDevAddrResp) packet (Section 6.3.22).

The action triggers the MA USB host to transmit a Modify EP0 Request (ModifyEP0Req) packet (Section 6.3.20) to the MA USB device in control of the target USB device, which includes the device handle of the target USB device, as well as the EP0 handle and the maximum packet size for endpoint 0. The MA USB host shall transmit a ModifyEP0Req packet after receiving the response to a Set USB Device Address Request packet sent to a USB device in Default or Address states.

Upon receipt of the ModifyEP0Req packet, the target MA USB device shall respond with a Modify EP0 Response (ModifyEP0Resp) packet (Section 6.3.21) to inform the MA USB host whether the request was successfully completed. The ModifyEP0Resp packet shall include a new EP0 handle if the EP0 handle in the ModifyEP0Req packet carried the default USB address (00h) (refer to the EP handle structure in Section 6.2.1.5), and the target USB device is in Address state. The MA USB host PAL shall return the result of the ModifyEP0Req to the MA USB host USB core system.

7.3.2.4 USB device address allocation

The address device action triggers the MA USB host to transmit a Set USB Device Address Request (SetUSBDevAddrReq) packet (Section 6.3.22) to the MA USB device managing the target USB device. The SetUSBDevAddrReq packet carries the device handle of the USB device for which the set address request is targeted.

Upon receipt of a SetUSBDevAddrReq packet, the target MA USB device shall respond with a Set USB Device Address Response (SetUSBDevAddrResp) packet (Section 6.3.23) to inform the MA USB host whether the request was successfully completed and return the USB device address to the MA USB host.

7.3.2.5 Update of USB device parameters

The evaluate device action triggers the MA USB host to transmit an Update Device Request (UpdateDevReq) packet (Section 6.3.24) to the target MA USB device. The UpdateDevReq packet includes the USB device handle for which the request and the USB device parameters are targeted.

NOTE — The host may choose not to transmit an Update Device Request packet when enumerating a non-hub integrated USB device.

Upon receipt of the UpdateDevReq packet, the target MA USB device shall respond with an Update Device Response (UpdateDevResp) packet (Section 6.3.25) to inform the MA USB host whether the request was successfully completed. The MA USB host shall return the result of Update Device Request to MA USB host USB core system.

7.3.3 Enumeration of a USB device downstream of an MA USB hub

Enumeration of a non-integrated USB device behind an MA USB hub (or behind an external hub downstream of an MA USB hub) follows the same procedure as the integrated device with the following differences:

1) Port status change event is not emulated by MA USB host and is a notification event delivered to MA USB host over the air.
2) The port manipulation actions are not handled locally by MA USB host and are transferred over the medium to the MA USB hub as control transfer requests.

Example generalized sequences of the enumeration of a USB device downstream of an MA USB hub is shown in Figure 48.

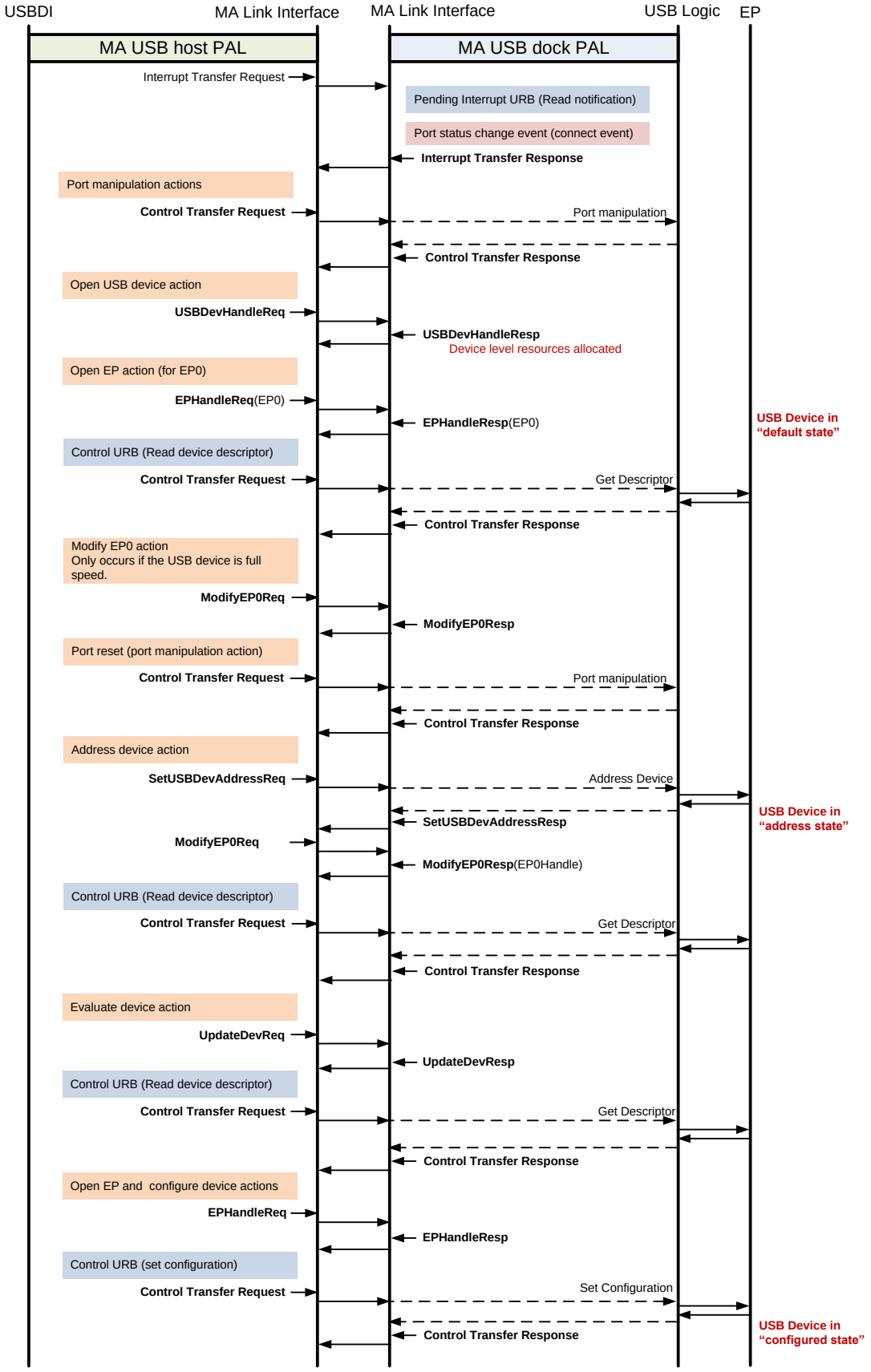


Figure 48—Enumeration of a USB device downstream of an MA USB hub

7.3.4 Support of Stream Protocol

In order to use the Enhanced SuperSpeed Stream Protocol [USB 3.1] on a bulk endpoint, the MA USB host first opens the streams on the endpoint by transmitting an Endpoint Open Streams Request (EPOpenStreamReq) packet (Section 6.3.44) indicating the number of streams to be opened. The MA USB host shall ensure that the requested number of streams is supported by the MA USB device (as indicated in CapResp packet). The MA USB device shall respond to an EPOpenStreamReq packet with an Endpoint Open Streams Response (EPOpenStreamResp) packet (Section 6.3.45) and inform the MA USB host whether the Endpoint Open Streams Request was successfully completed and return the Stream IDs for the opened streams. If the number of streams requested by the host is larger than the value supported by the MA USB device, the MA USB device shall set the value of the Status Code field in EPOpenStreamResp packet to INSUFFICIENT_RESOURCES. The number of streams that are included in the EPOpenStreamResp may be less than the number of streams requested by the MA USB host to meet the MA link MTU size. An MA USB host that receives a smaller number of Stream IDs than it asked for may transmit additional EPOpenStreamReq packets with the Open Stream field set to 0 to retrieve the remaining Stream IDs. Figure 49 illustrates an example in which the MA USB host transmits multiple EPOpenStreamReq packets to retrieve all the requested Stream IDs.

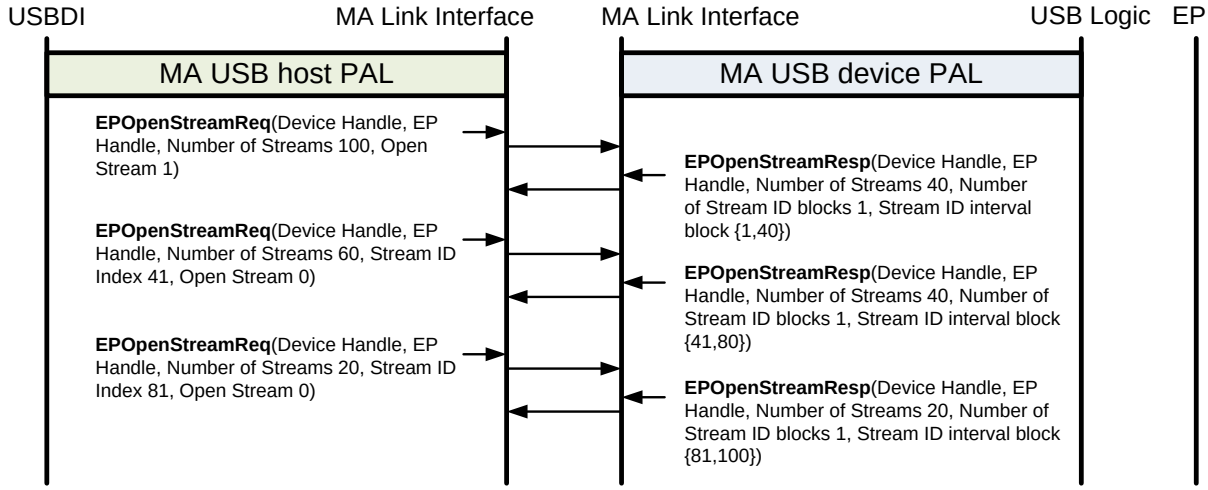


Figure 49—Example of Open Stream Request and Response packet exchanges

For the MA USB host to change the number of open streams on an endpoint, or to change the USB device configuration, or any other action that requires the endpoint to cease operation, it shall first close the open streams on the endpoint. To do so, the MA USB host transmits an Endpoint Close Streams Request (EPCloseStreamReq) packet (Section 6.3.46) to the MA USB device; the endpoint handle of the target endpoint shall be in Inactive state prior to receiving the EPCloseStreamReq packet and all pending transfers are considered cancelled following closing of the streams. The MA USB device shall respond to an EPCloseStreamReq packet with an Endpoint Close Streams Response (EPCloseStreamResp) packet (Section 6.3.47) and inform the MA USB host whether the Endpoint Close Streams Request was successfully completed.

7.3.5 USB device reset

The MA USB host transmits a USB Device Reset Request (USBDevResetReq) packet (Section 6.3.48) to the target MA USB device to request reset of the integrated USB device. In case of an MA USB hub, it informs the MA USB hub of the reset of a downstream USB device (Figure 50). If the USB device is connected to a physical USB controller implemented in the MA USB device, the USBDevResetReq packet informs the controller of transition of a USB device under its control to Default state and triggers

relevant actions if applicable (in case of an xHCI, for example, the USB controller would initiate USB Reset Device command). If the MA USB device does not implement a physical USB controller then the USBDevResetReq packet may result in no operation by the MA USB device.

The MA USB device shall respond to a USBDevResetReq packet with a USB Device Reset Response (USBDevResetResp) packet (Section 6.3.49).

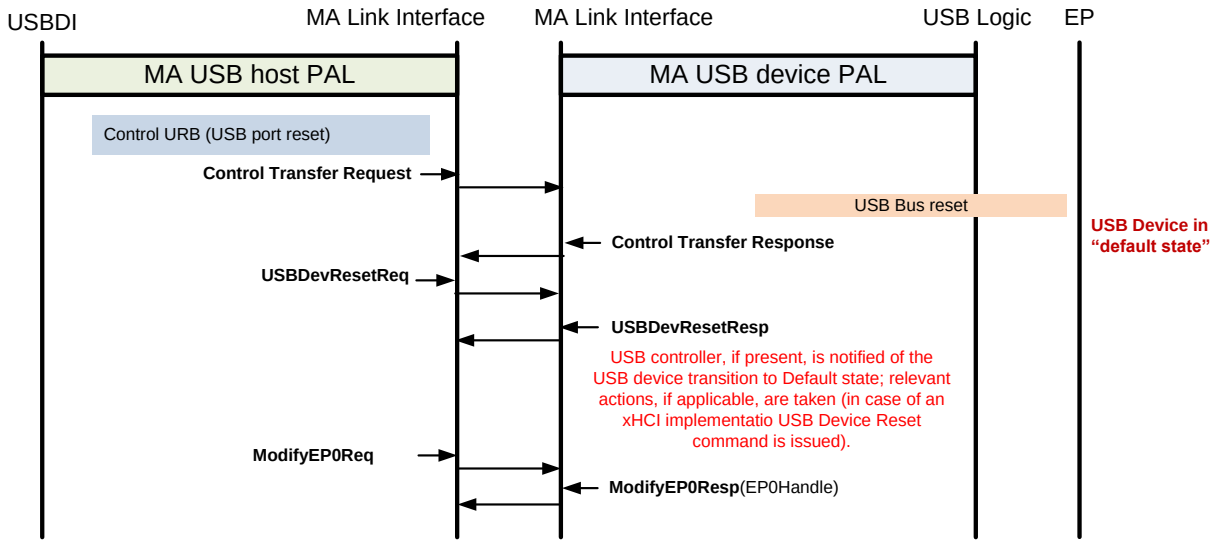


Figure 50—USB device reset

Note that with transitioning of a USB device to Default state, its USB device address is set to 0 and hence the EP0 Handle for the USB device is modified. The MA USB host shall follow a USBDevResetResp packet with ModifyEP0Req packet to receive the updated EP0 Handle from the MA USB device. The receipt of the USBDevResetReq packet results in transition of the assigned EP0 Handle to Active state and returns the state of the endpoint to the initial state. It is expected that next "address device action" in the MA USB host would trigger transmission of SetUSBDevAddrReq packet and consequently transition the MA USB device to Address state (Section 7.3.2).

8 MA USB host implementation

8.1 Session management

8.1.1 Session states

MA USB session is defined between an MA USB host and an MA USB device. The state diagram of MA USB session is depicted in Figure 51 and the states are described below.

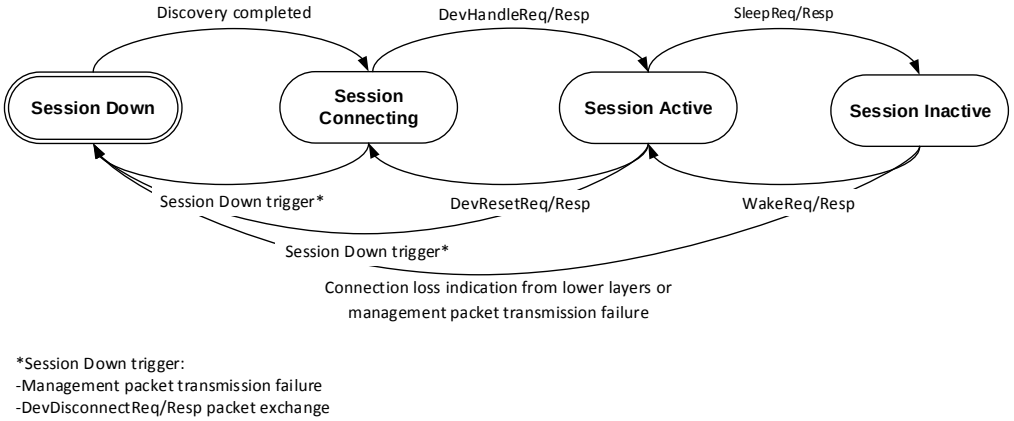


Figure 51—MA USB session state diagram

8.1.1.1 Session Down state

The Session Down state is entered on any of the following events:

- Power-on-reset.
- Management packet transmission failure (Section 5.2.1.1).
- DevDisconnectReq/Resp packet exchange (Section 8.1.3).

While in Session Down state, there is no communication between the MA USB host and the MA USB device and neither MA USB host nor MA USB device is active. In this state the lower layers of the MA USB device and the MA USB host are involved in discovery of either an MA USB host or MA USB devices with which the MA USB device and the MA USB host may connect, respectively.

The state exits when:

- The MA USB host/MA USB device receives an indication from the lower layers that discovery of an MA USB device/MA USB host is completed.

8.1.1.2 Session Connecting state

The Session Connecting state is entered on following events:

- The MA USB host/MA USB device receives an indication from the lower layers that discovery of an MA USB device/MA USB host is completed.
- From the Active state, following DevResetReq/Resp packet exchange (Section 6.3.18).

Entering this state triggers the session setup procedure by the MA USB host, consisting of MA USB reset and capability exchange mechanisms (Section 8.1.2). While in this state, the MA USB host may optionally initiate the MA USB session teardown mechanism (Section 8.1.3).

While in this state the MA USB device waits for either session setup related packets (Section 8.1.2) or MA USB session teardown related packets (Section 8.1.3) from the MA USB host.

The state exits on the following events:

- USBDevHandleReq/Resp packet exchange (Section 6.3.4), following completion of Session setup (Section 8.1.2) and the decision to continue the session with the MA USB device.
- There is a management packet transmission failure (Section 5.2.1).
- DevDisconnectReq/Resp packet exchange (Section 8.1.3).

8.1.1.3 Session Active state

The Session Active state is entered on the following events:

- USBDevHandleReq/Resp packet exchange (Section 6.3.4), following completion of session setup procedure (Section 8.1.2) and the decision to continue the session with the MA USB device.
- WakeReq/Resp packet exchange (Section 6.3.58).
- An implicit WakeReq/Resp packet exchange (Section 8.2).

While in this state, the MA USB host and the MA USB device may exchange management, control, or data packets.

The state exits on following events:

- DevResetReq/Resp packet exchange (Section 8.1.2).
- There is a management packet transmission failure (Section 5.2.1.1).
- DevDisconnectReq/Resp packet exchange (Section 8.1.3).
- SleepReq/Resp packet exchange (Section 8.2).

8.1.1.4 Session Inactive state

The Session Inactive state is entered on the following event:

- SleepReq/Resp packet exchange (Section 8.2).

While in this state, there is no communication between the MA USB host and the MA USB device except for the power management related packets including implicit WakeReq/Resp packets (Section 8.2) and the MA USB host and/or the MA USB device may be in a low power state. The state exits on the following events:

- There is an indication of loss of connection from lower layers.
- There is a management packet transmission failure (Section 5.2.1.1).
- WakeReq/Resp packet exchange (Section 8.2).
- An implicit WakeReq/Resp packet exchange (Section 8.2).

8.1.2 Session setup

After the device discovery is completed, the establishment of a secure communication link between MA USB host and MA USB device triggers the initiation of the MA USB host PAL on the MA USB host, and the MA USB device PAL on the device. After being initiated MA USB device PAL takes no action and waits for packets from the host, while the MA USB host initiates MA USB device reset and MA USB capability exchange (Figure 52).

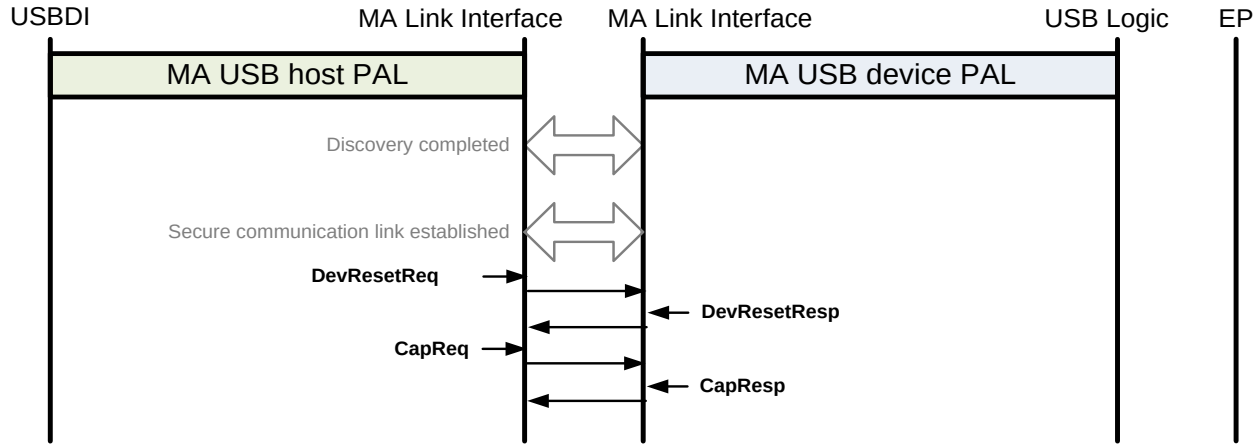


Figure 52—MA USB device session setup

8.1.2.1 MA USB device reset

The MA USB host transmits an MA USB Device Reset Request (DevResetReq) packet (Section 6.3.18) to the target MA USB device, to request the MA USB device to clear all its internal states and join the MSS operated by the MA USB host.

The target MA USB device is reached through a network address normally determined after media-specific discovery. The DevResetReq packet includes the SSID selected by the MA USB host (in the SSID field), as well as the nonzero address assigned to the MA USB device by the MA USB host (in the Device Address field). The SSID value selected by the MA USB host shall be a random integer between 1 and 254, and shall be different from any SSID values that the MA USB host has possibly observed during media-agnostic discovery as well as during normal operation after discovery. SSID values 0 and 255 are reserved for future protocol extensions, including diagnostics. The MA USB device address carried in the Device Address field shall be unique within the MSS in which the MA USB device is operating.

The target MA USB device shall respond to a DevResetReq packet with an MA USB Device Reset Response (DevResetResp) packet (Section 6.3.19) to indicate its willingness to join the MSS (i.e., move the state of its session with the MA USB host to Session Connecting), and whether the reset operation was successfully completed. A target MA USB device that responds to a DevResetReq packet and indicates successful reset shall store the SSID and the device address it received in the DevResetReq packet, and shall ignore any MA USB packets, other than possibly another DevResetReq packet, which does not carry the same SSID and device address values (with the exception of PingReq packets with the Device Address field set to 0xFF). The MA USB host shall use the same SSID and device address values in all following packet exchanges with the target MA USB device. The MA USB host shall not transmit any packet other than DevResetReq packet to a target MA USB device unless it receives a DevResetResp packet from the device with the status code of SUCCESS.

NOTE — An MA USB device may choose not to respond to a DevResetReq packet if it is not willing to join the MSS indicated in the DevResetReq packet. For example, an active MA USB device may choose to ignore a DevResetReq packet indicating a different MSS from what the device is operating in. The context for a received DevResetReq packet is normally made available through media-specific discovery mechanisms and is beyond the scope of this specification.

NOTE — MA USB device reset is different from USB device reset (Section 7.3.5).

8.1.2.2 Capability exchange

For MA USB capability exchange, the MA USB host transmits an MA USB Capability Request (CapReq) packet (Section 6.3.2) to the target MA USB device. The MA USB device shall respond with an MA USB Capability Response (CapResp) packet (Section 6.3.3) to inform the MA USB host whether the MA USB Capability Request was successfully completed on the MA USB device, and if yes include the maximum number of devices and the maximum number of endpoints for which the MA USB device can track state.

After a successful MA USB device capability exchange the session between the MA USB host and the MA USB device is considered established.

Establishment of MA USB session shall trigger emulation of one Port Status Change event (Section 7.3.2) in the MA USB host if the session is established with an MA USB device or an MA USB hub with an integrated USB 2.0 hub, and two Port Status Change events if the session is established with an MA USB hub with an integrated USB 3.1 hub.

8.1.3 Session tear down

The MA USB session is torn down when data communications are no longer available or needed between the MA USB host and the MA USB device. The session tear down can occur either explicitly, where either the MA USB host or device chooses to tear down an MA USB session, or implicitly when the communication between the MA USB host and the MA USB device is inhibited.

The explicit tear down may occur, for example, when the user performs a platform specific operation to indicate the session tear down. In case of an explicit tear down, the initiator shall notify the peer MA USB entity the MA USB session is being torn down.

The implicit tear down occurs with loss of connectivity between the MA USB host and the MA USB device (In a wireless medium this may happen when the MA USB host and device platforms move out of communication range, or when persistent interference exists). When a loss of connectivity is detected, the MA USB host and device shall each locally initiate session tear down procedures.

8.1.3.1 Implicit session tear down

Implicit tear down (Figure 53) is initiated locally by the MA USB host and the MA USB device. The implicit tear down is initiated when either the MA USB host or the MA USB device is notified by the lower layers of the loss of connectivity or when the transmission failure of a management packet indicates a connection loss.

In implicit session tear down the MA USB host emulates a port status change event equivalent to unplugging of a wired USB device from one of the root ports. However, all the actions resulting from the port status change event are handled locally by the MA USB host. Following the port status change event, the MA USB host initiates the USB device removal; again, all the actions resulting from the USB device removal are handled locally by the MA USB host. As a final step, the host clears all the resources allocated to the USB device as well as all the resources allocated to the MA USB device.

Similar to the MA USB host, in an implicit session tear down the MA USB device PAL emulates USB device removal, which is handled locally in the MA USB device, followed by clearance of all the allocated resources for the session.

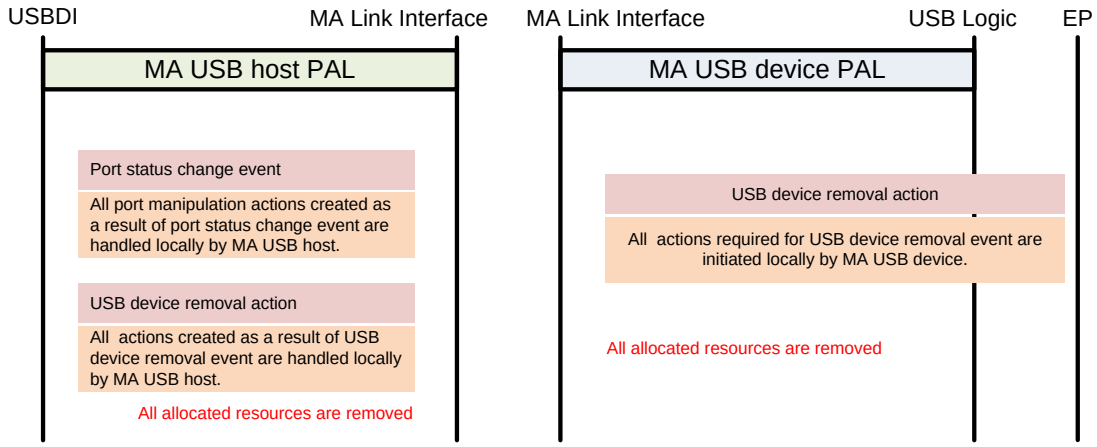


Figure 53—Implicit session tear down

8.1.3.2 Host initiated session tear down

In a host initiated session tear down (Figure 54), the tear down of the MA USB session is initiated by the user or an application on the MA USB host platform. This trigger prompts the MA USB host to emulate a port status change event equivalent to unplugging of a wired USB device from one of the root ports. All the actions resulting from the port status change event are handled locally by the MA USB host.

The port status change triggers the USB device removal and consequently the USB device removal procedure (specified in Section 8.1.3.4) between the MA USB host and the MA USB device. Following successful completion of the USB device removal procedure the host transmits MA USB Device Disconnect Request (DevDisconnectReq) packet (Section 6.3.36) to the MA USB device to explicitly terminate the session between the MA USB host and the MA USB device. The MA USB device shall respond to a DevDisconnectReq packet with an MA USB Device Disconnect Response (DevDisconnectResp) packet (Section 6.3.37). Following receipt of the DevDisconnectResp packet the MA USB host clears all the resources allocated to the USB device as well as all the resources (handles, etc.) allocated to the MA USB device. Similarly, the MA USB device clears all the allocated resources after successful transmission of DevDisconnectResp packet.

NOTE — An INVALID_DEVICE_HANDLE status code value returned in DevDisconnectResp packet indicates successful completion of DevDisconnectReq packet.

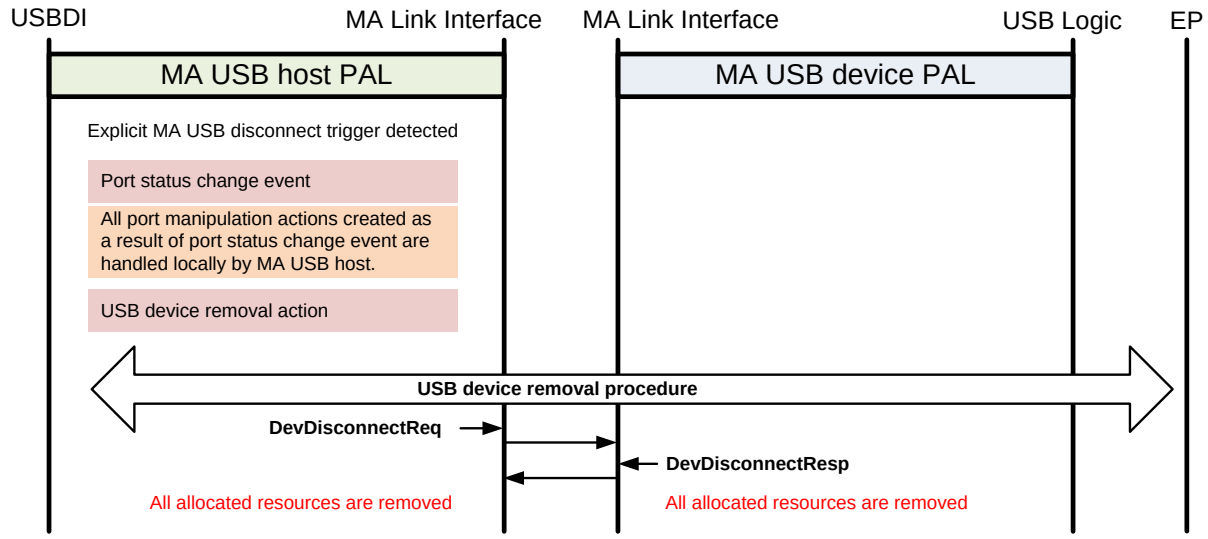


Figure 54—Host initiated session tear down

8.1.3.3 Device initiated session tear down

In device initiated session tear down (Figure 55), the MA USB device notifies the MA USB host of the MA USB device's intention of tearing down the session by transmitting an MA USB device Initiated Disconnect Request (DevInitDisconnectReq) packet (Section 6.3.38) to the MA USB host. The MA USB host shall respond to a DevInitDisconnectReq packet with an MA USB device Initiated Disconnect Response (DevInitDisconnectResp) packet (Section 6.3.39).

Following receipt of a DevInitDisconnectReq, the MA USB host shall initiate the session teardown, as specified in Section 8.1.3.2.

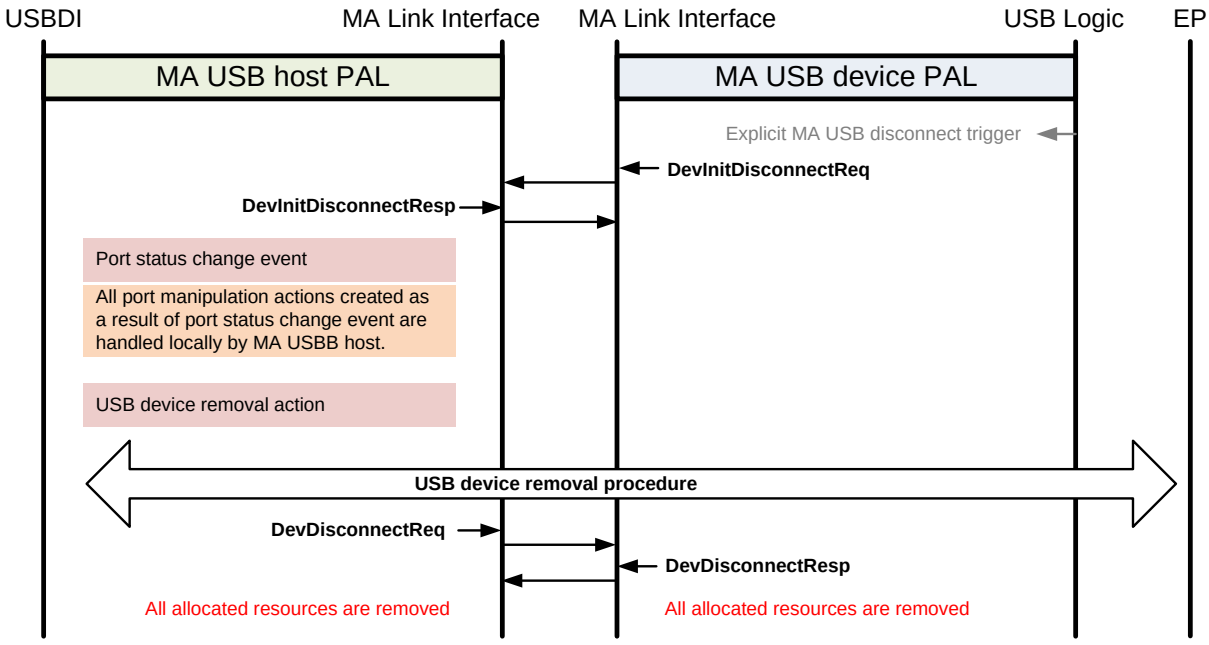


Figure 55—Device initiated session tear down

8.1.3.4 USB device removal procedure

The USB device removal procedure (Figure 56) consists of four steps:

1. The MA USB host inactivates and clears all the outstanding transfers on all the active endpoints. This step can occur in two different ways; in alternative 1 the host uses CancelTransferReq and EPInactivateReq packets (Sections 6.3.42 and 6.3.10) to stop the endpoint and clear the transfers; and in alternative 2 the host uses the EPInactivateReq packet (Section 6.3.10) followed by ClearTransfersReq packet (Section 6.3.14).
2. The MA USB host deletes all the EP handles, except for EP handle of EP 0.
3. The MA USB host deletes the EP handle of EP0.
4. The MA USB host sends a USB Disconnect Device Request (USBDevDisconnectReq) packet (Section 6.3.26) to the MA USB device to trigger the MA USB device to remove all the resources allocated to the USB device, including the device handle.

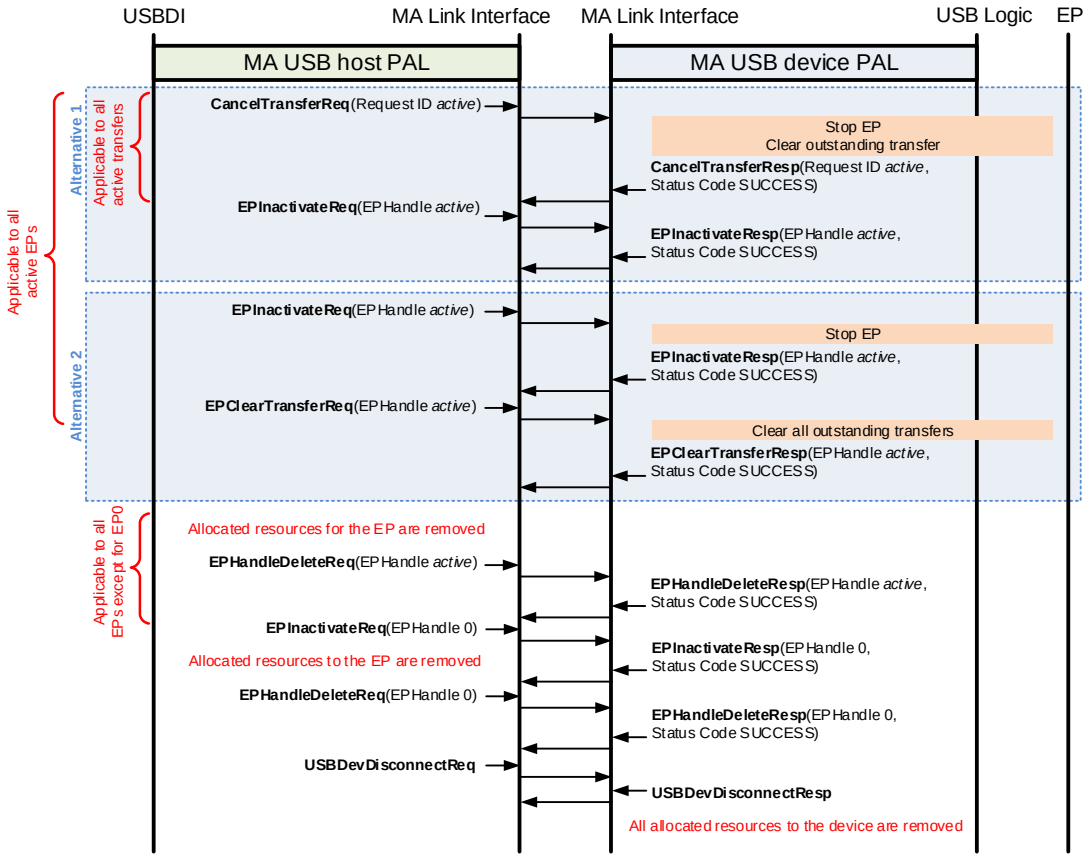


Figure 56—USB Device Removal Procedure

8.2 Power management

The MA USB power management framework defines mechanisms for the MA USB host and a target MA USB device to move their session states between Session Active and Session Inactive. The framework supports and is consistent with USB device-level power management, i.e., USB power management functions inside an MA USB host or device are not affected by the MA USB power management functions.

8.2.1 Transition to Session Inactive state

8.2.1.1 Initiation by the MA USB host

The MA USB host may initiate the process to move its session state to Session Inactive by following these steps in order: (1) inactivating all EP handles on the target MA USB device, (2) suspending the integrated USB device behind the target MA USB device, and (3) transmitting a SleepReq packet (Section 6.3.56) to the target MA USB device. To inactivate EP handles on the target MA USB device, the MA USB host transmits one or more EPInactivateReq packets (Section 6.3.10) to the MA USB device; the MA USB device responds to each EPInactivateReq packet with an EPInactivateResp packet (Section 6.3.11) to indicate whether the inactivate request was successful. To suspend the integrated USB device behind the target MA USB device the MA USB host transmits a USBSuspendReq packet (Section 6.3.28) to the MA USB device; the MA USB device responds to a USBSuspendReq packet with a USBSuspendResp packet (Section 6.3.29) to indicate whether the suspend request was successful.

The SleepReq packet carries the timeout values the target MA USB device has to observe when it initiates a management, control or data packet exchange in Session Inactive state. Upon receiving the SleepReq packet, the target MA USB device shall respond with a SleepResp packet with the Status Code field set to 0 (NO_ERROR) if it grants the transition request, and set to REQUEST_DENIED otherwise. Should the target MA USB device accept the request to move its session state to Session Inactive, through the SleepResp packet it shall indicate its own timeout values for management, control and data packet exchanges initiated by the MA USB host in Session Inactive state, with all timeout values less than or equal to the corresponding timeout values in the SleepReq packet.

> NOTE — The MA USB device is not allowed to deny the SleepReq packet with the timeout fields set to zero following the transition of the integrated USB device to suspend state.

After the target MA USB device transmits a SleepResp packet with the Status Code field set to 0 (NO_ERROR), it will move its session state to Session Inactive, and instruct its local management entity for the MA link connecting the MA USB device and the MA USB host to perform required actions to put the MA link in a suitable low-power mode that meets the timeout values specified in the SleepResp packet.

> NOTE — The MA USB device is still required to respond to a possible SleepReq packet retried by the MA USB host. The MA USB host should use the Management Request Timeout value indicated in the SleepReq packet for the retried SleepReq packets.

Once the MA USB host receives a SleepResp packet with the Status Code field set to 0 (NO_ERROR), it will move its session state to Session Inactive, and instruct its local management entity for the MA link connecting the MA USB host and the target MA USB device to perform required actions to put the MA link in a suitable low-power mode that meets the timeout values specified in the SleepReq packet.

Figure 57 illustrates the steps to move the session state to Session Inactive.

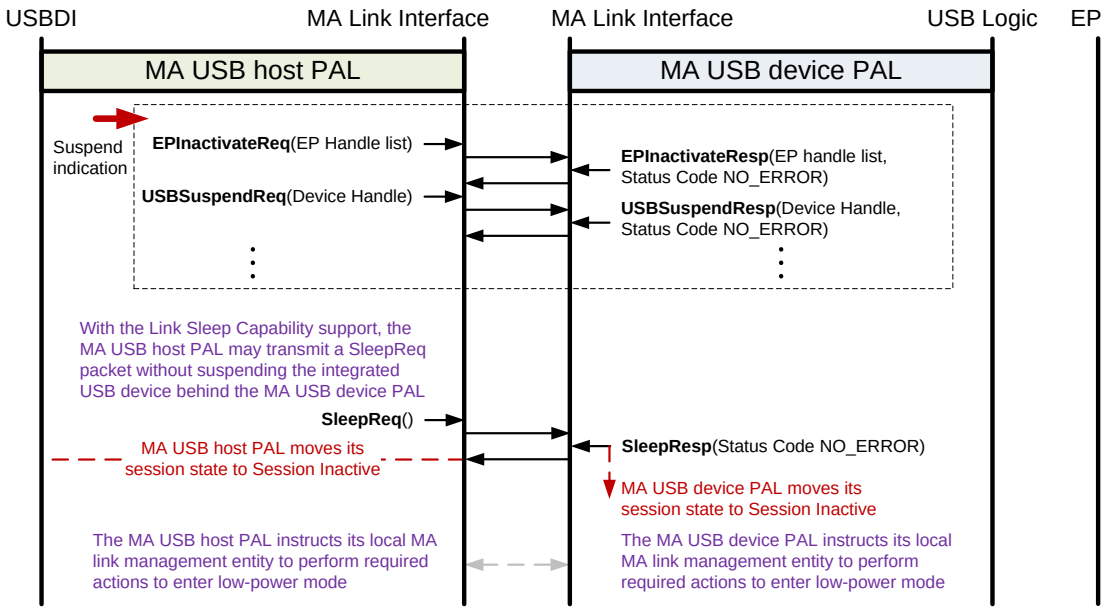


Figure 57—Transition to Session Inactive state initiated by the MA USB host

When transmitting the EPInactivateReq packet, the MA USB host sets the Suspend (SP) Flag field set to 1 to indicate to the MA USB device an EP handle is being inactivated in preparation for the transition (of the USB device the endpoint belongs to) to the Suspend state. The MA USB host shall not transmit an EPInactivateReq packet with the Suspend Flag field set to 1 to target EP handles that are already in the Inactive state.

NOTE — The MA USB device may use the Suspend Flag value for internal power management related to the endpoint. For example, an MA USB device that includes a physical USB host controller may use the Suspend Flag value to reduce the controller power consumption.

When both the MA USB host and the target MA USB device indicate support for Link Sleep capability (Sections 6.3.2.2 and 6.3.3.6), the MA USB host may initiate the process to move its session state to Session Inactive by transmitting a SleepReq packet to the target MA USB device without suspending the integrated USB device behind the MA USB device.

NOTE — An MA USB device may deny the request to move its session state to Session Inactive for a number of reasons, including for example, an upcoming IN transfer when its integrated USB device has not been suspended.

8.2.1.2 Initiation by the MA USB device

When both the MA USB host and the target MA USB device indicate support for Link Sleep capability (Sections 6.3.2.2 and 6.3.3.6), the MA USB device may initiate the process to move its session state to Session Inactive if it has only control or non-isochronous IN endpoints behind it, all IN endpoints are in the pending state, and it has had the pending state for each IN endpoint acknowledged by the MA USB host PAL.

NOTE — Examples of where a device-initiated move to Session Inactive state may be applicable are a native MA USB device with non-isochronous IN endpoints only, an MA USB hub with no USB device downstream, and an MA USB hub with USB devices downstream with non-isochronous IN endpoints only.

NOTE — To receive the MA USB host acknowledgement, an MA USB device PAL transmits a null TransferResp packet with the ARQ bit set to 1 and the Status Code field set to TRANSFER_PENDING. See Section 5.4 for details.

Once the MA USB device receives the MA USB host acknowledgement to the pending state for all its IN endpoints, it may initiate the process to move its session state to Session Inactive by transmitting a SleepReq packet (Section 6.3.56) to the MAUSB host, which also carries the timeout values the MA USB host has to observe in Session Inactive state when it initiates a management, control or data packet exchange. Upon receiving the SleepReq packet, the MA USB host shall respond with a SleepResp packet with the Status Code field set to 0 (NO_ERROR) if it grants the transition request, and set to REQUEST_DENIED otherwise. Should the MA USB host accept the request to move its session state to Session Inactive, through the SleepResp packet it shall indicate its own timeout values for management, control and data packet exchanges initiated by the target MA USB device in Session Inactive state, with all timeout values less than or equal to the corresponding timeout value in the SleepReq packet.

NOTE — The MA USB host may deny the MA USB device request to move the session state to Session Inactive for any reason, including for example, an upcoming control transfer that targets the MA USB device.

After the MA USB host transmits a SleepResp packet with the Status Code field set to 0 (NO_ERROR), it will move its session state to Session Inactive, and instruct its local management entity for the MA link connecting the MA USB host and the MA USB device to perform required actions to put the MA link in a suitable low-power mode that meets the timeout values specified in the SleepResp packet.

NOTE — The MA USB host is still required to respond to a possible SleepReq packet retried by the MA USB device.

Once the MA USB device receives a SleepResp packet with the Status Code field set to 0 (NO_ERROR), it will move its session state to Session Inactive, and instruct its local management entity for the MA link connecting the MA USB device and the MA USB host to put the MA link in a suitable low-power mode that meets the timeout values specified in the SleepReq packet.

Figure 58 illustrates the method to move the session state to Session Inactive through a SleepReq packet.

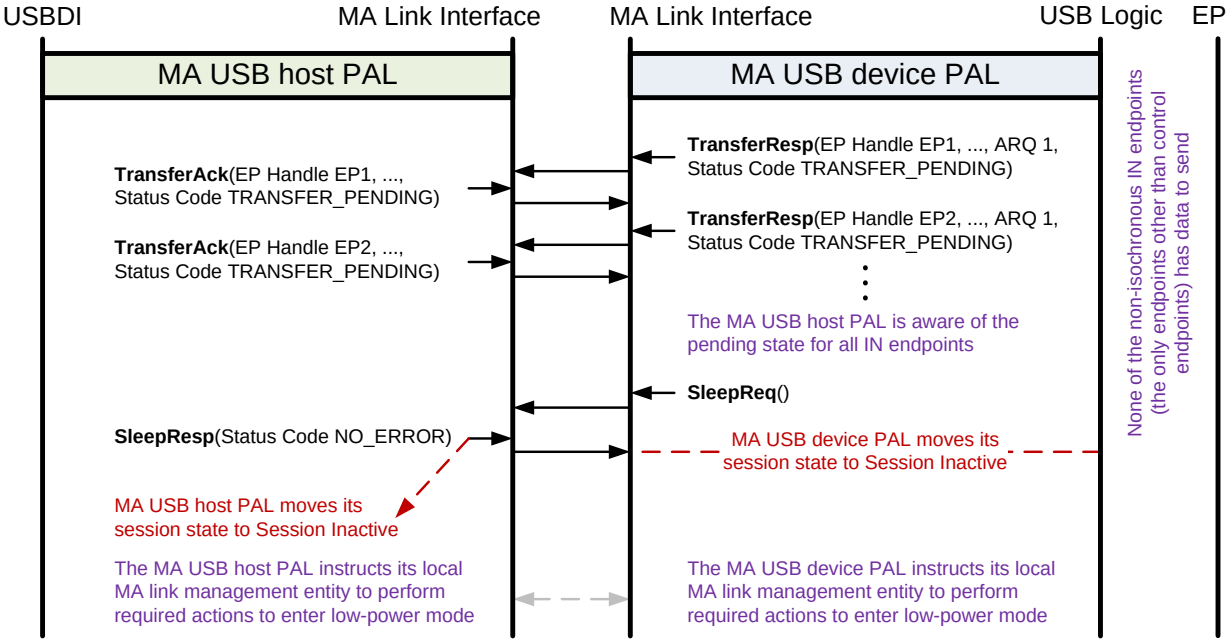


Figure 58—Transition to Session Inactive state initiated by an MA USB device

8.2.2 Transition to Session Active state

8.2.2.1 Initiation by the MA USB host

The MA USB host may move its session state from Session Inactive to Session Active (and trigger a similar transition in a target MA USB device) to resume exchange of a broader set of packets with the target MA USB device PAL.

The method to move the session state to Session Active is explicit, as it makes use of dedicated management packets.

The MA USB host instructs its local management entity for the MA link connecting the MA USB host and the target MA USB device to perform required actions to exit the low-power mode, and transmits a WakeReq packet (Section 6.3.58) to the target MA USB device. In response to a WakeReq packet, the target MA USB device shall release a WakeResp packet (Section 6.3.59) with the Status Code field set to 0 (NO_ERROR) to the management channel, move its session state to Session Active, and instruct its local management entity for the link connecting the MA USB device and the MA USB host to perform required actions to exit the low-power mode. The MA USB host moves its session state to Session Active after it receives a WakeResp packet.

Figure 59 illustrates the session state transition to Session Active when initiated by the MA USB host PAL.

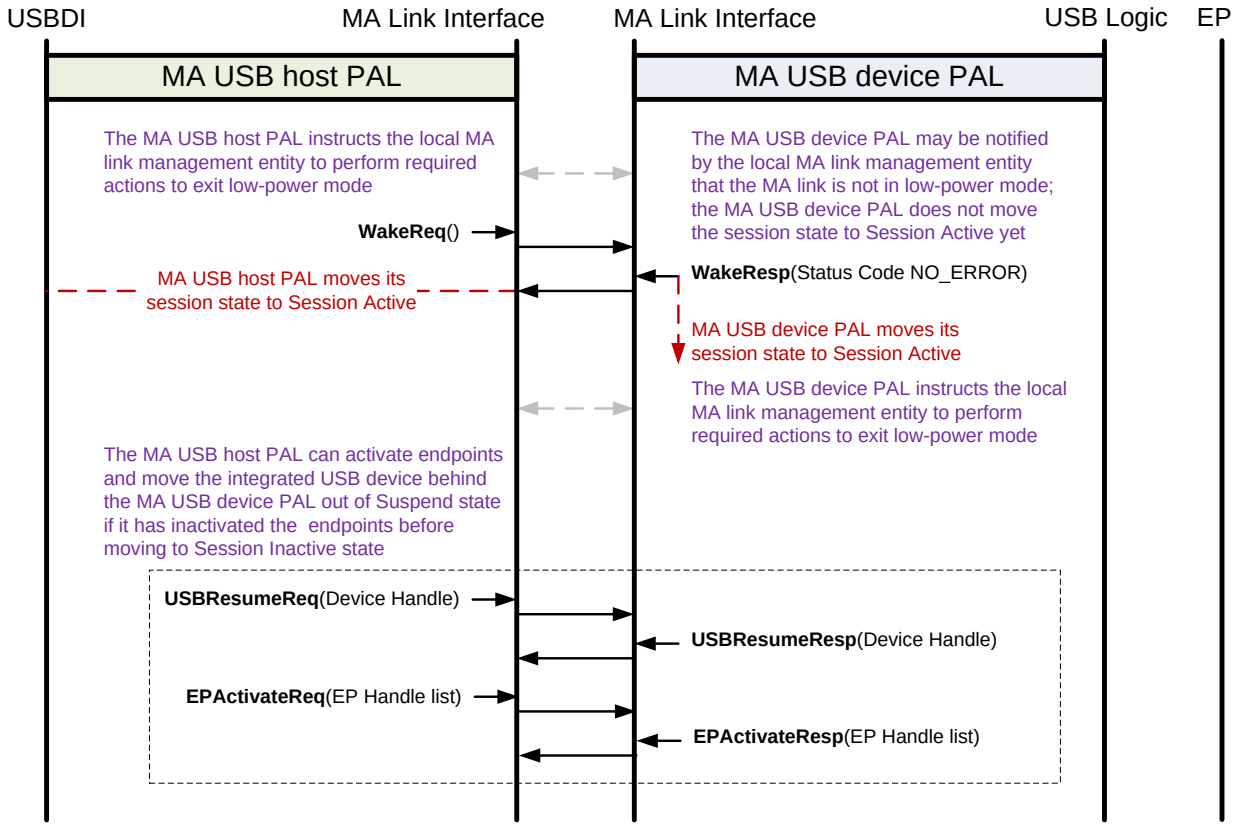


Figure 59—Transition to Session Active state initiated by the MA USB host

After receiving a WakeResp packet and moving its session state to Session Active, the MA USB host PAL can exchange management packets including USBResumeReq and EPActivateReq to move the integrated USB device out of the Suspend state and activate the relevant endpoints on the MA USB device.

8.2.2.2 Initiation by an MA USB device

An MA USB device with session in Session Inactive state may initiate the process to move its session state to Session Active for a number of reasons described below.

Move to Session Active state through a WakeReq packet (explicit request)

If both the MA USB host and the MA USB device indicate support for the Link Sleep capability (Sections 6.3.2.2 and 6.3.3.6), and the MA USB device has moved its session state to Session Inactive, the MA USB device may transmit a WakeReq (Section 6.3.58) packet to initiate the process to move its session state to Session Active. The MA USB device instructs its local management entity for the MA link connecting the MA USB device and the MA USB host to perform required actions to exit the low-power mode, and transmits a WakeReq packet to the MA USB host. In response to a WakeReq packet, the MA USB host shall release a WakeResp packet with the Status Code field set to 0 (NO_ERROR) to the management channel, move its session state to Session Active, and instruct its local MA link management entity to perform required actions to exit the low-power mode. The exchange of WakeReq and WakeResp packets follows the management packet exchange behavior defined in Section 5.2.1.1 except that the timeout value is the value of the Management Request Timeout field in the SleepReq or SleepResp packet that the MA USB host PAL transmitted when moving its session state to Session Inactive.

Figure 60 illustrates the explicit method to move the session state to Session Active through a WakeReq packet.

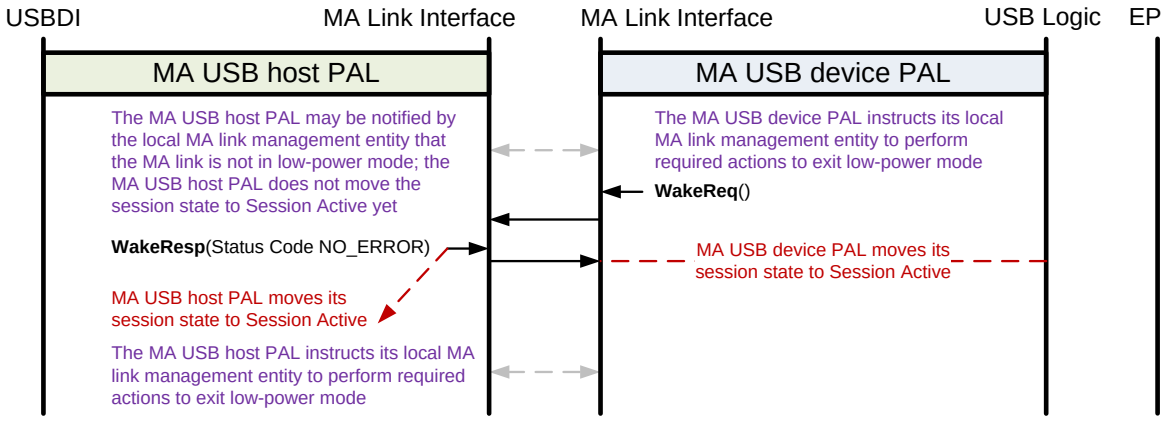


Figure 60—Transition to Session Active state initiated by an MA USB device (explicit request)

USB Remote wake (implicit request)

An MA USB device with integrated USB device in Suspend state may transmit a RemoteWakeReq (Section 6.3.32) packet. In response to a remote wake indication by the integrated USB device, the MA USB device instructs its local management entity for the MA link connecting the MA USB device and the MA USB host to perform required actions to exit the low-power mode, and transmits a RemoteWakeReq packet to the MA USB host, with the Device Handle field in the packet identifying the USB device that initiated the remote wake function. The exchange of RemoteWakeReq and RemoteWakeResp packets follows the management packet exchange behavior defined in Section 5.2.1.1, except that the timeout value is the value of the Management Request Timeout field in the SleepReq or SleepResp packet that the MA USB host PAL transmitted when moving its session state to Session Inactive.

NOTE — An MA USB device does not transmit a RemoteWakeReq packet unless the USB DEVICE_REMOTE_WAKEUP feature is set for its integrated USB device [USB 2.0].

NOTE — An MA USB device that receives a remote wake indication after it has initiated the process to move its session state to Session Inactive completes the move to Session Inactive state before transmitting a RemoteWakeReq packet.

Upon receiving a RemoteWakeReq packet while in Session Inactive session state, the MA USB host shall move its session state to Session Active (i.e., the RemoteWakeReq packet serves as an implicit WakeReq packet). The MA USB host shall acknowledge a received RemoteWakeReq packet with a Remote Wake Response (RemoteWakeResp) packet (Section 6.3.33), with the Device Handle field set to the same value as the Device Handle field in the corresponding RemoteWakeReq packet.

In response to a RemoteWakeReq packet with the USB Device Resumed field set to 0, the MA USB host shall first move the integrated USB device behind the MA USB device PAL out of the Suspend state by transmitting a USBResumeReq packet (Section 6.3.30), and then activate the relevant endpoints on the MA USB device by transmitting one or more EPActivateReq packets (Section 6.3.8).

In response to a RemoteWakeReq packet with the USB Device Resumed field set to 1, the MA USB host shall directly activate the relevant endpoints on the MA USB device without transmitting the USBResumeReq packet to the MA USB device.

Figure 61 illustrates the implicit method to move the session state to Session Active through the USB remote wake function.

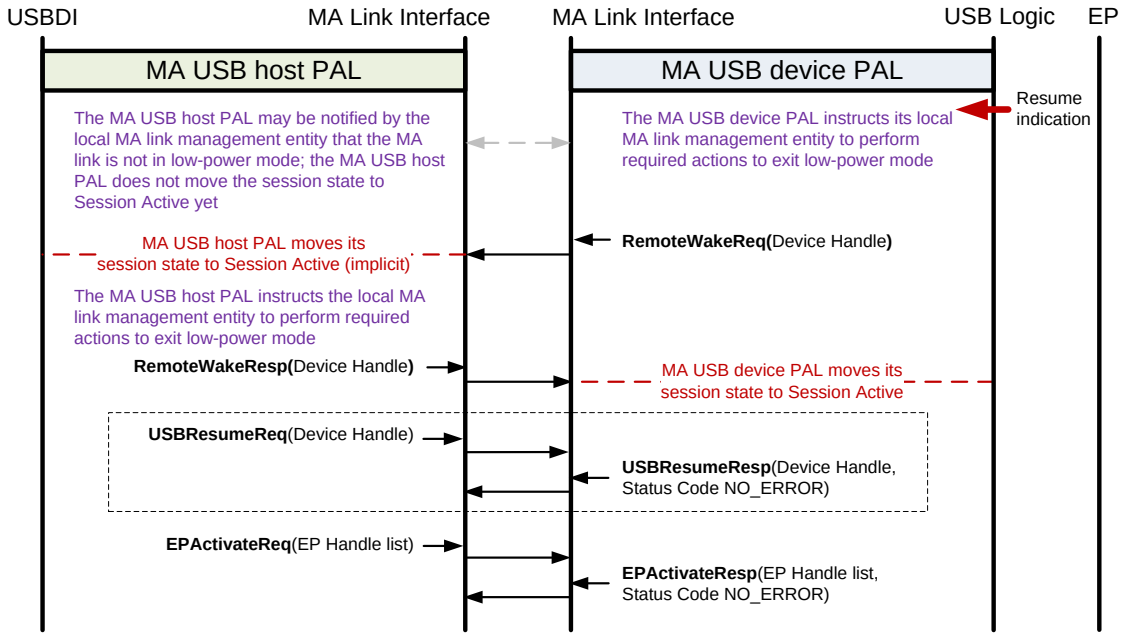


Figure 61—Transition to Session Active state initiated by an MA USB device (remote wake)

Data or management packet exchange (implicit request)

When both the MA USB host and the target MA USB device indicate support for Link Sleep capability (Sections 6.3.2.2 and 6.3.3.6), the MA USB device may move its session state from Session Inactive to Session Active (and trigger a similar transition in the MA USB host) by transmitting any packet that requires a response. The timeout applicable to the initial exchange is given by the Management Request Timeout or Data Request Timeout field in the SleepReq or SleepResp packet transmitted by the MA USB host when it moved its session state to Session Inactive. For example, an MA USB device with

pending IN endpoints may initiate the process to move its session state to Session Active once an IN endpoint in pending state has data to transmit to the MA USB host.

NOTE — This method is exclusive to an MA USB device with control or non-isochronous IN endpoints only (e.g., a native MA USB device with non-isochronous IN endpoints only, an MA USB hub with no USB device downstream, or an MA USB hub with USB devices downstream with non-isochronous IN endpoints only) that has all IN endpoints in the pending state, and has had the pending state for each IN endpoint acknowledged by the MA USB host.

In response to detecting traffic from a previously pending IN endpoint, the MA USB device instructs its local management entity for the MA link connecting the MA USB device and the MA USB host to perform required actions to exit the low-power mode, and resumes a pending IN transfer by transmitting one or more TransferResp packets belonging to the transfer, with the EoT field set to 1in at least the first TransferResp packet that the MA USB device PAL transmits.

NOTE — The follow-on TransferResp packets may have the ARQ field set to 1 to solicit the MA USB host for a TransferReq or TransferAck packet that serves as an implicit WakeReq packet.

In response to the first TransferResp packet received while its session is in Session Inactive state, the MA USB host shall move its session state to Session Active (i.e., the TransferResp packet serves as an implicit WakeReq packet), instruct its local management entity for the MA link connecting the MA USB host and the MA USB device to perform required actions to exit the low-power mode, and reset the transfer timers for the pending transfer request the received TransferResp packet identifies through its Request ID field. The exchange of the first TransferResp packet and the corresponding TransferAck packet follows the data packet exchange behavior defined in Section 5.2.1.2, except that the timeout value is the value of the Data Request Timeout field in the SleepReq or SleepResp packet that the MA USB host transmitted when moving its session state to Session Inactive.

NOTE — For example, if the received TransferResp packet has the EoT field set to 0, the MA USB host will expect to receive a follow-on TransferResp packet no later than aTransferKeepAlive after it receives the first TransferResp packet, or the MA USB host will transmit a TransferReq packet to inquire about the transfer status.

Figure 62 illustrates the implicit method to move the session state to Session Active through resuming a pending IN transfer.

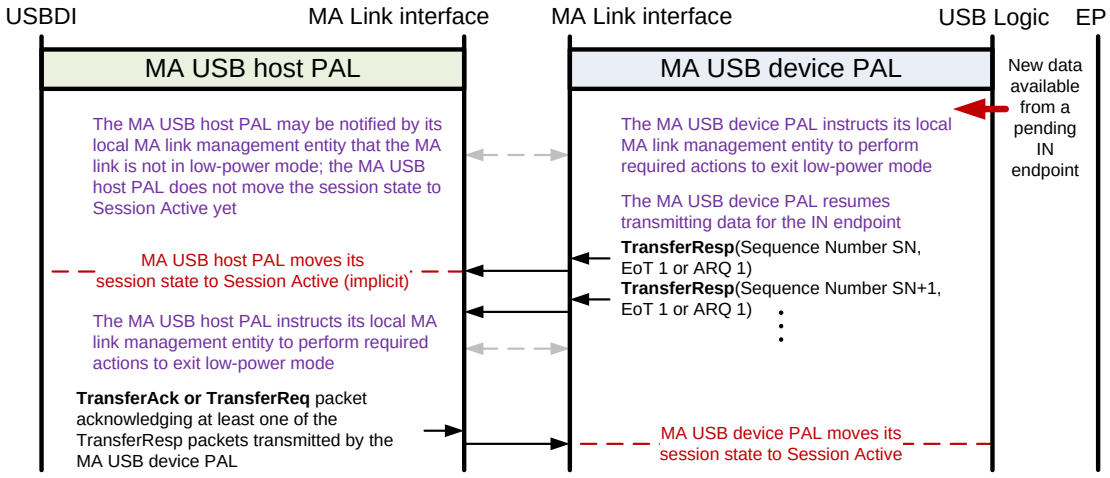


Figure 62—Transition to Session Active state initiated by an MA USB device (IN transfer)

9 MA USB hub

9.1 MA USB hub enumeration

The enumeration of an integrated USB hub on MA USB device is the same as enumeration of a non-hub integrated USB device, except for the case that the integrated hub is a USB 3.1 hub. In that case the following modifications apply:

The integrated USB device enumeration procedure will occur twice, i.e., the MA USB host learns whether the integrated USB device is a USB 3.1 hub as part of the capability exchange. If it is a USB 3.1 hub, then the MA USB host emulates two port status change events.

> NOTE—For a fully compliant hub it does not matter which hub is enumerated first, but it is recommended that the Enhanced SuperSpeed hub be enumerated first.

Hence, enumeration of the MA USB hub with an integrated USB 3.1 hub consists of the enumeration of the Enhanced SuperSpeed hub as a standalone integrated USB device (as described in Section 7.3.2 and depicted in Figure 47) followed by the enumeration of the USB 2.0 hub as a standalone integrated USB device (again as described in Section 7.3.2 and depicted in Figure 47).

9.2 MA USB hub session tear down

The session teardown for an MA USB hub is similar to the MA USB device as described in Section 8.1.3 with the following changes:

- If the MA USB hub integrates a USB 3.1 hub, i.e., a USB 2.0 hub and an Enhanced SuperSpeed hub, there are two port status change events and USB device removal events and procedures instead of one. Note, however, that only one DevInitDisconnectReq and/or DevDisconnectReq is sufficient for the MA USB hub.
- The USB device removal procedure is first carried out for all the USB devices downstream of the integrated hub(s), and then for the integrated hub(s) on the MA USB hub.

9.2.1 Removal procedure for a USB device downstream of an MA USB hub

Removal of a USB device connected downstream of an MA USB hub is as captured in Section 8.1.3.4, except for the fact that the host is notified of the port status change event (the disconnect event) on the MA USB hub via USB control transfers. Following the completion of the port manipulations action triggered by the host, the USB device removal procedure as depicted in Figure 56 follows.

9.3 MA USB hub power management

The power management mechanisms defined for an MA USB device in Section 8.2 also apply to an MA USB hub. The following clarifications, however, may be found useful:

For transition of session state of an MA USB hub to Inactive State, all the endpoints on and behind the integrated USB 2.0 and the Enhanced SuperSpeed hubs shall be first in the Suspend state and the corresponding endpoint handles deactivated. Following that, one USBSuspendReq packet is transmitted for each integrated hub to transition the hub to the Suspend state. A single SleepReq packet then moves the session state to the Inactive State.

Similarly, for session state transition of an MA USB hub out of Inactive State, one WakeReq packet is transmitted by the MA USB host to the MA USB hub. This packet is followed by USBResumeReq

1 packets for each integrated hub and the transition of all the endpoints (and their corresponding endpoint
2 handles) on and behind the integrated USB 2.0 and/or the Enhanced SuperSpeed hubs to Active state.

10 Protocol constants

Table 70 lists the MA USB protocol constants.

Table 70—MA USB protocol constants

Protocol constant	Value
aDataChannelDelay	Media dependent
aManagementChannelDelay	Media dependent
aMaxIsochLinkDelay	Media dependent
aMaxFrameDistance	895 frames
aManagementResponseTime	5 msec
aManagementRequestTimeout	aManagementResponseTime + 2×aManagementChannelDelay
aTransferResponseTime	10 msec
aTransferTimeout	aTransferResponseTime + 2×aDataChannelDelay
aTransferKeepAlive	aTransferResponseTime + aDataChannelDelay
aDefaultKeepAliveDuration	0
aMaxTransferLifetime	1 sec
aTransferRepeatTime	10 msec
aMaxMediaTimeError	10 µs
aMaxMediaTimeSamplingError	10 µs
aMaxTransmissionDelayError	10 µs
aMinSynchFrequencyActive	20 msec
aMinSynchFrequencyIdle	1 sec
aMaxRequestID	$2^8 - 1$
aMaxSequenceNumber	$2^{24} - 2$
aInvalidSequenceNumber	$2^{24} - 1$
aMaxDialogToken	$2^{10} - 1$
aMinControlTransferBufferSize	4,104 Bytes

Table 71 lists the MA USB protocol variables.

Table 71—MA USB protocol variables

Protocol variables	Value
aBulkTransferRetries	Minimum value 5
aControlTransferRetries	Minimum value 5
aInterruptTransferRetries	Minimum value 3
aManagementRetries	Minimum value 4
aTransferSetupRetries	Minimum value 4

Appendix A – Discovery information prior to establishing a secure connection

When connecting devices for the first time it is necessary to go through a process of discovery before establishing a secure connection. As part of the discovery process an MA USB host should present to a user a list of devices which are available for association. This list can be filtered by a discovery application to only include devices suited to a particular application or which can be supported by a given MA USB host.

USB hosts fall into two main categories:

- Standard Hosts with a full OS which can support the majority of devices available on the market.
  - These hosts will have a large number of built in drivers as well as the option to install third party drivers e.g. from the internet.
- Embedded Hosts, with support targeted at a sub-set of products
  - These hosts will have a limited set of device classes which can be supported, and potentially limited support within those classes.
  - They typically have a Targeted Peripheral List [OTG&EH3] which is the list of peripherals they are guaranteed to support.
  - They may have limited hardware in terms of the number of supported endpoints.

The intention of this section is to outline the information which should be provided by an MA USB device in order to facilitate the process of discovery for different types of host in order to enable the best user experience. This section does not detail the exact mechanism which is to be used for the information since this is a media specific decision.

There are two key aspects to this process:

- Users are able to successfully identify the device they wish to connect to.
- Hosts are able to determine which devices they can support with a minimal number of either false positive or false negative detections.

A.1 User identification of a device

The most basic information that a user needs e.g. to match the MA USB device "on their desk" to the MA USB device listed on their screen is the manufacturer name and a description of the device.

The following USB related information should be made available:

- Manufacturer name string
  - As indexed in the Standard Device Descriptor by iManufacturer
- Product name string
  - As indexed in the Standard Device Descriptor by iProduct
- Device Release Number
  - As defined in the Standard Device Descriptor by bcdDevice
- Device Serial Number
  - As indexed in the Standard Device Descriptor by iSerialNumber

It would also be helpful to identify the category and sub-category of device to the user. This feature is not supported by USB but may be part of the Media being used.

A.2 Platform driver matching

The information and process required to successfully determine driver support for a given host is complex. However, it is potentially possible to determine driver support with a minimal amount of information.

A.2.1 Driver Identification

Initial driver identification in [USB3.1] is currently either carried out by matching a Vendor ID and Product ID or by using Device Class. Devices may support multiple classes, each representing a different function, and so multiple tuples of information should be provided.

The minimal information required to perform initial discovery is the following:

- Vendor ID
  - As defined in the Standard Device Descriptor by idVendor
- Product ID
  - As defined in the Standard Device Descriptor by idProduct
- A list of the supported device classes in the form of tuples of the following:
  - Device class, Sub-Class, Protocol
    - As defined in the Standard Device, IAD [USB 3.1], WHCM [WMC 1.1] or Interface Descriptors

For simple devices, when the device class tuple used in combination with a category or sub-category provided by a particular Media this may be sufficient to identify which part of the device class will be needed.

A.2.2 Configuration Descriptor Set

Real USB devices are described by their descriptors. These provide the major of information about what the device requires in terms of driver support.

In the ideal case the MA USB Device will expose sufficient functionality prior to secure connection, to allow multiple GET_DESCRIPTOR requests to be made and responded to in sequence. If multiple requests cannot be made on a given media then a two-step process should be used. In the first step information is provided which provides sufficient information to request the descriptors in the second step.

To facilitate this process the driver should provide the following information as part of initial discovery:

- Number of Configurations
  - As defined in the Standard Device Descriptor by bNumConfigurations
- Total size of all Configuration Descriptor Sets combined in bytes.
- Number of Strings in all configurations
  - Total number of strings defined to enable indexing/retrieval of strings.
- Maximum size of string table in bytes for any given language.
- String descriptor zero
  - As defined in [USB3.1] contains the list of LANGID codes supported by the device.

It is recommended that a request is made available in order to retrieve the appropriate descriptors rather than these being broadcast. This request should contain the following information based on the Standard Get Descriptor device request:

- bmRequestType = 10000000B

- Only required if a Control Endpoint is being emulated otherwise some other means can be used to identify GET_DESCRIPTOR requests.

- bRequest=8 (GET_DESCRIPTOR)
  - Only required if a Control Endpoint is being emulated otherwise some other means can be used to identify GET_DESCRIPTOR requests.
- wValue:
  - High Byte=Descriptor Type (as defined in Table 9-6 of [USB3.1])
  - Low Byte=Index into required configuration or string descriptor
- wIndex=zero or Language ID (see list on http://www.usb.org/developers/docs/)
- wLength=Descriptor Length

Multiple of these GET_DESCRIPTOR requests can be aggregated at one time in order to retrieve multiple descriptors.

Devices support one or more Configurations; a "bundle" of descriptors describing a particular set of functions which can be selected. Each of these descriptor sets can typically be up to 4 KB long. By using the Number of Configurations value it is possible to retrieve all available Configurations if required.

Devices may support multiple strings contained in an indexed table of strings. For each string there can be multiple languages supported. It is possible to retrieve all strings for a given language or all available strings if required.

A.2.3 Morphing devices

Some devices will determine heuristically the characteristics of the host operating system they are attached to and then present descriptors which are appropriate. After the descriptors have been obtained they will retain their settings until they are power cycled. This is particularly an issue for an MA USB hub which wants to present information on attached USB devices which may morph differently when interrogated by the MA USB hub and by the MA USB host. MA USB hubs should therefore power cycle USB devices once they have obtained descriptor information to present prior to secure connection, in order to ensure that the USB devices interoperate correctly on full enumeration.

Appendix B – WiGig specific requirements

B.1 Recommended MAC and PHY features for MA USB products using WiGig Certified radios

Table 72 lists some 802.11ad MAC and PHY features that vendors are recommended to provide for MA USB products using WiGig Certified radios. These features are not strictly required for MA USB operation but can result in improved performance or functionality.

NOTE — Mandatory MAC and PHY features for WiGig Certified radios are defined in Annex B (Protocol Implementation Conformance Statement (PICS) pro forma) of the IEEE 802.11ad standard [IEEE 802.11].

NOTE — Improvements in performance or functionality may depend on feature availability in both the MA USB host and a target MA USB device.

Table 72—Recommended 802.11ad MAC and PHY features for MA USB products

802.11ad PICS item	Protocol capability
MAC protocol capabilities	
PC39	Multi-band operation
PC39.3.1	Transmission of FST Tear down
QoS base functionality	
QB4.3.2	Extended Compressed Block Ack
QoS enhanced distributed channel access (EDCA)	
QD3	Multiple frame transmission support
DMG MAC features	
DMG-M4.4	Transmission of A-MPDU
DMG-M6	Reverse direction aggregation exchange
DMG-M7.3.2	CBAP allocation
DMG-M7.4.2.2	Support for four transmit queues with a separate channel access entity associated with each
DMG-M7.5.1	Scheduling of pseudo-static allocation
DMG-M10	DMG Block Ack with flow control
DMG-M11	DMG link adaptation
DMG-M14.2	STA power management with wakeup schedule
DMG-M14.3	PCP power management
DMG-M18	Changing DMG BSS parameters
DMG PHY features	
DBandP2.5.2.2	MCS 5-12

B.2 WiGig MA USB Protocol Constants

Table 73 lists the recommended MA USB protocol constants when implemented over the WiGig radio.

Table 73—MA USB protocol constants for WiGig

Protocol constant	Value (802.11 mode)	Value (IP mode)
aDataChannelDelay	25 msec	100 msec
aMaxIsochLinkDelay	25 msec	100 msec
aManagementChannelDelay	25 msec	100 msec

B.3 Synchronization in WiGig

WiGig devices shall use the transmission delay mechanism for synchronization.

MEDIA DEPENDENT NOTE — In 802.11 mode, if the MA USB packet is transmitted as part of a MAC-level A-MSDU aggregation and is not the first MSDU in the aggregation, the transmitter may choose not to initialize the Transmission Delay field in the outgoing packet. In this case, the transmitter shall set the MTD Valid subfield (in the I-Flags field or in SynchReq packet) to 0 to indicate an invalid Transmission Delay field.

MEDIA DEPENDENT NOTE — In 802.11 mode, if the receiving MA USB device determines that the MA USB packet may have experienced considerable latency (e.g., as a result of retransmission at the MAC layer), it may set the MTD Valid subfield in the I-Flags of the received packet to 0 to invalidate the Transmission Delay field.

B.4 WiGig implementation of L-managed OUT transfer

With WiGig radio, the reliable, in-order and flow-controlled delivery required for a link-managed transfer can be achieved by sending all the data-bearing packets belonging to the transfer (TransferReq packets for OUT transfers, TransferResp packets for IN transfers) through a dedicated Traffic Identifier (TID) with normal or block acknowledgement policy. In order to achieve the best performance, it is recommended to use the first acknowledgement policy from the following list that is supported by and available to both the MA USB host MAC and the target MA USB device MAC: (1) Block Ack with flow control, (2) any other Block Ack variant, and (3) Normal Ack.

For l-managed OUT transfers, the MA USB host PAL instructs the local MAC to set up a Block or Normal Ack agreement from the MA USB host MAC to the target MA USB device MAC on a dedicated Traffic identifier (TID). All TransferReq packets carrying the transfer payload shall be sent over the established session. Other packets may be sent using any control or data channel available to MA USB communication.

NOTE — If the session is a TC, the actual User Priority (UP) applied to the TransferResp packets in l-managed IN transfers, and TransferReq packets in l-managed OUT transfers may be imposed by TID availability.

The 4-bit Traffic Identifier (TID) designated for the l-managed transfer serves as the flow-controlled connection ID indicated in the TransferSetupReq and TransferSetupResp packets. TID is always selected by the MA USB host and communicated through the TransferSetupReq Packet. As part of the connection set up process, the MA USB host may create airtime allocations in the form of Service Periods (SPs) or Contention-based Access Periods (CBAPs), and one or more Traffic Streams (TSs), and map the TSs to the created allocations. When using SPs, packets sent using the Block Ack with flow control session (TransferResp packets for IN transfers, TransferReq packets for OUT transfers) may make use of the MAC-level Reverse Direction (RD) mechanism to increase the transmit opportunity for TransferReq or TransferResp packets in the opposite direction.

Figure 63 illustrates two l-managed OUT transfers over the WiGig link. The first OUT transfer goes through set up and data phases. The MA USB host receives a flow-control event in the middle of the transfer, and momentarily stops the flow of TransferReq packets. The flow is resumed once the flow control event clears, and ultimately the data phase completes with no error. In anticipation of more OUT transfers, the MA USB host keeps the resources allocated to the transfer in place, and starts the second

OUT transfer without going through the set up phase. The second transfer experiences a STALL condition on the target endpoint, which triggers the MA USB device to send a TransferResp packet reporting the error. The MA USB host in this case releases the resources allocated to the transfer and sends a TransferTearDownConf packet to the target MA USB device.

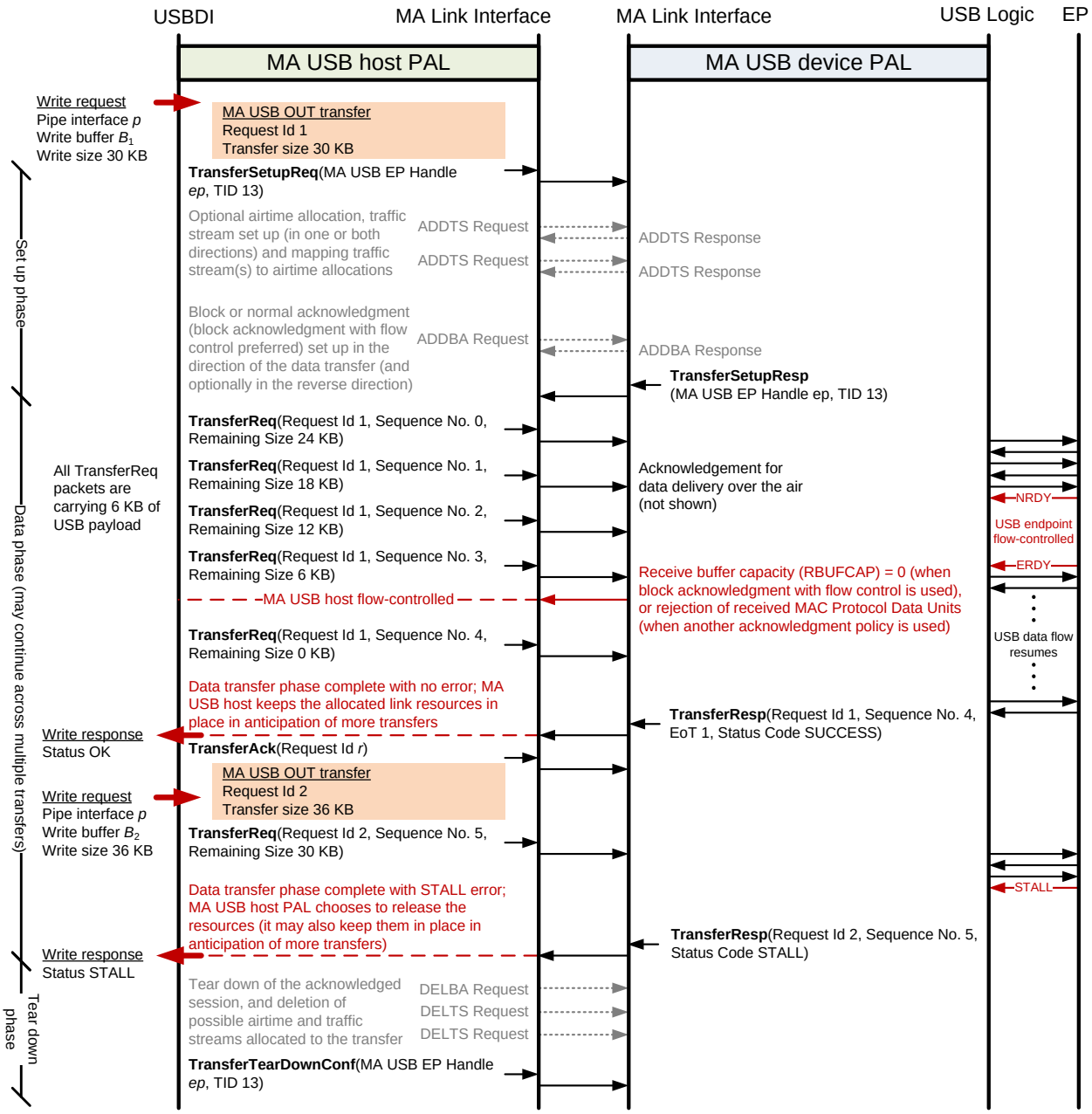


Figure 63—Link-managed OUT transfer in WiGig implementation