Wireless Universal Serial Bus Specification

Agere

Hewlett-Packard

Intel

Microsoft

NEC

Philips

Samsung

May 12, 2005

Revision 1.0

Scope of this Revision

The 1.0 revision of the specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

Revision	Date	Comments
0.9	December 29, 2004	Incremental update with significant additions throughout the major chapters, including finalization of isochronous header for data packets, definition of explicit device disconnect mechanisms and details on directed beaconing devices, device and host power management, updates to wire adapter isochronous streaming model and radio control interface, to name a few.
0.91 0.91a	January 21, 2005 January 22, 2005	Incremental update from comments on the 0.9. Of particular note is a better device state machine, new isochronous out examples and a security mechanisms overview in the data flow chapter. Revision 0.91a includes a 0.9b revision of the protocol chapter, which was omitted from 0.91 draft.
0.95	Feburary 24, 2005	Incremental update from comments to the 0.91a. Of particular note is a new 'active' disconnect detection model, the addition of explicit mechanisms for transmit power control, updates to the security and wire adapter chapters.
0.96	March 11, 2005	Incremental update from comments to 0.95. Significant updates to wire adapter and new commands added to the framework.
1.0 rc	March 31, 2005	Incremental updates to improve consistency and accuracy, better examples and improved readability.
1.0 rc2	April 27, 2005	Final updates of technical issues and pagination completed.
1.0	May 12, 2005	Final specification.

Please send comments via electronic mail to techsup@usb.org
For industry information, refer to the USB Implemnters Forum web page at http://www.usb.org

Acknowledgement of Wireless USB Technical Contribution

The authors of this specification would like to recognize the following people who participated in the Wireless USB Key Developers technical working groups. We would also like to thank other in the Wireless USB Promoter companies and throughout the industry who contributed to the development of this specification.

John S. Howard	Intel Corporation (Chair: Protocol/Editor)
John Keys	Intel Corporation (Chair: Security/Editor)
Dan Froelich	Intel Corporation (Chair: Isochronous/Editor)
Masami Katagiri	NEC Corporation (Chair: Wire Adapter)
David Thompson	Agere Systems, inc.
Nirmalendu Patra	Alereon, Inc.
Ed Beeman	Hewlett Packard
Brad Hosler	Intel Corporation (Editor: Architecture, Overview)
Abdul (Rahman) Ismail	Intel Corporation (Editor: Wire Adapter)
James J. Choate	Intel Corporation
Fred Bhesania	Microsoft Corporation
Randy Aull	Microsoft Corporation
Glen Slick	Microsoft Corporation
Mark Maszak	Microsoft Corporation
Masahiro Noda	NEC Corporation
Hiromitsu Sakamoto	NEC Corporation
Masao Manabe	NEC Corporation
Bart Vertenten	Royal Philips Electronics
Kawshol Sharma	Royal Philips Electronics
Hilbert Zhang	Royal Philips Electronics
Jay O'Conor	Royal Philips Electronics
Young Kim	Royal Philips Electronics
Takashi Sato	Royal Philips Electronics
Larry Taylor	Staccato Communications
Shyam Narayanan	Staccato Communications
Tim Gallagher	Staccato Communications
Bill Long	Staccato Communications
Valerio Filauro	STMicroelectronics
Matt Myers	Synopsys
Jin-Meng Ho	Texas Instruments
Sue Vining	Texas Instruments
Yaser Ibrahim	Texas Instruments
Haim Kupershmidt	Wisair Ltd.
Ran Hay	Wisair Ltd.

TABLE OF CONTENTS

Chapter 1 Introduction

1.1 Motivation

The original motivation for the Universal Serial Bus (USB) came from several considerations, two of the most important being:

- **Ease-of-use**
  The lack of flexibility in reconfiguring the PC had been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., did not have the attributes of plug-and-play.
- **Port expansion**
  The addition of external peripherals continued to be constrained by port availability. The lack of a bi-directional, low-cost, low-to-mid speed peripheral bus held back the creative proliferation of peripherals such as storage devices, answering machines, scanners, PDA's, keyboards, mice, etc. Existing interconnects were optimized for one or two point products. As each new function or capability was added to the PC, a new interface had been defined to address this need.

Initially, USB provided two speeds (12Mb/s and 1.5Mb/s) that peripherals could use. But as PCs became increasingly powerful and able to process vast amounts of data, users needed to get more and more data into and out of their PCs. USB 2.0 was defined in 2000 to provide a third transfer rate of 480Mb/s while retaining backward compatibility.

Since then, USB has arguably become the most successful PC peripheral interconnect ever defined. In 2005, analysts predict there will be over 500 million USB products in use. End users 'know' what USB is. Product developers understand the infrastructure and interfaces necessary to build a successful product. USB has gone beyond just being a way to connect peripherals to PCs. Printers use USB to interface directly to cameras. PDAs use USB connected keyboards and mice. The USB On-The-Go definition, provides a way for two host-capable devices to be connected and negotiate which one will operate as the 'host'. USB, as a protocol, is also being picked up and used in many non-traditional applications such as industrial automation.

Now, as technology innovation marches forward, wireless technologies are becoming more and more capable and cost effective. Ultra-WideBand (UWB) radio technology, in particular, has characteristics that match traditional USB usage models very well. UWB supports high bandwidth (480Mb/s) but only at limited range (~3 meters). Applying this wireless technology to USB frees the user from worrying about cables; where to find them, where to plug them in, how to string them so they don't get tripped over, how to arrange them so they don't look like a mess, … It makes USB even easier to use. Because no physical ports are required, port expansion, or even finding a USB port, is no longer a problem.

Of course, losing the cable, also means losing a source of power for peripherals. For self-powered devices, this isn't an issue. But for portable, bus-powered devices, Wireless USB presents some challenges where creative minds will provide innovative solutions that meet their customers needs.

USB (wired or wireless) continues to be the answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable interface that is consistent with the requirements of the PC platform of today and tomorrow.

1.2 Design Goals

Wireless USB is a logical evolution of USB. The goal is that end users view it as the same as wired USB, just without the wires. Several key design areas to meet this goal are listed below.

- Leverage the existing USB infrastructure. There are a large number of USB products being used today. A large part of their success can be traced to the existence of stable software interfaces, easily developed software device drivers, and a number of generic standard device class drivers (HID, Mass Storage, audio, etc.) Wireless USB is designed to keep this software infrastructure intact so that developers of peripherals can continue to use the same interfaces and leverage all of their existing development work.
- Preserve the USB model of smart host and simple device. Even though wireless technology introduces complexity, the Wireless USB architecture continues to have a significant split in responsibility between host and device. Wireless USB is designed to keep devices as simple as possible and let the host manage as much of the complexity as possible.
- Provide effective power management mechanisms. Without wires, many more traditional USB devices will have to run on batteries. Wireless USB is designed to allow devices to be as power efficient as possible, providing explicit times when radios need to be on so that radios can be in lower power modes otherwise.
- Provide security. Wireless USB is designed to provide a comparable amount of security to that which users enjoyed with wired USB. This translates to mechanisms to assure the user that their device is communicating only with their intended host and vice-versa. All data communications between host and device are encrypted to ensure privacy.
- Ease of use. This has always been a key design goal for all varieties of USB. Wireless USB is engineered to continue that tradition, while preserving strong security requirements.
- Investment preservation. There are a large number of PCs that support wired USB in use. There are a larger number of wired USB peripherals in use. Wireless USB defines a new USB device class, the Wire Adapter device class, that allows existing PCs to be 'upgraded' to include Wireless USB support, and that same device class allows wired USB devices to have a wireless connection back to the host PC.

1.3 Objective of the Specification

This document defines an industry-standard Wireless USB. The specification describes the protocol definition, types of transactions, bus management, and the programming interface required to design and build systems and peripherals that are compliant with this standard. This specification does not describe the underlying physical and MAC layers. These layers are defined in the PHY and MAC specifications, see [4] and [3]. This specification was written specifically targeting these sub-layer definitions, and the features of Wireless USB take specific advantage of the characteristics of the PHY and MAC Layer.

The goal is to enable wireless devices from different vendors to interoperate in an open architecture, while maintaining and leveraging the existing USB infrastructure (device drivers, software interfaces, etc.). The specification is intended as an enhancement to the PC architecture, spanning portable, business desktop, and home environments, as well as simple device-to-device communications. It is intended that the specification allow system OEMs and peripheral developers adequate room for product versatility and market differentiation without the burden of carrying obsolete interfaces or losing compatibility.

1.4 Scope of the Document

The specification is primarily targeted to peripheral developers and platform/adapter developers, but provides valuable information for platform operating system/ BIOS/ device driver, adapter IHVs/ISVs, and system OEMs. This specification can be used for developing new products and associated software.

Product developers using this specification are expected to know and understand wired USB as defined in the USB 2.0 Specification. Specifically, wireless USB devices must implement device framework commands and descriptors as defined in the USB 2.0 specification. Product developers may also need to know and understand aspects of the WiMedia MAC [3] and PHY [4] specifications depending on the type of Wireless USB product being developed.

1.5 USB Product Compliance

Adopters of the Wireless USB specification have signed the Wireless USB Adopters Agreement, which provides them access to a reasonable and non-descriminatory (RAND) license from the Promoters and other Adopters to certain intellectual property contained in products that are compliant with the Wireless USB specification. Adopters can demonstrate compliance with the specification through the testing program as defined by the USB Implementers Forum. Products that demonstrate compliance with the specification will be granted certain rights to use the USB Implementers Forum logos as defined in the logo license.

1.6 Document Organization

Chapters 1 through 3 provide an overview for all readers, while Chapters 4 through 8 contain detailed technical information defining Wireless USB.

- Peripheral implementers should particularly read Chapters 4 through 7.
- Host Controller implementers should particularly read Chapters 4 through 8.

Readers are also requested to contact operating system vendors for operating system bindings specific to Wireless USB.

This page intentionally left blank

Chapter 2 Terms, Conventions and References

2.1 Terms

ACK	Acknowledgment, usually in the context of a protocol handshake packet
ASIE	Application Specific Information Element
AES	Advanced Encryption Standard – FIPS publication 197
BER	Bit Error Rate (really low for wired environments, really high for wireless environments).
BOS	Binary device object store
BP	Beacon Period. Physical channel time during which the MAC Layer will transmit a beacon packet
BPOIE	Beacon Period Occupancy Information Element. This is an information structure defined in reference [3]. It is used in keeping track of members of a beacon period.
BPST	Beacon Period Start Time. The super-frame reference time at which a MAC Layer compliant device determines is the start of the super-frame.
PER	Packet Error Rate (also really low for wired environments, really high for wireless environments).
CC	Connection Context, including CHID,CDID and CK
CCM	Counter with CBC-MAC – A mode of operation built on AES
CDID	Connection Device ID
CHID	Connection Host ID
CK	Connection Key
CSMA/CA	Carrier sense multiple access with collision avoidance.
CTA	Channel Time Allocation
CRC	Cyclic Redundancy Check
DATA	Packet ID value indicating the associated packet is a data packet.
DID	Device ID, either CHID or CDID
DN	Device Notification
DNonce	Used in the definition of the four-way handshake to refer to a nonce generated by the device
DNTS	Device Notification Time Slot
DR	Data Receive; usually used in the context of a Wireless USB channel time slot during which a particular function endpoint is assigned to received transmissions from the host.
DRD	Dual-Role-Device

DRD-Device	Dual-Role-Device in its role of a Wireless USB Device
DRD-Host	Dual-Role-Device in its role of a Wireless USB Host
DRP	Distributed Reservation Protocol (part of the MAC Layer constructs)
DT	Data Transmit; usually used in the context of a Wireless USB channel time slot during which a particular function endpoint is assigned to transmit data packet(s).
DWA	Device Wire Adapter
Endpoint	A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device.
FS	Full Speed (defined in USB 2.0, reference [1]) as 12 Mb/s.
FSK	Fixed Symmetric Key
FFI	Fixed Frequency Interleaving
Frame	Typical nomenclature for communications protocols is that a frame is a packet of transmitted information. In Wireless USB the term Packet is used (see Packet), because in USB Frame is a specific term meaning a 1 millisecond time base for full and low-speed data communications.
GTK	Group Temporal Key
HDR	Shorthand for Header, usually in context of the Wireless USB application packet header
HNDSHK	Packet ID value indicating the associated packet is a protocol handshake packet.
HNonce	Used in the definition of the four-way handshake to refer to a nonce generated by the host
HS	High Speed (defined in USB 2.0, reference [1] as 480 Mb/s).
HWA	Host Wire Adapter; defined in this specification as a USB 2.0 connected Wireless USB Host Controller.
IAD	Interface Association Descriptor (defined in USB 2.0, reference [2]).
IDATA	Packet ID value indicating the associated packet is an Isochronous data packet with a defined data stream format.
IHV	Independent Hardware Vendor
ISO	Isochronous
ISV	Independent Software Vendor
IE	Information Element. A unique set of information that is included in its entirety in a data structure, such as a Beacon or MMC packet.
KCK	Key Confirmation Key
LLC	Logical Link Control
LQI	Link Quality Indication
MAC Layer	Media Access Control Layer. In this specification, the MAC Layer is specifically the MBOA MAC [3].
MAS	Media Access Slot; defined in reference [3]

MBOA	Multi-Band OFDM Alliance; industry special interest group promoting UWB physical and MAC layer standardization. Merged with WiMedia (see below).
MIC	Message Integrity Code (part of the MBOA secure packet encapsulation).
MIFS	Minimum Inter-frame Spacing. The minimum time between to successive transmitted packets. For burst-mode transfers, this is the exact required time between successive packet transmissions.
MMC	Micro-scheduled Management Command
MS-CTA	Micro-scheduled CTA
MSDU	MAC Service Data Unit. Information that is delivered as a unit between medium access control service access points.
MSTA	Micro-scheduled Time slot Allocation.
NAK	Negative Acknowledgement; usually in the context of a protocol handshake packet
Nonce	A term used by cryptographers to refer to an item that is used one time, such as a random number
OFDM	Orthogonal Frequency Division Multiplexing
OOB	Out-of-band
PAL	Programming Application Layer
PC	Personal Computer
PCA	Prioritized Contention Access
P2P-DRD	The DRD has a Point-to-point link with another DRD
PictBridge	A Direct Printing Protocol for Digital Still Camera with USB device function.
PER	Packet Error Rate
PID	Packet Identifier
Pipe	A logical abstraction representing the association between an endpoint on a device and software on the host.
PLCP	Physical Layer Convergence Protocol
PMK	Pairwise Master Key
PK	Public Key cryptography.
PHY	Physical layer. In this specification, the PHY is specifically the MBOA PHY [4].
PRF	Pseudo-Random Function
PTK	Pairwise Temporal Key
RAND	Reasonable And Non-Discriminatory (usually with regards to licensing intellectual property)
RC	Replay Counter
RCCB	Radio Control Command Block
RCEB	Radio Control Event Block

RPipe	Remote Pipe
RSSI	Received Signal Strength Indication
SC	Session Context, including CHID, CDID, Session Key and Security Frame Counter[SFC]
SFC	Secure Frame Counter
SFN	Secure Frame Number
SIFS	Short Interframe Spacing. The maximum allowed TX-to-RX or RX-to-TX turnaround time.
SK	Session Key
SOF	Start Of Frame. The first transaction in a USB 2.0 Frame or Micro-frame.
Slotted Aloha	A contention media access communications protocol technique for reducing the chance of collisions by multiple transmitters by dividing the channel into time slots and stating rules for how individual transmitters should select the slots for transmissions.
SME	Security management entity
SNR	Signal to Noise Ratio
SOF	Start Of Frame
STALL	Handshake code indicating an unrecoverable error on the function endpoint
Super Frame	The periodic time interval used in the MAC Layer [3] to coordinate packet transmissions between devices.
TDMA	Time Division Multiple Access
TF Code	Time/Frequency Code
TFI	Time Frequency Interleaving
TKID	Temporal Key Identifier (part of the MAC Layer [3] secure packet encapsulation).
TPC	Transmit Power Control
Transaction Group	Refers to the combination of MMC plus allocated protocol time slots (MSTAs) during which one or more Wireless USB transactions are conducted.
TrustTimeout	A timing threshold, measured from the reception of a successfully authenticated packet, after which a device or host must force a re-authentication before resumption of normal “trusted” data communications.
USB	Universal Serial Bus, usually in reference to USB 2.0.
UDR	Unused DRP Response; see reference [3]
UWB	Ultra-wideband , an emerging high data-rate radio standard.
WiMedia	Industry special interest group promoting UWB device standardization.
WUSB	Wireless USB
W_XCTA	Wireless USB channel allocation block; $_X$= DNTS, DT, DR

2.2 Conventions:

Figure 2-1 illustrates the convention for figures in Section 5.4 for illustrating the burst data phase protocol (explained later in this specification).

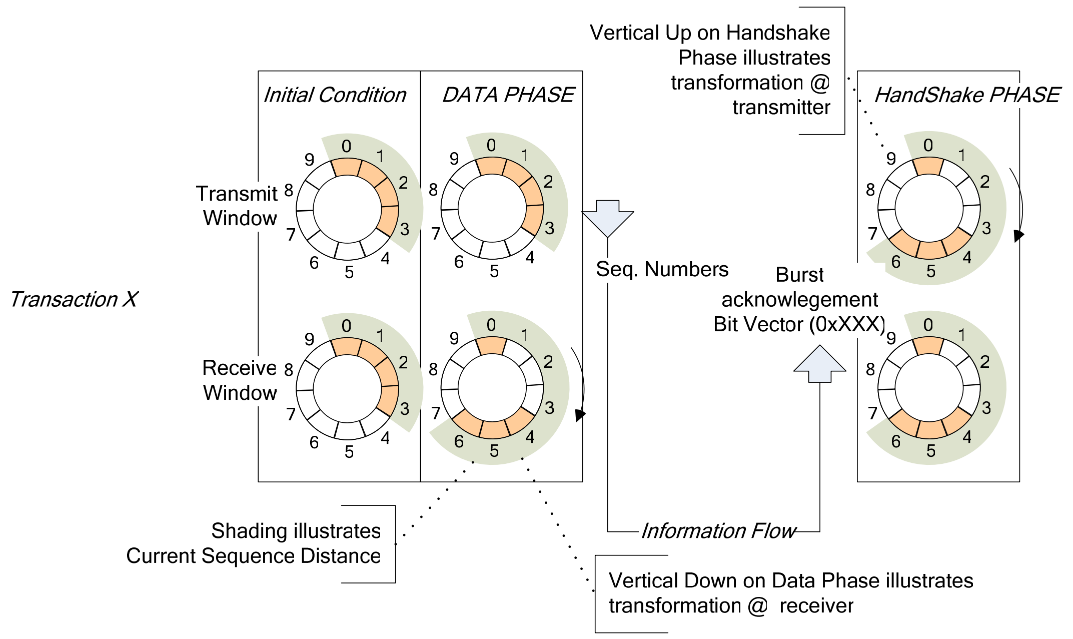
Vertical Up on Handshake Phase illustrates transformation @ transmitter
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Receive Window
Transaction X
Seq. Numbers
Burst acknowlegement Bit Vector (0xXXX)
Information Flow
Shading illustrates Current Sequence Distance
Vertical Down on Data Phase illustrates transformation @ receiver
0
1
2
3
4
5
6
7
8
9

Figure 2-1. Data Burst Transaction Convention

There may be more than one transaction per illustration/example. For each transaction, there is an initial condition of what is called the Transmit and Receive windows, illustrated as a number wheel, with shading in the spoke region indicating the current window. The numbers on the wheel represent the sequence numbers associated with the window locations. Shading on the outside of the wheel indicates the current distance between the current extremes of the sequence numbers in the current window (called sequence distance).

Figure 2-2 illustrates the conventions used in for the transaction diagrams in Chapter 5.

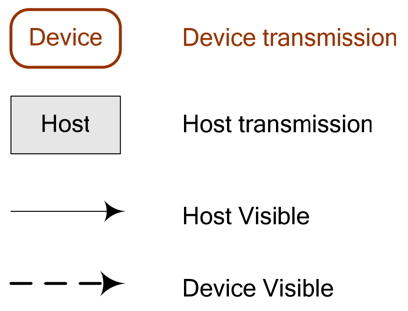

Figure 2-2. Transaction Diagram Conventions.

Light/Yellow shading/highlights in tables is used to illustrate standard/required portions of dynamic structures. If there is no highlighting, then the entire table contents are required.	🟨
If a table has only white and shaded portions, the shaded portion(s) indicate valid portion and the white indicates invalid portion(s). If there is no shading, then the entire table contents are valid values.	Invalid Value ▨ Valid Value

- Variable, field names and Device Notifications are *italicized.*
- Device states are **bold**.
- Numbers without a base indicator are in decimal. Non-decimal numbers have a base indicator appended to the value. The base indicators used in this specification are: (H - Hexi-decimal and B - Bindary). Note that some examples use a (0x) prefix base indicator for Hexi-decimal values.

2.3 References

[1] *Universal Serial Bus Specification* (Revision 2.0). April 27, 2000. Universal Serial Bus Implementers Forum (USBIF). Including all published Errata.

[2] *Interface Association Descriptor Engineering Change Notice* (Revision 1.0). July 23, 2003. This is an Engineering Change Notice to Universal Serial Bus Specification (Revision 2.0)

[3] *Distributed Medium Access Control (MAC) Specification for Wireless Networks* (Revision 1.0). 2005. WiMedia Alliance. This is also currently defined as the MBOA MAC.

[4] *Multiband OFDM Physical Layer Specification*. (Revision 1.) January 14, 2005. WiMedia Alliance. This is also currently known has the MBOA PHY specification.

[5] NIST FIPS Pub 197: *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T., November 16, 2001.

[6] NIST Special Publication 800-38C, Recommendation for Block Cipher Modes of Operation: *The CCM Mode for Authentication and Confidentiality*.

[7] *WiMedia MAC Convergence Architecture Specification* (Revsion 1.0). 2005. WiMedia Alliance.

Chapter 3
Architectural Overview

This chapter presents an overview of the Wireless USB architecture and key concepts. Wireless USB is a logical bus that supports data exchange between a host device (typically a PC) and a wide range of simultaneously accessible peripherals. The attached peripherals share bandwidth through a host-scheduled, TDMA-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation. Security definitions are provided to assure secure associations between hosts and devices, and to assure private communication.

Later chapters describe the various components of Wireless USB in greater detail.

3.1 USB System Description

A USB system consists of a host and some number of devices all operating together on the same time base and the same logical interconnect. A USB system can be described by three definitional areas:

- USB interconnect
- USB devices
- USB host

The USB interconnect is the manner in which USB devices are connected to and communicate with the host. This includes the following:

- Topology: Connection model between USB devices and the host.
- Data Flow Models: The manner in which data moves in the system over the USB between producers and consumers.
- USB Schedule: The USB provides a shared interconnect. Access to the interconnect is scheduled in order to support isochronous data transfers and to eliminate arbitration overhead.

USB devices and the USB host are described in more detail in subsequent sections.

3.1.1 Topology

Wireless USB connects USB devices with the USB host using a ‘hub and spoke’ model. The Wireless USB host is the ‘hub’ at the center, and each device sits at the end of a ‘spoke’. Each ‘spoke’ is a point-to-point connection between the host and device. Wireless USB hosts can support up to 127 devices and because Wireless USB does not have physical ports there is no need, nor any definition provided, for hub devices to provide port expansion. Figure 3-1 illustrates the topology of Wireless USB.

.

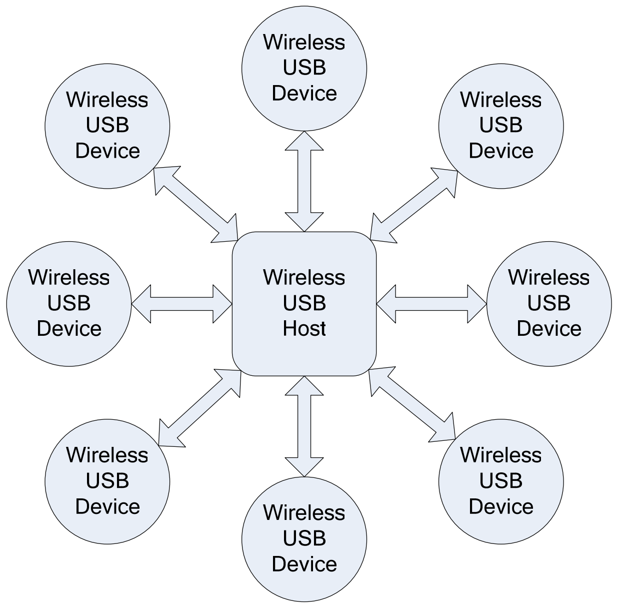

Figure 3-1 Bus Topology

3.1.1.1 USB Host

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller. Host controllers are typically connected to PCs through an internal bus such as PCI. The Host Controller may be implemented in a combination of hardware, firmware, or software.

This specification defines another way that a host controller may be 'connected' to a PC. Chapter 8 describes a Wire Adapter device class that allows USB host functionality to be connected to a PC through a USB connection (either wired or wireless).

Wire Adapters that directly connect to the PC using wired USB are known as Host Wire Adapters. Host Wire Adapters add Wireless USB capability to a PC.

Wire Adapters that are Wireless USB devices and hence connect to the PC wirelessly are known as Device Wire Adapters. Device Wire Adapters typically have USB 'A' connectors (ie. they look like wired hubs) and allow wired USB devices to be connected wirelessly to a host PC.

Note that each Wire Adapter creates a new 'USB system', in that there is one host (the wire adapter) talking to one or more devices using the same time base and interconnect.

Wire Adapters are important enabling devices for Wireless USB. Host Wire Adapters enable existing PCs to support Wireless USB. Device Wire Adapters allow existing wired USB devices to have a wireless connection to the host PC.

Additional information concerning hosts may be found in Section 3.10 and in Chapter 4.

3.1.1.2 Wireless USB Devices

Wireless USB devices are one of the following:

- Functions, which provide capabilities to the system, such as a printer, a digital camera, or speakers
- Device Wire Adapter, which provides a connection point for wired USB devices.

Wireless USB devices present a standard USB interface in terms of the following:

- Their comprehension of the Wireless USB protocol
- Their response to standard USB operations, such as configuration and reset

- Their standard capability descriptive information

Additional information concerning USB devices may be found in Section 3.9 and Chapter 4.

3.2 Physical Interface

The physical layer of Wireless USB is described in the Multiband OFDM Alliance (MBOA) UWB PHY specification, see reference [4]. The PHY supports information data rates of 53.3, 80, 106.7, 200, 320, 400 and 480 Mb/s and multiple channels. The PHY also provides appropriate error detection and correction schemes to provide as robust a communication channel as possible.

For Wireless USB devices, the support of transmitting and receiving data at rates of 53.3, 106.7, and 200 Mb/s is mandatory. The support for the remaining data rates of 80, 160, 320, 400 and 480 Mb/s is optional. Wireless USB Hosts are required to support all data rates for both transmission and reception. All Wireless USB implementations must support PHY channels 9 thru 15 (Band Group 1, all TF Codes). Implementations that support other Band Groups must support all TF Codes for any Band Group supported.

3.3 Power Management

A Wireless USB host may have a power management system that is independent of the USB. The USB System Software interacts with the host's power management system to handle system power events such as suspend or resume. Additionally, USB devices typically implement additional power management features that allow them to be power managed by system software.

This specification defines mechanisms and protocols that allow hosts and devices to be as power efficient as possible.

3.4 Bus Protocol

Logically, Wireless USB is a polled, TDMA based protocol, similar to wired USB. The Host Controller initiates all data transfers. Like wired USB, each transfer logically consists of three 'packets': token, data, and handshake. However, to increase the usage efficiency of the physical layer by eliminating costly transitions between sending and receiving, hosts combine multiple token information into a single packet. In that packet, the host indicates the specific time when the appropriate devices should either listen for an OUT data packet, or transmit an IN data packet or handshake (see Figure 3-2). Details of the WUSB protocol are provided in chapters 4 and 5.

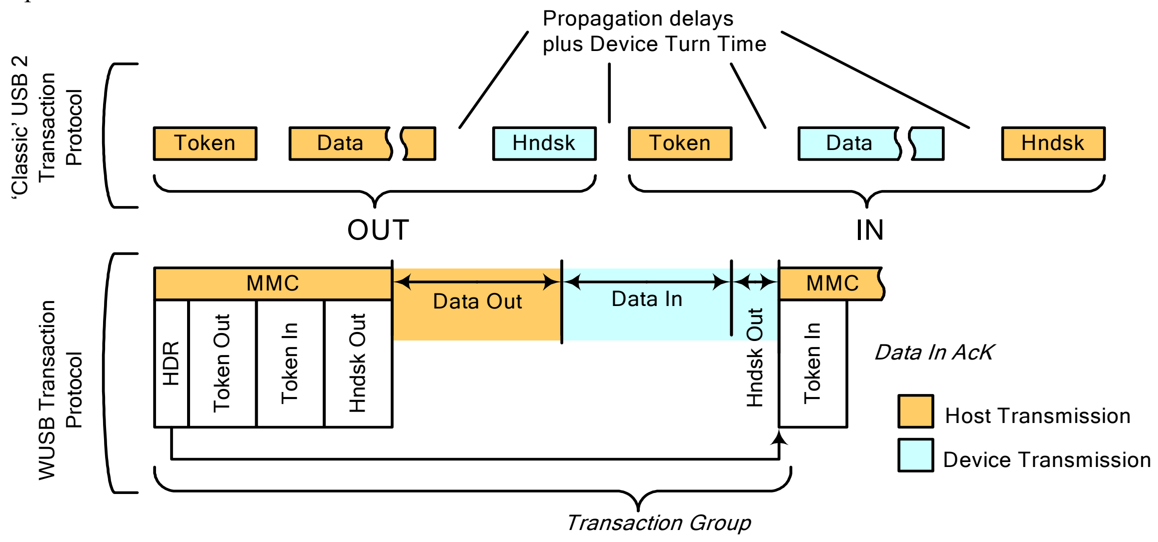

Figure 3-2. Wired to Wireless Protocol comparison

As in wired USB, the Wireless USB data transfer model between a source or destination on the host and an endpoint on a device is referred to as a pipe. Wireless USB defines new maximum packet sizes for some endpoint types to enhance channel efficiency. Similarly, some new flow control mechanisms are defined to enhance channel efficiency and to allow more power-friendly designs. New mechanisms are defined for isochronous pipes (see chapter 4) to deal with the lower reliability of the wireless medium.

3.5 Robustness

There are several attributes of wireless USB that contribute to its robustness:

- The physical layer, defined by [4], is designed for reliable communication and robust error detection and correction.
- Detection of attach and detach and system-level configuration of resources
- Self-recovery in protocol, using timeouts for lost or corrupted packets
- Flow control, buffering and retries ensure isochrony and hardware buffer management

3.5.1 Error Handling

The protocol allows for error handling in hardware or software. Hardware error handling includes reporting and retry of failed transfers. A Wireless USB Host will try a transmission that encounters errors up to a limited number of times before informing the client software of the failure. The client software can recover in an implementation-specific way.

3.6 Security

All hosts and all devices must support Wireless USB security. The security mechanisms ensure that both hosts and devices are able to authenticate their communication partner (avoiding man-in-the-middle attacks), and that communications between host and device are private. The security mechanisms are based on AES-128/CCM encryption, providing integrity checking as well as encryption. Communications between host and device are encrypted using ‘keys’ that only the authenticated host and authenticated device possess. Security details can be found in Chapter 6.

3.7 System Configuration

Like wired USB, Wireless USB supports devices attaching to and detaching from the host at any time. Consequently, system software must accommodate dynamic changes in the physical bus topology.

3.7.1 Attachment of Wireless USB Devices

Unlike wired USB, Wireless USB devices ‘attach’ to a host by sending the host a message at a well defined time. The host and device then authenticate each other using their unique IDs and the appropriate security keys. Details on device connections can be found in Chapter 4.

After the host and device have been authenticated and authorized, the host assigns a unique USB address to the device and notifies host software about the attached device.

3.7.2 Removal of Wireless USB Devices

Devices can be detached explicitly by either the host or device using protocol mechanisms. Device detach also happens when a host is not able to communicate with a device for an extended period of time.

3.7.3 Bus Enumeration

Bus enumeration is the activity that identifies and assigns unique addresses to devices attached to a logical bus. Because Wireless USB allows devices to attach to or detach from the logical bus at any time, bus enumeration is an on-going activity for the USB System Software. Additionally, bus enumeration for Wireless USB also includes the detection and processing of removals.

3.8 Data Flow Types

Wireless USB supports the same data transfer types and pipe types as wired USB. Basic usage and characteristics of the transfer types and pipe types are the same as wired USB. Because of the higher error rate characteristic of wireless communications, Wireless USB protocol defines different mechanisms for performing isochronous data transfers. These mechanisms include handshakes on data delivery as well as device specific amounts of buffering to allow devices some measure of control on the overall reliability of the isochronous pipe.

Bandwidth allocation for Wireless USB is very similar to wired USB.

Details of how the basic transfer types are implemented in Wireless USB can be found in Chapter 4.

3.9 Wireless USB Devices

Just like wired USB, Wireless USB devices are divided into device classes such as human interface, printer, imaging, or mass storage device. Wireless USB devices are required to carry information for self-identification and generic configuration. They are also required at all times to display behavior consistent with defined USB device states.

Notably, hubs are NOT a supported Wireless USB device class. Because Wireless USB hosts can support the architectural limit of 127 devices, there is no need for hubs. However, a new device class called Wire Adapter is defined. This device class describes a standard way for a device of one USB type (wired or wireless) to connect devices of the other type. A USB 2.0 connected Wire Adapter (known as a Host Wire Adapter) acts as the host for a Wireless USB system and provides a way to upgrade an existing PC to have Wireless USB capability. A Wireless USB Wire Adapter (known as a Device Wire Adapter) acts as a host for a wired USB system and allows wired USB devices to be connected wirelessly to a host PC. Figure 3-3 shows an example PC system including both a Host Wire Adapter and a Device Wire Adapter.

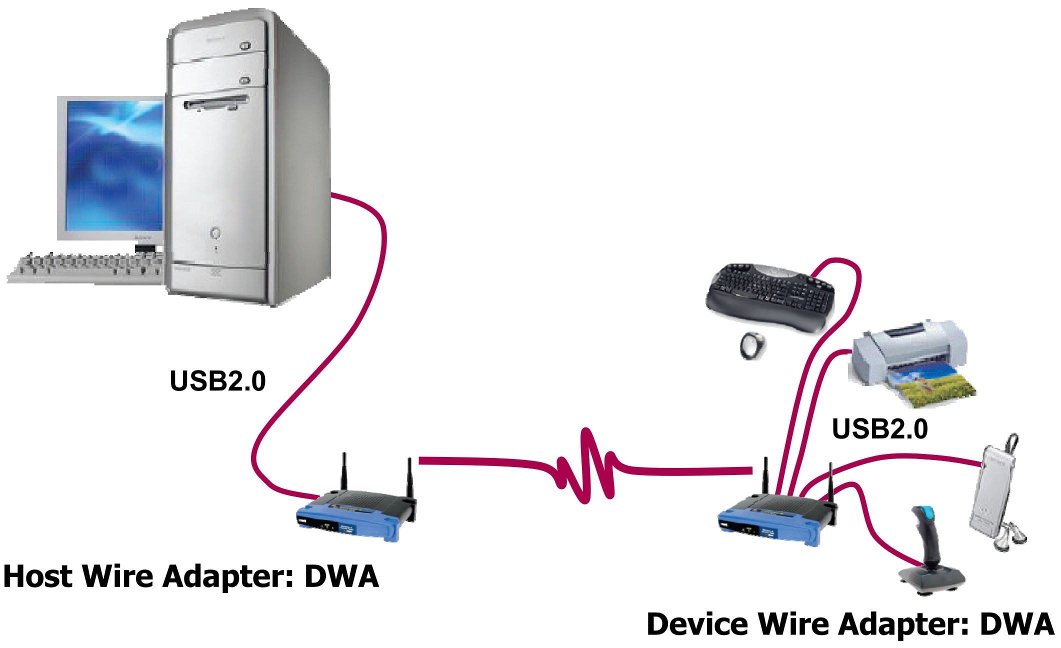

Figure 3-3. Wire Adapters

3.9.1 Device Characterizations

Like wired USB, all Wireless USB devices are accessed by a USB address that is assigned when the device is attached and enumerated. Each Wireless USB device additionally supports one or more pipes through which the host may communicate with the device. All Wireless USB devices must support a specially designated pipe at endpoint zero to which the USB device's USB control pipe will be attached. All Wireless USB devices support a common access mechanism for accessing information through this control pipe.

Associated with the control pipe at endpoint zero is the information required to completely describe the Wireless USB device. Standard descriptors for Wireless USB devices have been augmented (beyond those required for USB 2.0) to include the necessary information to support wireless communication. Detailed information about these descriptors can be found in Chapter 7.

3.9.2 Devices and MAC Layer

Wireless USB devices must also be well-behaved MAC Layer devices, see reference [3]. Wireless USB defines three categories of devices providing different degrees of awareness of MAC Layer mechanisms. The three categories are:

- Self beaconing device: These devices are fully aware of MAC Layer protocol and do all related beaconing.
- Directed beaconing device: These devices are unaware of MAC Layer protocol and rely on the host for direction to properly beacon and detect neighbor devices.
- Non-beaconing device: These devices have reduced transmit power and receiver sensitivity so that they don't interfere with, and are not interfered with, by neighbor devices that the host cannot detect.

More information on the behaviors of these devices can be found in Section 4.3.7 of the Data Flow chapter.

3.10 Wireless USB Host: Hardware and Software

The Wireless USB host has extended responsibilities beyond those of a wired USB host. The Wireless USB host must be a responsible MAC Layer device and ensure that all connected wireless USB devices also behave as responsible MAC Layer devices. This may mean helping devices to beacon properly and having devices relay beacon information to the host. Chapter 4 describes this behavior.

Many Wireless USB hosts will need mechanisms to share the UWB radio with other applications running on the host. For instance, on a standard PC the radio will be shared between the Wireless USB application and a wireless networking application. Rules and mechanisms for radio sharing are beyond the scope of this spec, but radio sharing is a required feature of Wireless USB hosts.

Wireless USB hosts are also responsible for coexisting with other UWB devices (including other Wireless USB hosts) in accordance with orderly rules for interference mitigation and bandwidth allocation. Wireless USB hosts follow policies set forth for these behaviors in the WiMedia WiMCA specification (see reference [7]). Wireless USB hosts are also responsible for managing the cluster such that Wireless USB device behavior follows the WiMCA policies. Wireless USB devices do not require explicit knowledge of the WiMCA policies.

In this specification, requirements for Wireless USB Hosts cover hosts that are implemented as part of standard PCs (notebooks, desktops, …). Wireless USB Hosts that are not standard PCs (portable devices, embedded hosts) may choose to implement a subset of the requirements. Defining the requirements for these ‘limited hosts’ is outside the scope of this specification.

This page intentionally left blank

Chapter 4
Data Flow Model

This chapter presents high-level information on how data and information moves across the Wireless USB 'Link'. The information in this chapter affects all implementers. The information presented is above the signaling and protocol definition(s) of the system. Consult Chapter 5 for details on the low-level protocol. This chapter provides framework overview information that is further expanded in Chapter 7. All implementers should read this chapter so they understand the key concepts of Wireless USB.

4.1 Implementer Viewpoints

Wireless USB is very similar to USB 2.0 in that it provides communication services between a Wireless USB Host and attached Wireless USB Devices. The Wireless USB communication model view preserves the USB 2.0 layered architecture and basic components of the communication flow (i.e. point-to-point, same transfer types, etc., See Section 5 in the Universal Serial Bus Specification Revision 2.0).

This chapter describes the differences (from USB 2.0) of how data and control information is communicated between a Wireless USB Host and its attached Wireless USB Devices. In order to understand Wireless USB data flow, the following concepts are useful:

- Communications Topology: Section 4.2 reviews the USB communications topology including differences in the physical topology from USB 2.0.
- Communication Flow Models: Section 4.3 defines the general mechanisms for accomplishing information exchanges, including data and control information, between a host and devices.
- Data Transfers: Section 4.4 provides an overview of how data transfers work in Wireless USB and subsequent sections define the operating constraints for each Wireless USB transfer type.
- Device Notifications: Section 4.9 provides an overview of Device Notifications, a feature which allows a device to asynchronously notify its host of events or status on the device.
- Media Reliability: Section 4.10 summarizes the information and mechanisms available in Wireless USB that a host might use to manage the reliability of the wireless data flows.
- Isochronous transfer model: Section 4.11 provides a detailed model for how isochronous data streams work over a Wireless USB channel.
- Connection Process: Section 4.13 outlines the basic connection process and introduces the basic mechanisms for getting devices connected to hosts.
- Security Mechanisms: Section 4.15 summaries the security features provided by Wireless USB.
- Power Management: Wireless enables mobility and mobility implies battery powered devices. Section 4.16 summarizes the power management model and features provided by Wireless USB.

4.2 Communications Topology

The general communications topology of Wireless USB is identical to that used in USB 2.0 (see Figure 4-1). The obvious advantage of this is that many existing USB 2.0 functional components (in hosts and devices) continue to work without modification when the physical layer components supporting USB 2.0 are replaced with those supporting Wireless USB. The delta change from USB 2.0 to Wireless USB is illustrated to the right-hand side of Figure 4-1. The *Function Layer* is (almost) completely the same. The only difference is the isochronous transfer model has some enhancements to allow functions to react to the increased unreliability of the "Bus Layer". The *Device Layer* includes a small number of framework extensions to support security (see below) and management commands required to manage devices on the wireless media. Finally, the *Bus Layer* includes significant changes to provide an efficient, secure communication service over a wireless media.

The copper wire in USB 2.0 provides significant value with regards to security of data communications. The User knows which host the device is associated with because the device has to be physically plugged into a receptacle and the wire provides a specific path for data communications flow between a host and devices that cannot be casually observed by devices not purposely connected. Replacing the physical layer copper with a radio results in ambiguity about the actual association between devices and hosts, and also exposes data communication flows to all devices within listening range. In other words, the loss of the wire results in a significant loss of security which must be replaced by other mechanisms in order for Wireless USB to be a viable and usable technology.

Wireless USB defines processes which allow a device and host to exchange the information required to establish a *Secure Relationship* (see Section 6.2.8). After a secure relationship has been established, the host and device have the necessary information required to support data encryption for "over the air" communications. Figure 4-1 illustrates how the standard USB data communications flow topology is extended for Wireless USB to include the concept of a secure relationship between a host and device and also illustrates that over-the-air data communications are encrypted. Notice that these new features extend only up to the device layer of the topology, allowing existing applications and device functions to exist and work without modification.

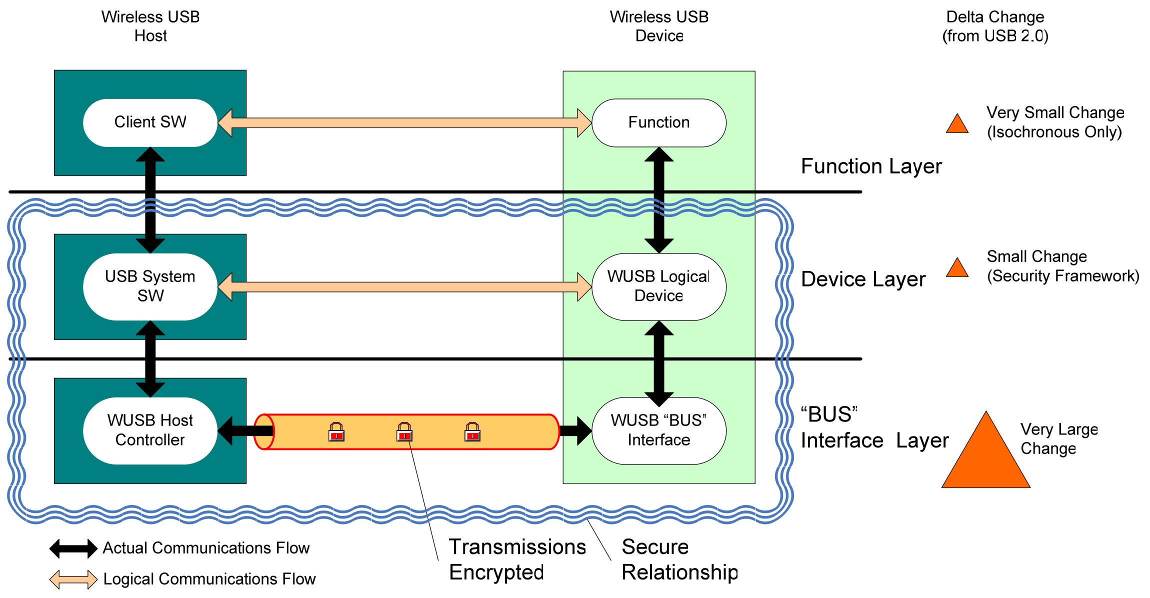

Figure 4-1. Wireless USB Data Communications Topology

Another side-effect of replacing the copper interconnect with a radio is that all low-level signaling events need to be provided mechanisms in the data flow topology that achieve equivalent functions. These include replacements for signaling events such as Connect, Disconnect and Resume.

4.2.1 Physical Topology

Wireless USB Devices are not physically *attached* to a Wireless USB Host. Devices within radio range of a host establish a secure relationship with the host before application data communications are allowed. A host and its associated devices are referred to as a Wireless USB Cluster. A Wireless USB Cluster is comprised of a Wireless USB Host and all the Wireless USB Devices that it directly manages.

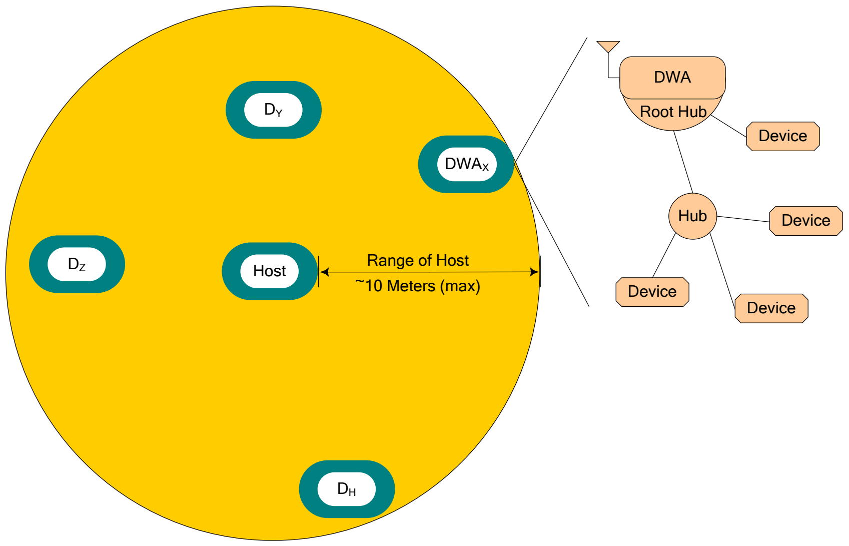

Figure 4-2. Physical Wireless USB Connection Topology

Figure 4-2 illustrates an example physical topology enabled by Wireless USB. The host has a radio range of about 10 meters. Devices within the host's range can establish a secure relationship with the host and become part of the host's Wireless USB Cluster. All communication flows between the host and devices are point-to-point which means the physical topology of Wireless USB is a 1:1 match with the defined logical communications topology familiar to USB architecture. Likewise the client software-to-function relationship remains unchanged (see Section 5.2 in the Universal Serial Bus Specification, Revision 2.0).

Wireless USB also defines a specific class of device called the Wire Adapter (see Chapter 8) that bridges between a Wireless USB bus and a USB 2.0 bus. The effect on the communications topology is essentially a cascading of USB busses.

4.3 Wireless USB Communication Flows

Wireless USB retains the familiar concepts and mechanisms and support for Endpoints, Pipes, and Transfer types, please refer to the USB Specification, Revision 2.0 for details. This section describes additions required to support Wireless USB. Subsequent sections cover co-existence, host and device requirements and an overview of the methods employed to manage data communications over a Wireless USB Channel.

A channel is a transmission path between nodes. The wireless physical layer (i.e. PHY) formats radio transmissions in a frequency range, via encoding and other techniques into a channel (or set of channels) through which basic bit streams are transmitted and received. A data link layer (on top of the PHY layer) encodes/decodes bit streams into/out of data packets, furnishes transmission protocol knowledge and management and handles errors in the physical layer, flow control and frame synchronization. The data link layer includes the MAC (Media Access Control) and LLC (Logical Link Control) for managing information over the physical channel. Application layers utilize the low-level channel services provided by the MAC and PHY.

Wireless USB utilizes the MAC Layer and PHY, which define several access methods for accessing the MAC Layer channel, including Beacons (for discovery and some distributed control) and Distributed Reservation Protocol (DRP - for TDMA type data communications). MAC Layer channel time is organized into super-frames as illustrated in Figure 4-3. Super-frames begin with a Beacon Period (BP) and are 65 milliseconds in duration. Super-frames are logically segmented into 256 Media Access Slots (MAS – each 256µs). The MASs at the beginning of a super-frame are allocated for use by the Beacon Period.

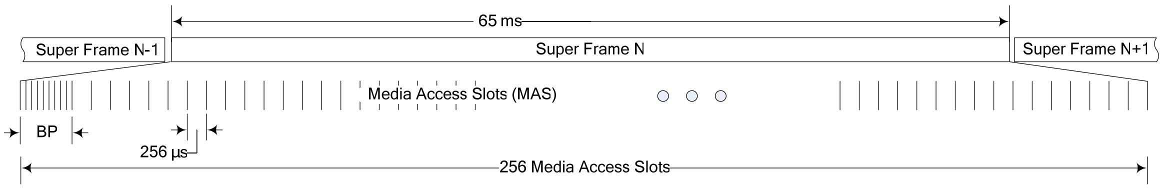

Figure 4-3. Basic Layout of Channel Time organization for a MAC Layer

Wireless USB defines a Wireless USB Channel which is encapsulated within a set of MAC Layer super frames via a set of MAC Layer MAS reservations (DRPs). The Wireless USB Channel is a continuous sequence of linked application-specific control packets, called MMCs (Micro-scheduled Management Commands), which are transmitted by the host within MAC Layer reservations (see Figure 4-4). MMCs contain host identifying information, I/O control structures and a time reference to the next MMC in the sequence (i.e. a link). These links provide a continuous thread of control which can be simply followed by devices that join the Wireless USB Cluster. This encapsulated channel provides the structure that serves as the transmission path for data communications between a host and devices in a Wireless USB Cluster.

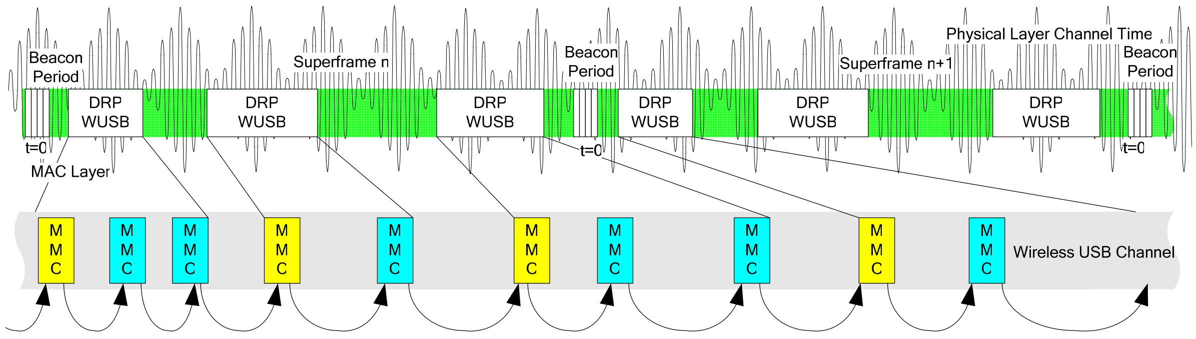

Figure 4-4. General Model of a Wireless USB Channel

The Wireless USB Channel is an efficient and extensible medium access protocol that provides low latency and fine grained bandwidth control. The Wireless USB Channel allows a host to rapidly and efficiently change the amount of channel time allocated to individual function endpoints. The Wireless USB Channel is used for both cluster broadcast and point-to-point data communications.

When establishing a Wireless USB Channel, the host allocates a *Broadcast Cluster ID* and uses it as the Destination *DevAddr* in describing the MAC Layer channel reservation (DRP IEs) and in the MMCs. These are the control mechanism for a host to manage the Wireless USB Channel. MMCs are used to broadcast command and I/O control information to all devices belonging to the Wireless USB cluster. MMCs are also used to communicate channel time allocations for point-to-point data communications, which are specifically between the host and individual function endpoints in the cluster. The addressing context for MMCs includes the host *DevAddr* and a *Broadcast Cluster ID*. The *Broadcast Cluster ID* for the Wireless USB channel is assigned by the host via a process that is appropriate to ensure the value is tightly coupled with the Wireless USB application and is unique for the lifetime of the Wireless USB Channel. The lifetime of a Wireless USB channel spans the time from when a host first starts the channel (i.e. after a boot or re-boot) to the next time the host needs to boot or reboot. The intent is that a host retains Wireless USB channel parameters across power states, PHY channel changes, etc. Refer to Section 4.13 for details about how a device identifies a host to connect to. The addressing context for point-to-point data communications includes the host DevAddr, the device DevAddr and a Stream Index value. A single stream index value is allocated by the host for each Wireless USB Channel.

A Wireless USB Channel consists of a continuous sequence of MMC transmissions from the host. The linked stream of MMCs is used primarily to dynamically schedule channel time for data communications between host applications and Wireless USB Endpoints. An MMC specifies the sequence of micro-scheduled channel time allocations (MS-CTAs) up to the next MMC within a reservation instance or to the end of a reservation instance. It may be followed by another MMC without the existence of MS-CTAs between the two MMCs. In this case, the MMC is only used to convey command and control information. The channel time between two

MMCs may also be idle time, where no MS-CTAs are scheduled. The general layout of the MMC is defined below with detailed information elements for the MMC defined in Sections 5.2 and 7.5.

The MS-CTAs within a reservation instance can only be used by the devices that are members of the associated Wireless USB Cluster. The direction of transmission and the use of each MS-CTA is fully declared in each MMC instance. An MMC can declare an MS-CTA during any channel time following the MMC. Section 5.2 provides detailed requirements for using MMCs to accomplish the Wireless USB protocol via the Wireless USB channel. Figure 4-5 illustrates an example MMC sequence within an instance of a MAC Layer channel reservation for Wireless USB.

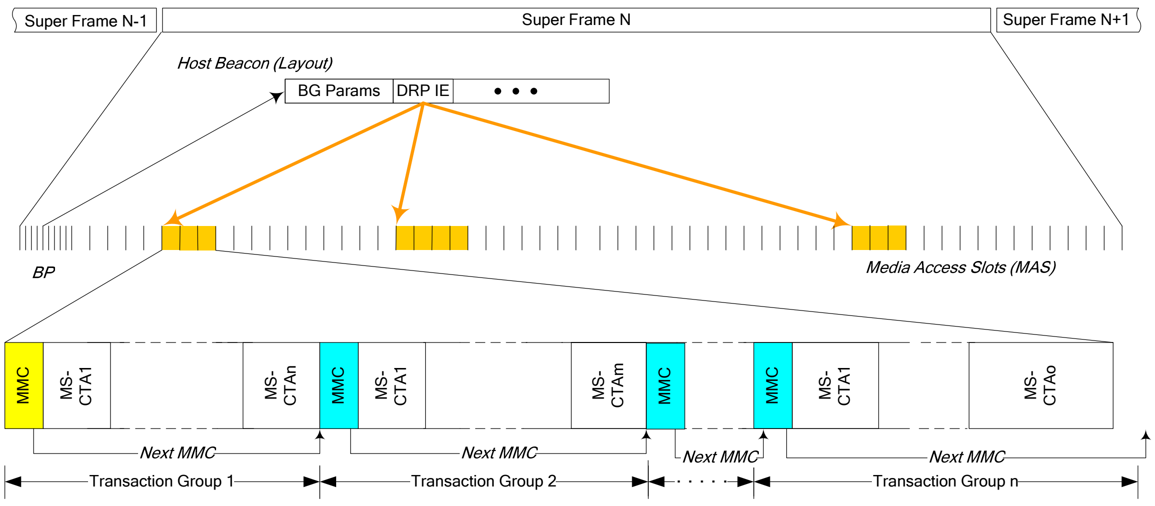

Figure 4-5. Example Wireless USB Channel Control Sequence within a MAC Layer Reservation

An MMC contains the information elements necessary to identify the Wireless USB Channel, declare any MS-CTAs, or other information elements that are used for command and control. The general structure of an MMC packet is defined in Section 7.5 and the exact structure of the information elements contained in an MMC is defined in Sections 7.5 and 5.2.1. The MMC is a broadcast control packet that is for receipt only by devices that are members of the Wireless USB cluster. The host must use the Broadcast Cluster ID value in the *DestAddr* field of an MMC packet's MAC header. This technique identifies this packet transmission as a broadcast targeting all devices in a Wireless USB cluster, and avoids potential confusion at Non-Wireless USB devices in listening range of the host. The MMC data payload must be encapsulated within a MAC Layer secure packet; however its data payload is transmitted in plain text, thus using the security encapsulation for authentication purposes only.

A host is required to implement the MAC Layer protocol, establish and maintain (one or more) Wireless USB Channels by allocating sequences of MAC Layer MAS reservations (i.e. DRPs). A device may implement the full MAC Layer protocol; however it is nominally only required to implement the Wireless USB protocol which operates within the Wireless USB Channel.

This section reviews a small subset of MAC features that are relevant to Wireless USB. It is not intended to provide a working knowledge of the MAC Layer and also uses many terms defined in the MAC Layer standard, please refer to reference [3] for full details. Section 4.3.8 describes the requirements on a host for establishing and maintaining a Wireless USB Channel, both for data communications within the Wireless USB Cluster and interactions with other MAC Layer devices. Section 4.3.7 describes the minimum requirements on a device to accomplish data communication over the Wireless USB Channel. This specification includes all information required to implement Wireless USB Devices. For the host, it contains only the information required to implement a Wireless USB Host; it does not include all information required for a compliant MAC Layer implementation.

4.3.1 Wireless USB Channel Time

Identical to USB 2.0, a Wireless USB Host maintains a free running timer that effectively defines USB channel time. USB channel time can be used by isochronous devices and is used by Directed Beaconing devices and is also utilized by several other features of Wireless USB. Accuracy of the USB channel time must be 20ppm (matching PHY requirements).

Current USB channel time is communicated by the host in MMCs. Each MMC contains a 24-bit timestamp value that indicates when (in USB channel time) the MMC was transmitted. The timestamp consists of two parts, a 1/8th millisecond value and a microsecond value. The microsecond counter is 7 bits, and counts from 0 to 124, and then wraps back to zero. The 1/8th millisecond value is 17 bits and wraps to zero after reaching a value of all 1's. The 1/8th millisecond value increments when the microsecond value wraps from 124 to 0. The timestamp indicates the time when the beginning of the MMC packet was transmitted.

1/8th millisecond counter (17 bits)	microsecond counter (7 bits)

Figure 4-6. MMC Time Stamp

There are no requirements on a specific start value when the host first starts a USB Channel. However, once the timestamps are started they must continue to be provided based on the free-running timer as long as the host is producing MMCs. USB channel time must not be affected by changes in the host's beacon start time or any other event during normal operation.

4.3.2 MMC Transmission Accuracy

The host must ensure that MMC transmissions begin on microsecond boundaries with +/-40 nanosecond accuracy and that the timestamp value in the MMC accurately reflects the USB channel time. For example – it would be incorrect for a host to produce an MMC with the current time stamp and then wait 2 microseconds to send the MMC.

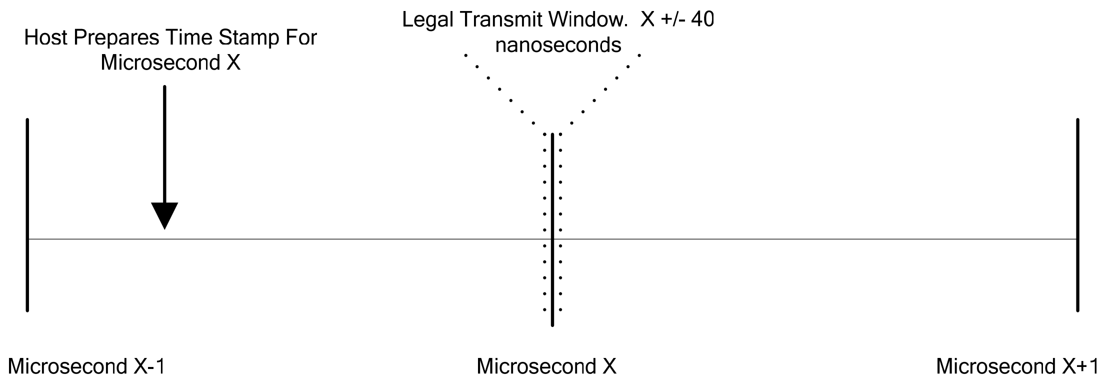
Host Prepares Time Stamp For Microsecond X
Legal Transmit Window. X +/- 40 nanoseconds
Microsecond X-1
Microsecond X
Microsecond X+1

Figure 4-7. MMC Time Stamp Accuracy

Figure 4-7 shows a timescale for the free running timer in a host. The host intends to send an MMC when its free running timer reaches microsecond X. The host has a small window of 40 nanoseconds before to 40 nanoseconds after microsecond X on its free running timer to start transmission of the MMC. If the MMC transmission starts earlier or later, the host has not met the accuracy requirement for the time stamp in MMC.

4.3.3 USB Time across Device Wire Adapters

For a device wire adapter USB channel time must be consistent (same rate and same value) across its upstream bus and its downstream bus. A device wire adapter is required to synchronize its downstream SOF packets with the USB channel time on its upstream wireless connection. The DWA generated *FrameNumber* values in SOFs must match the 1/8th millisecond values in MMC timestamps on the upstream wireless bus.

As an example, consider Figure 4-8 where the top part of the figure illustrates the upstream and downstream buses on a DWA. Bus A is a Wireless USB bus connecting the Host PC to a Device Wire Adapter (DWA). Bus B is a USB 2.0 hi-speed connection between the DWA and a hi-speed device. The bottom part of the figure shows a sequence of time when the USB channel time generated by the host is crossing a millisecond boundary. The vertical dotted lines in the figure show 1/8th millisecond points of USB channel time. The DWA is required to make the transmission times of the SOFs on the downstream bus be consistent with the upstream channel time as illustrated in the figure.

The *FrameNumber* value of the SOFs on the downstream wired bus must match bits 13:3 of the 1/8th millisecond value in the timestamp of any MMCs transmitted during a 1/8th millisecond period. Similarly, bits 2:0 of the 1/8th millisecond value in the MMC timestamp must reflect the SOF instance (microframe) on the downstream bus.

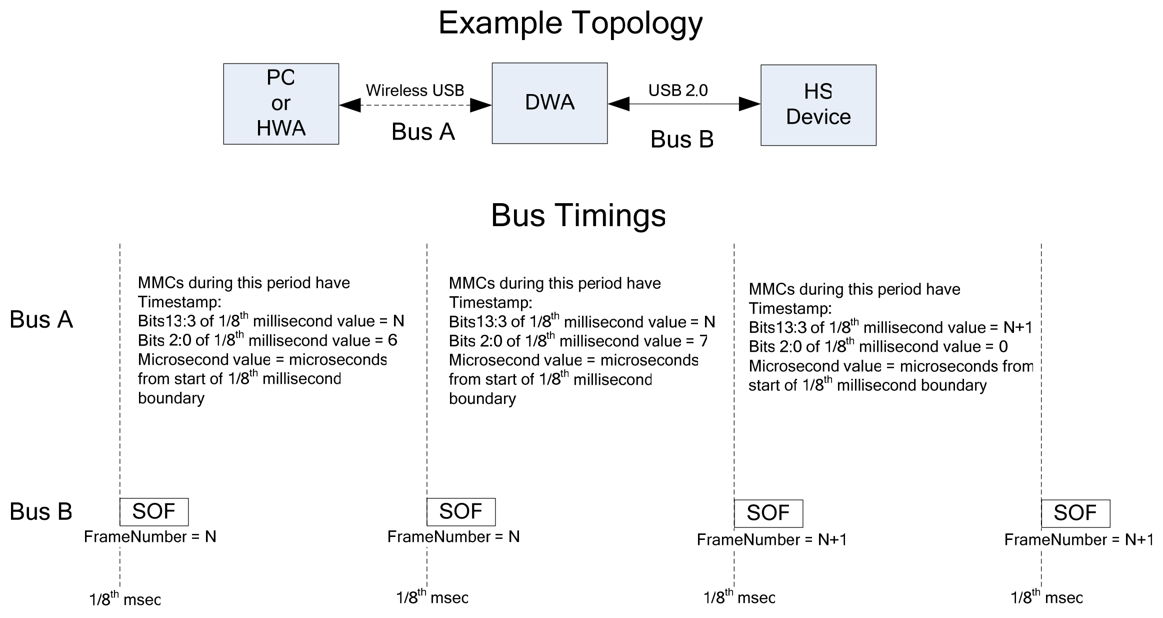
Example Topology
PC or HWA
Wireless USB
Bus A
DWA
USB 2.0
Bus B
HS Device
Bus Timings
Bus A
MMCs during this period have Timestamp: Bits13:3 of 1/8th millisecond value = N Bits 2:0 of 1/8th millisecond value = 6 Microsecond value = microseconds from start of 1/8th millisecond boundary
MMCs during this period have Timestamp: Bits13:3 of 1/8th millisecond value = N Bits 2:0 of 1/8th millisecond value = 7 Microsecond value = microseconds from start of 1/8th millisecond boundary
MMCs during this period have Timestamp: Bits13:3 of 1/8th millisecond value = N+1 Bits 2:0 of 1/8th millisecond value = 0 Microsecond value = microseconds from start of 1/8th millisecond boundary
Bus B
SOF
FrameNumber = N
SOF
FrameNumber = N
SOF
FrameNumber = N+1
SOF
FrameNumber = N+1
1/8th msec
1/8th msec
1/8th msec
1/8th msec

Figure 4-8. USB time across hierarchical buses

Note, HWAs are not required to synchronize their downstream USB channel time with their upstream USB 2.0 channel time (i.e. SOFs). HWAs must meet all Wireless USB channel time accuracy requirements.

4.3.4 Device and Application Co-existence

The Wireless USB Channel is mapped onto **reserved** MAC Layer channel time, so the host is required to satisfy the MAC Layer protocol beaconing requirement. This means hosts must manage two-hop topology in order to respect declared reservations from other devices that carry reservation declarations (DRP IEs) in their beacon data structures. DRP IEs are used to reserve MAC Layer channel time for the Wireless USB Channel and the host must utilize MAC Layer beacons to propagate the Wireless USB reservations to neighbors of the host and individual devices (see Section 4.3.8). Note that protection of Wireless USB reservations requires that under most circumstances a host and devices from the Wireless USB Cluster transmit beacons during the Beacon Period. Figure 4-9 illustrates an example.

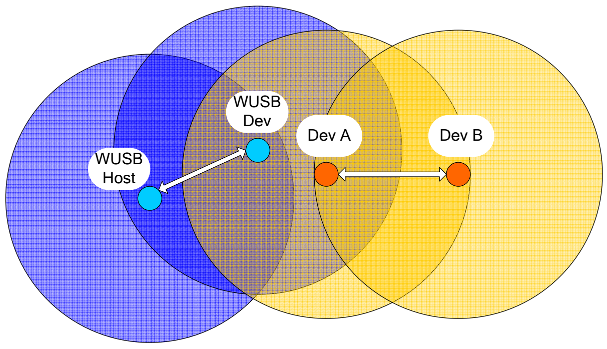

Figure 4-9. Example Two-Hop Topology

The host (WUSB Host) and device (WUSB Dev) communicate via the Wireless USB Channel. The device has neighbors Dev A and Dev B which are not neighbors of the host (i.e. are not in range of the host, so therefore cannot successfully receive the host's Beacon). Therefore, in order to protect the Wireless USB Channel reservation, the device needs to transmit a Beacon that includes a DRP IE similar to that of the host. Dev A will observe the device's beacon and become aware of the Wireless USB Channel reservation and thus avoid transmitting during the advertised MAC Layer channel time.

A Wireless USB device may choose to be fully MAC Layer aware. These device implementations must meet the MAC Layer standards for participating in the MAC Layer channel, including managing beacon period synchronization and processing neighbor beacons. As noted above devices can choose to implement only support for communicating via the Wireless USB Channel, which means they have no knowledge of the encapsulating MAC Layer protocol. If all devices in the Wireless USB cluster have the same neighbors as the host, then only the host needs to transmit a beacon (with a DRP IE(s)) to reserve the MAC Layer channel time for the Wireless USB channel. However, when devices have neighbors that are different (than what the host observes), they need to transmit a beacon which can be observed by its neighbors that are not neighbors of the host. Wireless USB accomplishes this by providing mechanisms that allow a host to perform proxy processing on behalf of the non-MAC Layer protocol devices.

Wireless USB defines protocol mechanisms to allow a host to direct devices to capture and propagate MAC Layer channel protocol information without having knowledge of the MAC Layer channel protocol (see section 4.3.7.2 describing Directed Beaconing devices). These mechanisms are used by a host to command Wireless USB (only) devices to capture and/or transmit raw packets at specific times, relative to the Wireless USB Channel time. Using these mechanisms, a host can command devices to capture frames during the beacon period and forward them to the host for processing. The host can accumulate this information and then determine whether the device has neighbors that are out of range of the host. When a host determines that a device needs to transmit a MAC Layer beacon, the host constructs the device beacon, forwards it to the device via a standard Wireless USB request on the Default Control Endpoint, then directs the device to transmit the stored beacon packet at a Wireless USB Channel time that occurs co-incident with an open beacon slot time. The requirements on the host for managing Directed Beaconing devices are detailed in Section 4.3.8.

Over time, the host will encounter situations when it has too much or too little MAC Layer channel time reserved for the current Wireless USB Channel communications load. There are two basic forms of over-budgeting which need to be handled: Short-term and Long-term. The definition of a Short-term over-budget situation is where the host has no scheduled transfers for the Wireless USB Channel. When this occurs, the host may release the remainder of the current MAC Layer reservation by instructing all affected devices in the cluster to transmit a UDR (Unused DRP Response) control packet. MAC Layer compliant devices that hear the UDR can then use the remainder of the MAC Layer channel reservation time. A short-term over-budget condition does not cause any modifications to the MAC Layer channel reservations for the Wireless USB Channel, as declared in the beacon data structures transmitted by devices from the Wireless USB Cluster. A Long-term over-budget scenario can result in a modification to the MAC Layer channel reservation time as declared in cluster's beacons. For example, the host may detect that it has much more MAC Layer channel time required than the aggregate communication load requires. It can therefore release a portion of its reservation by reducing the number of reserved MASs declared in its DRP IEs for the Wireless USB Channel.

Whenever a host detects an under-budget scenario, it can either keep the current MAC Layer channel reservation (and endure the performance (throughput) impact), or it may grow its MAC Layer channel reservation appropriately in order to provide better service for the duration of the increased Wireless USB Channel communication load. Modifications to the overall MAC Layer channel reservation are accomplished via changing the DRP IE information for the Wireless USB Channel in both the host's and Cluster device's Beacons.

4.3.5 Device Endpoints

Wireless USB preserves the device Endpoint as the terminus of a communication flow between a host and a device. Wireless USB Endpoint characteristics are extended from the wired counterparts, in particular to support security and efficiency. As with USB 2.0, all Wireless USB devices must implement at least the Default Control Pipe (Endpoint zero). The Default Control Pipe is a Control pipe as defined in the USB 2.0 specification and is available once a device has completed the initial connection data exchange (see Sections 4.3.8 and 4.3.7).

4.3.6 Wireless USB Information Exchange Methods

The types of information exchanges between a host and its associated devices via the Wireless USB Channel are characterized by three basic functional buckets: host transmitted control information, asynchronous device transmitted control information and Wireless USB transaction protocol (between a host and function endpoints). Wireless USB defines the following methods of information exchange:

- **Wireless USB Device Notification Time Slots**. Wireless USB allocates specific 'management' channel time (Device Notification Time Slots (DNTSs)) for asynchronous, device initiated communications. This asynchronous upstream (i.e. Device to Host) communication is used for signaling connect, remote-wake and other events that are analogous to the wired signaling events that occur in wired USB. It is also used as a general purpose device-to-host notification mechanism through which devices can transmit asynchronous command, status and request messages to host. This channel time may only be used for Wireless USB Device Notification Messages as defined in Sections 5.5.3 and 7.6.
- **Broadcast Control Information.** All Wireless USB data communications occur within the Wireless USB Channel using a host-scheduled protocol. A host transmits control packets called Micro-schedule Management Commands (see Section 7.5 for the definition of MMCs), which contain control information to devices, including acknowledgements to Device Notifications and general purpose Wireless USB Cluster management.
- **Wireless USB Transactions**. Wireless USB data communications between the host and function endpoints utilize a transaction-based communication protocol similar to the USB 2.0 transaction protocol.

Most packets transmitted between a host and devices in the Wireless USB cluster are encapsulated in MAC Layer secure packets. The only exceptions to this rule pertain to those data exchanges required to establish initial association (i.e. before a secure relationship is established).

4.3.7 Device Perspective

A Wireless USB Device must implement all of the required features of the Wireless USB protocol (see Chapter 5) in order to communicate via a Wireless USB Channel. The Wireless USB Channel is encapsulated by the MAC Layer channel, so it is possible that Wireless USB devices can be implemented that don't know about the MAC Layer channel. It is also possible to add MAC Layer compliant device capability to a Wireless USB Device. A device always uses the Wireless USB Channel and the communication protocol defined in this specification to connect and communicate within a particular Wireless USB Cluster. A host can request a device's MAC-layer channel capabilities so that the host will know whether it needs to manage MAC Layer mechanisms for the device (see section 4.3.8.4).

Section 4.13 details a device's requirements to participate in the Wireless USB Cluster Connection process.

A device that implements the full MAC Layer protocol is called a 'Self Beaconing' device. A device that is not aware of the MAC Layer channel and instead relies on the host to help it be a good MAC Layer citizen is called a Directed Beaconing Device. Devices that are unaware of the MAC Layer channel and have restricted transmit and receive capabilities such that they don't need to beacon are called Non Beaconing Devices. Hosts identify the class of a device from the *Beacon Behavior* field of the Wireless USB Device Capabilities – UWB descriptor (see Section 7.4.1.1). These three classes of devices are described in more detail below.

4.3.7.1 Self Beaconing Devices

A device that implements the full MAC Layer protocol is called a 'Self Beaconing' device and it knows how to manage the MAC Layer channel including synchronization and maintenance of MAC Layer Beacon periods. A Self Beaconing device must support several Wireless USB device requests in order to allow the host to manage DRP reservations.

Self Beaconing devices must be able to determine which MAS slots are available for communication with the host. All DRP reservations seen by the device, except the reservations that comprise the Wireless USB channel reservation, must be excluded from the devices MAS availability information. Section 7.7 summarizes the value settings for DRP IEs for the host and cluster members that are beaconing. The device identifies the host's DRP IE based on the following keys:

- *Reservation Type* field is **Private**
- *Stream Index* field has the value of the Wireless USB channel's stream index. This is derived from the MAC Header *Delivery ID* field.
- *DevAddr* field set to the channel's Broadcast Cluster ID.

The device identifies a cluster member's DRP IE based on the following keys:

- *Reservation Type* field is **Private**
- *Stream Index* field has the value of the Wireless USB channel's stream index. This is derived from the from the MAC Header *Delivery ID* field.
- *DevAddr* field set to the host's *DevAddr*.

A host uses the GetStatus(MASAvailability) request to retrieve a device's MAS availability information. A host uses the SetWUSBData(DRPIEInfo) request to provide the device with the appropriate DRP IE to include in the device beacon. The host uses a SetFeature(TX_DRPIE) request to instruct the device to start including the provided DRP IE in its beacon. A Self-beaconing device may detect reservation collisions (as defined in the MAC Layer specification) and can notify the host of a MAS Availability information change using the *DN_MASAvailChanged* notification. A host uses a ClearFeature(TX_DRPIE) request to instruct a device to cease transmitting the DRP IE in its beacon. Details on all of these requests can be found in Section 7.3.

4.3.7.2 Directed Beaconing Devices

A device that only implements the Wireless USB channel must support Wireless USB defined capabilities that allow a host to learn about neighbors of the device that are out of range from the host, but in range of the device (called hidden neighbors), and to have the device transmit a beacon defined by the host. These capabilities are required for several reasons, including ensuring the MAC Layer reservations (DRPs) are observable to hidden neighbors so they can be respected. Devices that support these capabilities are called 'Directed Beaconing' devices.

Directed beaconing devices must support three functions: Transmit Packet, Count Packets, and Capture Packet. Hosts must ensure that devices are never enabled for more than one of these functions at a time.

4.3.7.2.1 Transmit Packet Function

The Transmit Packet function is typically used to have the directed beaconing device transmit a beacon packet at regular intervals. In order to do this, the device needs to know what packet data to transmit and when to transmit that packet.

The host provides the complete packet data that the device transmits using the SetWUSBData(Transmit Data) standard device request (see Section 7.3.1.5). The information provided to the device is the complete MAC header and payload (without FCS). The device is responsible for building a properly constructed packet with the provided MAC header and payload. Directed beaconing devices must have 200 bytes of buffering to store the transmit packet data.

To tell the device when to transmit the packet, the host provides two values to the device. The first value is a 24-bit Transmit Time. The second value is an 8-bit Transmit Adjustment. When enabled for directed packet transmission, the device will first send the transmit packet when USB channel time matches the host provided Transmit Time. After transmitting the first packet, the device will continue to transmit the packet every 64K (2^{16}) + Transmit Adjustment microseconds as long as it is enabled for directed packet transmission. The host provides the necessary time values using the SetWUSBData(Transmit Params) standard device request (see Section 7.3.1.5). A host must not send a SetWUSBData(TransmitParams) command to a device that is already enabled for directed packet transmission. A host can change the transmit adjustment value to all devices currently enabled for directed packet transmission by adding a Transmit Packet Adjustment IE to transmitted MMCs. When a device receives a Transmit Packet Adjustment IE, it will use the new Transmit Adjustment value for all intervals after the next Transmit Time occurs.

A host enables a device for directed packet transmission by sending the device a SetFeature (DEV_XMIT_PACKET) request. Hosts are responsible for initializing the transmit packet data and time values before enabling a device for directed packet transmission. Devices are disabled for directed packet transmission after reset and when they receive a ClearFeature (DEV_XMIT_PACKET) request. See the Section 7.3.1 for complete descriptions of these requests. Devices are also disabled for directed packet transmission anytime the device goes to sleep (see Section 4.16.1.1) or exits the **Authenticated** device state.

4.3.7.2.2 Count Packets Function

The Count Packets function causes the directed beaconing device to receive for a period of time and have the device record how many packets were received, store some basic packet information, and record when the packets were received. Hosts can use this functionality to have the device scan other PHY channels to see if the device sees any activity on that channel, or to scan during a beacon period to see if the device sees any beacons that the host doesn't see.

The host uses SetWUSBData(Receive Params) standard request to provide the device with the USB channel time window, bounded by *Receive Start Time* and *Receive End Time*, during which the device should receive. The host also provides a PHY channel parameter telling the device which channel to receive on, and a packet filter parameter, Receive Filter, that the device uses to match and record particular packets. See Section 7.3.1.5 for details on this standard request and its parameters.

A host enables a device for counting packets by sending the device a SetFeature (COUNT_PACKETS) request. Hosts are responsible for appropriately initializing the device with a SetWUSBData(Receive Params) request before enabling a device for directed packet transmission. Devices are disabled for counting packets after reset, when they have completed a count packet function, when the capture buffer is full, or when they receive a ClearFeature (COUNT PACKETS) request. See the 7.3 for complete descriptions of these requests.

When enabled for counting packets, the device will logically go through the following steps:

- When USB channel time matches *Receive Start Time*, the device begins trying to receive packets. The device will continue to try to receive packets until the USB channel time matches *Receive End Time*, at which time the device will disable itself for counting packets.
- For every correctly received header, the device will extract the Frame Type field from the Frame Control field of the MAC header and generate an 8-bit value with one set bit that corresponds to the

value of the Frame Type field. For example, if the Frame Type field has a value of three, then bit 3 of the generated value will be set. The device then does a logical AND of this generated value with its Receive Filter value and if the result is non-zero then the device proceeds to the following steps. If the logical AND of those fields is zero, then the device waits for the next packet.

- For packets that pass the filter, the device records the channel time when the packet was received (time marker is the beginning of the preamble), the first six bytes of the MAC header (*Frame Control*, *DestAddr*, *SrcAddr*), and the LQI of the received packet.

Devices must have 512 bytes of receive data buffer space to store count packet information. Note that this buffer can be the same buffer used for the Capture Packet function. This buffer cannot be the same as the transmit packet buffer.

A host can retrieve the recorded information by sending the device a GetStatus (Received Data) request. In addition to the recorded information, the device will indicate how many packets were recorded and if the receive data buffer space was filled before the count packet function was complete. The device will return the recorded information in the format described in Section 7.3.1.2.

4.3.7.2.3 Capture Packet Function

The Capture Packet function causes the directed beaconing device to try to receive and store a single packet during a specified period of time. The common host usage for this is to have a device try to receive and store a packet during a particular MAC Layer beacon slot so that the host can see the full beacon contents.

The host uses SetWUSBData(Receive Params) standard request to provide the device with the USB channel time window, bounded by *Receive Start Time* and *Receive End Time*, during which the device should receive. The host also provides a PHY channel parameter telling the device which channel to receive on, and a packet filter parameter, Receive Filter, that the device uses to match and record particular packets. See Section 7.3.1.5 for details on this standard request and its parameters. Note that this interaction is identical to that done when counting packets.

A host enables a device for capturing a packet by sending the device a SetFeature (CAPTURE_PACKET) request. Hosts are responsible for appropriately initializing the device with a SetWUSBData(Receive Params) request before enabling a device to capture a packet. Devices are disabled for capturing a packet after reset, when they have captured a packet that matches the filter criteria, when the receive period has terminated, or when they receive a ClearFeature (CAPTURE PACKET) request. See Section 7.3 for complete descriptions of these requests.

When enabled for capturing a packet, the device will logically go through the following steps:

- When USB channel time matches *Receive Start Time*, the device begins trying to capture a packet. The device will continue to try to capture a packet until the USB channel time matches *Receive End Time* or a packet is captured. After USB channel time matches *Receive End Time* or capturing a packet that matches the specified filter criteria, the device will disable itself for capturing packets.
- For every correctly received header, the device will extract the Frame Type field from the Frame Control field of the MAC header and generate an 8-bit value with one set bit that corresponds to the value of the Frame Type field. For example, if the Frame Type field has a value of three, then bit 3 of the generated value will be set. The device then does a logical AND of this generated value with its Receive Filter value and if the result is non-zero then the device proceeds to the following steps. If the logical AND of those fields is zero, then the device waits for the next packet.
- For a packet that passes the filter, the device records the channel time when the packet was received (time marker is the beginning of the preamble), stores the MAC header and payload of the packet, and stores the LQI of the received packet.

Devices must have 512 bytes of receive data buffer space to store capture packet information. Note that this buffer can be the same buffer used for the Count Packets function. This buffer cannot be the same as the transmit packet buffer.

A host can retrieve the packet data by sending the device a Get Status (Received Data) request. In addition to the stored information, the device will indicate whether or not the packet was successfully received and if the captured packet was larger than the receive data buffer space. The device will return the information in the format described in Section 7.3.1.2.

4.3.7.3 Non Beaconing Devices

Wireless USB also defines what is called a "Non-Beaconing" device, which due to reduced transmit power and receiver sensitivity, can only exist close to its host. So close in fact that any neighbor of this device will be in range of the host and thus be able to observe the host's beacon before the Non-beaconing device can interfere with the neighbor's traffic. Characteristics of the type of device are not specified in this revision of the specification.

4.3.7.4 Selecting A Wireless USB Host

Wireless USB devices use information transmitted in a host's MMCs to select which host to connect to. See Section 4.13, Connection Process, for details. Devices may have to scan several PHY channels before finding the appropriate host. Depending on user preferences and device capabilities, a device may choose to automatically connect to a host, or wait for a user to instruct the device to make a connection (possibly by pushing a button).

4.3.8 Host Perspective

When a Wireless USB Host becomes active, it must choose a PHY channel in which to operate the Wireless USB channel. Once the host is beaconing it then establishes a Wireless USB Channel by reserving MAC Layer channel time for Wireless USB data communications.[1] Figure 4-10 illustrates the relationship between the host's Beacon and an example set of MAC Layer channel time reservations.

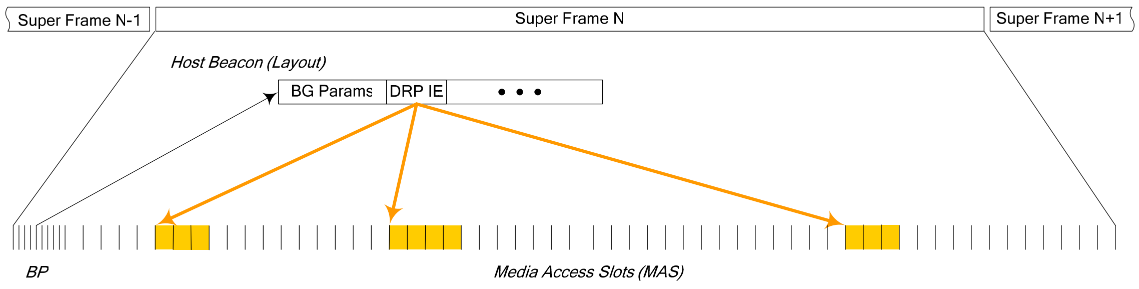

Figure 4-10. Wireless USB Application-specific Host Information Element in Beacon

Figure 4-10 illustrates an example DRP allocation for a Wireless USB Channel. The reservation of MAS slots for a Wireless USB Channel depends on the application load. To understand how the Wireless USB Channel is established and maintained, the host functionality can be separated into three logical entities as shown in Figure 4-11 e.g. Host System Management, Wireless USB Host and MAC Layer compliant device which are described in the following sections.

[1] Note that allocation of channel time must eventually take into consideration information from devices joining the Wireless USB Channel.

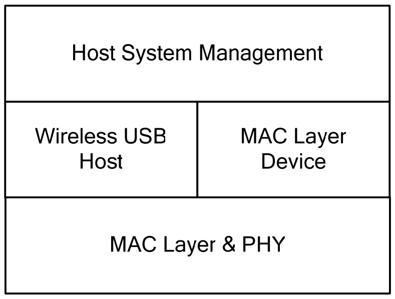

Figure 4-11. Wireless USB Host logical components

By separating the Wireless USB Host and MAC Layer compliant device roles, it is easier to describe the establishment and maintenance of the Wireless USB Channel which encapsulates (but is independent) of the operation of the Wireless USB protocol.

4.3.8.1 MAC Layer Compliant Device

The primary MAC Layer channel management operation is the generation and exchange of Beacon Frames between MAC Layer compliant devices, which carry the channel use and MAC Layer compliant device identification information elements. The MAC Layer compliant device implements the MAC Layer rules which enable the MAC Layer channel to be shared between a set of MAC Layer compliant devices in radio range. It implements the MAC Layer protocol in particular the generation and interpretation of the MAC Layer beacon frames which are the principle means by which the Wireless USB Channel permissions are obtained since they carry the DRP reservation Information Elements. The MAC Layer compliant device will inform Host System Management of any DRP conflicts which may arise in the operation of the MAC Layer protocol (owing to mobility or other effects changing the topology of the MAC Layer beacon group, or any traffic change from the MAC Layer compliant devices in range).

The MAC Layer compliant device implements MAC Layer superframe synchronization where adjustments to the superframe timing are made for the slowest clock in the neighborhood. These adjustments are signaled to Host Management which in turn may have to work with the Wireless USB Host to adjust Directed Beaconing devices.

The MAC Layer protocol requires hidden neighbor effects to be mitigated by the exchange of neighbor information in beacons. The Host Management can utilize features in both the MAC Layer compliant device and Wireless USB Host in managing the two-hop neighbor topology so that the Wireless USB channel reservations are appropriately respected by all neighbors of devices in the Wireless USB Cluster.

4.3.8.2 Wireless USB Host

The Wireless USB Host entity implements the host roles of schedule generation and maintenance of the Wireless USB Channel. It is responsible for scheduling data communications on the Wireless USB Channel between itself and Wireless USB devices belonging to the Wireless USB cluster. The host must ensure that it does not schedule Wireless USB channel communications to cross the boundary of a permitted MAC Layer channel access period (i.e. DRP reservation). Figure 4-12 illustrates a view of how the Wireless USB Host operation simplifies this into a series of contiguous (but disjoint in time) time intervals separated by MMCs.

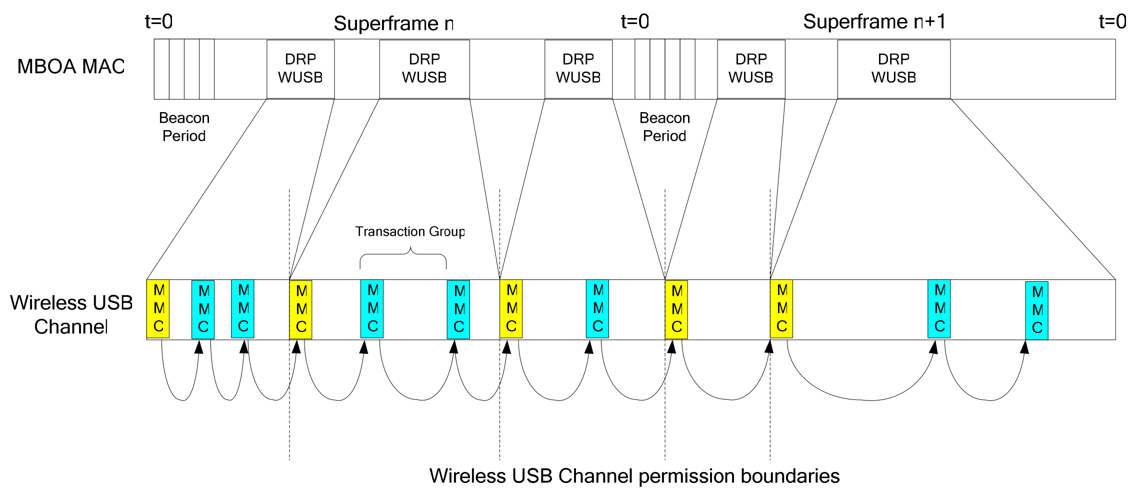

Figure 4-12. Example Map of Wireless USB Channel to MAC Layer Channel Reservation Boundaries

The logical end of a Transaction Group (as bounded by the mapping onto the MAC Layer Channel in time) must be managed by the Wireless USB host scheduler so that they do not violate MAC Layer channel time structures.

4.3.8.3 Host System Management

Host system management controls the interactions and information flow between the Wireless USB Host and MAC Layer compliant device modules, including providing the mapping of the Wireless USB Channel on the PHY via the establishment and maintenance of a series of DRP reservations. Many of the possible functions of this module are beyond the scope of this specification. However it is important to recognize that this module can utilize the services provided by the host and MAC Layer compliant device modules to accomplish the behavior required of a Wireless USB Host and the devices in its cluster. It bridges the Wireless USB Host and MAC Layers such that requirements of each can be converted into structures recognized by the other, for example it can map MAC Layer time onto Wireless USB channel time so that the host time references used by the Wireless USB devices won't violate the reservation boundaries.

The policy decisions on how to manage MAC Layer channel time for the Wireless USB channel are implemented in a host-specific manner within this module, including how long to hold MAC Layer reservations, when to expand them, when to reduce and when to release them.

4.3.8.4 Managing Two-Hop Reservation Topology

If all the neighbors of a Wireless USB device are also neighbors of the host, then the host's beacon is sufficient to maintain the integrity of the Wireless USB Channel and respect of neighbor's announced DRP reservations. When the host's beacon provides coverage for all the devices in its Wireless USB Cluster, there is no benefit for devices in the cluster to transmit MAC Layer Beacons.

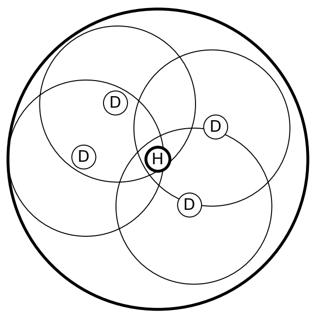

Figure 4-13 Host covers all Wireless USB device

Periodically, the host must check with all members of its Wireless USB Cluster to ensure the neighborhood has not changed. If new MAC Layer compliant devices have entered the radio range of a Wireless USB cluster member device but are not in the range of the host, a Hidden neighbor situation exists, as illustrated in Figure 4-14 where M is hidden from H and could interfere with traffic at the D(s) in its range.

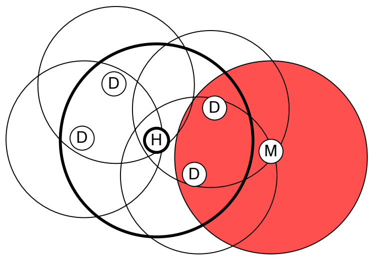

Figure 4-14 Hidden Neighbor

The host uses the capabilities of Self Beaconing and Directed Beaconing devices to gather information about neighbors in their range and to propagate host beacon information to hidden neighbor devices. Only Directed Beaconing device members of the Wireless USB cluster which have neighbors hidden from the host need to be instructed to transmit and receive packets (e.g. beacons in this particular case).

The host uses the mechanisms described in section 4.3.7.1 to gather reservation information about hidden neighbors from Self Beaconing devices. If the MAS availability information returned by the device is not a superset of the host's current Wireless USB channel DRP, then it knows there is at least one hidden neighbor and the host may need to adjust its DRP reservations to avoid conflicts.

The host uses the mechanisms described in section 4.3.7.2 to gather similar information from Directed Beaconing devices. The typical process would be that the host first has the device 'Count Packets' during the beacon period. The host can then determine if there are any hidden neighbors. If there is a hidden neighbor, the host has the device 'Capture Packet' at the appropriate beacon time to get the full beacon of the hidden neighbor. The host can then adjust its DRP reservation if needed.

To direct a device to transmit a beacon, the host uses the Transmit Packet mechanisms defined in section 4.3.7.2.1 to provide the device with a beacon data structure with DRP IEs appropriate for the devices in the neighborhood of the hidden neighbor, as well as a report of the local beacon group constructed to inform the hidden neighbors(s) of the host and its neighbors. The host also provides the device channel time information so that the device will transmit the beacon during the appropriate beacon slot period for multiple MAC Layer superframes.

4.3.8.5 Other Host Considerations

If a host does not have any active data flows when a reservation instance begins, it may inform its neighbors that the current reservation time is available for other communications. The MAC Layer standard defines an explicit method for releasing a reservation. The method is based on participants in the reservation transmitting a specific control packet which can be observed by neighbors. A Wireless USB reservation instance is released by the host instructing relevant members of a Wireless USB Cluster to transmit the UDR control packet. Note that the host must transmit the UDR control packet also, see Section 7.5.6. After a DRP instance release operation is complete, the host must not use the DRP instance for Wireless USB communications.

Table 4-1 summarizes how MAC *DevAddr* values are allocated by a host to manage devices through the process of admittance to a Wireless USB Cluster (see Section 4.13 for details). It is the responsibility of the host to avoid *DevAddr* value conflicts within its Wireless USB Cluster.

Table 4-1. Summary of how MAC Layer DevAddr Address Space is used for Wireless USB

Address Tag	Range	Explanation
MAC Layer Generated, Multicast and Broadcast DevAddr Range	256-65535 (0100H-FFFFH)	Wireless USB Hosts must have a full 64-bit MAC Address from which a 16-bit DevAddr is generated. The MAC Layer also assigns addresses in the upper portion of this range to Multicast and Broadcast DevAddrs.
UnConnected_Device_Address	255 (00FFH)	A Wireless USB device will use this DevAddr value for its Wireless USB DevAddr when it is in the UnConnected device state.
UnAuthenticated_Device_Address_Range	128-254 (0080H-00FEH)	A Wireless USB Host may assign a connecting device a Device Address in this range in response to a *DN_Connect* notification. It will also choose an address in this range to serve as the Broadcast Cluster ID.
WUSB_Device_Address_Range	0-127 (0000H-007FH)	The host will assign a device a Device Address in this range as part of the normal enumeration process.

4.4 Data Transfers

Wireless USB preserves all of the basic Data Flow and Transfer concepts defined in USB 2.0, including the Transfer Types, Pipes and basic data flow model. The differences with USB 2.0 are enumerated below, starting with description of differences at the protocol level, then the differences in transfer type constraints.

The USB 2.0 specification utilizes a serial transaction model. This essentially means that a host starts and completes one bus 'transaction' {Token, Data, Handshake}, before starting the next transaction. 'Split' transactions also adhere to this same model as they are comprised of complete high-speed transactions {Token, Data, Handshake} that are completed under the same model as all other transactions.

Wireless USB maps the USB 2.0 transaction protocol onto the TDMA Micro-scheduling feature. The result is that the Wireless USB transaction protocol is essentially a split-transaction protocol that allows more than one 'bus transaction' to be active on the bus at the same time. The split-transaction protocol scales well (across multiple transactions to multiple function endpoints) with signaling bit-rates as it is not completely subject to propagation delays. The basic USB protocol is recognizable within the Wireless USB split transaction architecture, however there are modifications to certain aspects of the protocol in order to reduce or hide some protocol overheads.

Figure 4-15 illustrates the high-level differences between the USB 2.0 "one at a time" transaction protocol and the basic structure of the Wireless USB protocol.

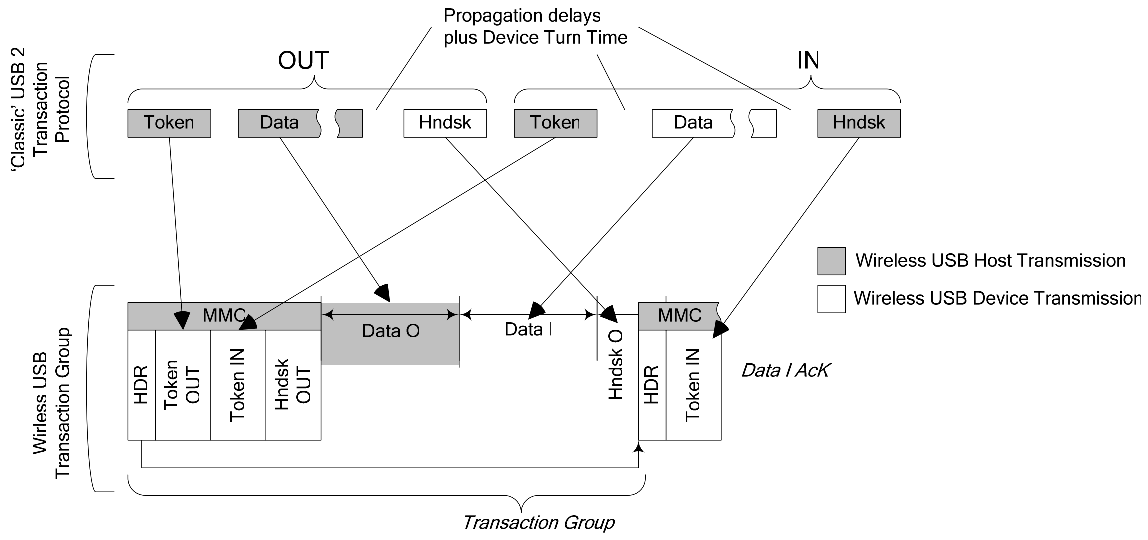
Propagation delays plus Device Turn Time
OUT
IN
'Classic' USB 2 Transaction Protocol
Token
Data
Hndsk
Token
Data
Hndsk
Wireless USB Host Transmission
Wireless USB Device Transmission
Wirless USB Transaction Group
MMC
Data O
Data I
MMC
HDR
Token OUT
Token IN
Hndsk OUT
Hndsk O
HDR
Token IN
Data I AcK
Transaction Group

Figure 4-15. USB 2.0 vs Wireless USB Transaction Footprints

The USB 2.0 protocol completes an entire IN or OUT transaction (Token, Data and Handshake phases) before continuing to the next bus transaction for the next scheduled function endpoint. The Wireless USB protocol broadcasts USB Token (equivalents) in the MMC and utilizes TDMA time slots for the Data and Handshake phases as appropriate for the transfer type and direction of data communication. Utilizing this method, a host can 'start' a group of transactions at the same time (e.g. because the MMC may contain 'Tokens' for more than one Wireless USB transaction). Within the context of the Wireless USB application, the Micro-scheduled sequence (e.g. MMC plus associated time slots) is called a *Transaction Group*. A Wireless USB Host determines how individual transactions are scheduled into individual transaction groups in order to satisfy the needs (and priorities) of the applications controlling the devices in the Wireless USB Cluster. Figure 4-15 illustrates a transaction group with an OUT followed by an IN compared with the same sequence of transactions using the USB 2.0 protocol.

The token blocks in the MMC (Figure 4-15) actually contain several important pieces of information, including Token information (device, endpoint, direction, etc.) and a description of the time slot for the Data or Handshake phase of the transaction. A host must order the time slots in a transaction group so that all of the host-to-device data phases (OUTs) are scheduled to run first in the transaction group (directly following the MMC) then the host will schedule all of the device-to-host time slots. The host must also construct the MMC so that W_XCTAs are in time-slot order.

The bit signaling rates provided by the device PHY implementation are nominally available across all function endpoints provided by a device implementation. The signaling rate capabilities of a device are reported in the Wireless USB Device Capabilities descriptor, see Section 7.4.1.1. The host selects the packet bit transfer rate for data phase data packets based on a number of criteria, including: the channel conditions and transfer type constraints defined in Sections 4.5, 4.6, 4.7, and 4.8. For host to device transactions, a host must transmit data phase packets based on the current configured characteristics of the addressed function endpoint. Configured characteristics in this context are associated with the currently active (configured) device/interface configuration characteristics. For device to host transactions, (i.e. data phase packet transmissions), the host directs the device on bit transfer rate, function endpoint payload size and burst size to use per data phase. The host must adhere to the constraints of the transfer type and advertised capabilities of the function endpoint.

Endpoint maximum packet sizes in this specification indicate 'application' data payloads only. They do not include any of the MAC or PHY Layer components or any of the security encapsulation or Wireless USB header overhead components. See Section 5 for details.

4.4.1 Burst Mode Data Phase

The USB 2.0 protocol allows a maximum of one data packet per USB transaction. Due to the significant packet delimiter overheads for wireless (long packet preambles, MIFS, SIFS, etc.), Wireless USB includes the capability to send multiple data packets during a transaction's data phase (see Figure 4-16). This feature allows for potentially better efficiency because packet delimiters and inter-packet gaps can be reduced. The general term for this capability is a Burst Mode Data Phase. All Wireless USB Data Phases use the Burst Mode Data Phase rules; even if burst size is one (see Section 5.4).

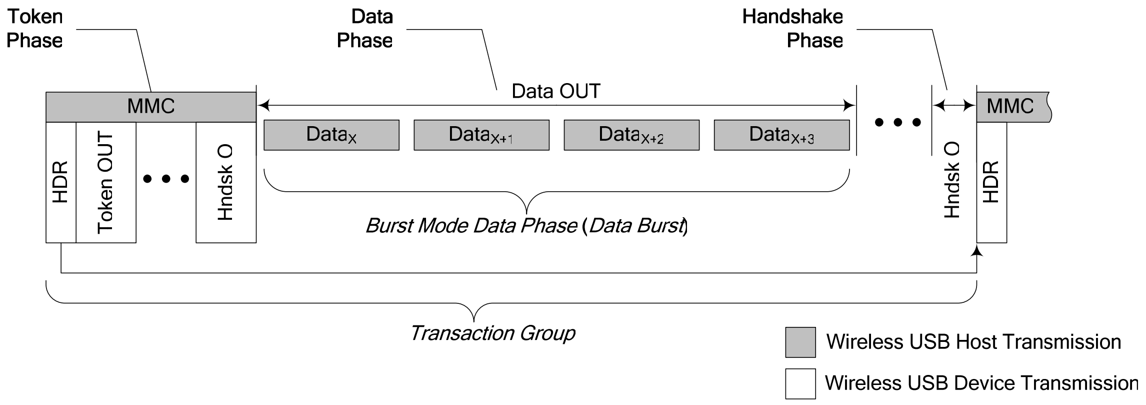

Figure 4-16. Example WUSB Data Phase Data Burst Footprint

"Data Burst" is a generic term for the series of data packets that are transmitted during the data phase of a Wireless USB transaction (see Figure 4-16). Maximum data burst size is an individual function endpoint capability which depends on the function endpoint's current configuration. A host determines a function endpoint's maximum data burst size from its Wireless USB Endpoint Companion descriptor (see Section 7.4.4). The size of each data packet in a data burst must be the configured function endpoint's maximum packet size or adjusted maximum packet size (see Section 4.10.2), (except for short-packet situations and isochronous streams).

The host may dynamically change the burst size on a per-transaction basis (up to the configured maximum burst size). The detailed description of Wireless USB Data Bursting is provided in Section 5.4.

The host may use any burst size up to the configured maximum burst size. Examples of when a host may use different burst sizes include (but are not limited to) a fairness policy on the host, retries for an isochronous stream, etc. When the function endpoint is an OUT, the host can trivially control the burst size (receiver must always be able to manage a transaction burst size). Note that the host must observe the configured maximum endpoint sequence range as defined below, regardless of the actual burst size it is using. When the function endpoint is an IN, the host can limit the burst size for the function endpoint on a per-transaction basis via a field in the Token block of the MMC (see Section 5.2.1). Note that a host may override the configured burst size by specifying a value less than the configured maximum burst size of the function endpoint.

4.5 Bulk Transfers

The purpose and characteristics of Bulk Transfers is similar to that defined in USB 2.0 (Section 5.8 of the USB 2.0 Specification). Chapter 5 of this specification describes the details of the packets, bus transactions and transaction sequences used to accomplish Bulk transfers. Bulk transfer type is intended to support devices that want to communicate relatively large amounts of data at highly variable times where the transfer can use any available Wireless USB channel bandwidth. A Wireless USB Bulk function endpoint provides the following:

- Access to the Wireless USB channel on a bandwidth available basis
- Guaranteed delivery of data, but no guarantee of bandwidth or latency

Bulk transfers occur only on a bandwidth-available basis. With large amount of channel time and good channel conditions, bulk transfers may happen relatively quickly; for conditions with little channel time available, bulk transfers may take a long time.

Wireless USB retains the following characteristics of bulk pipes:

- No data content structure is imposed on communication flow for bulk pipes
- A bulk pipe is a stream pipe, and therefore always has communication flow either into or out of the host for any pipe instance. If an application requires a bi-directional bulk communication flow, two bulk pipes must be used (one IN and one OUT).

4.5.1 Bulk Transfer Packet Size and Signaling Rate Constraints

An endpoint for bulk transfers specifies the maximum data packet payload size and burst size that the endpoint can accept from or transmit to the Wireless USB channel during one transaction. The allowable maximum data payload sizes for bulk endpoints are packet size values between 512 and 3584 that are integral multiples of 512 (i.e. 512, 1024, 1536, 2048, 2560, 3072 and 3584). The maximum allowable burst size bulk endpoints may specify is any value in the range 1 to 16.

A host may use any of the device's reported PHY signaling rates for data packets transmitted during the data phase of a Wireless USB transaction. For OUT (host to device) transactions to a bulk endpoint, the host may use any supported PHY signaling rate for data packets. For INs (device to host) the host may direct the device to use any one of the supported PHY signaling rates for data packets transmitted during the data phase.

A host is required to support any Wireless USB bulk endpoint. A host must support all combinations of bulk packet sizes and bulk burst sizes. No host is required to support larger than maximum packet sizes. The host ensures that no data payload of any data packet in a transaction burst will be sent to the endpoint that is larger than the reported maximum packet size, it will not send more data packets than the reported maximum burst size and it will not use sequence numbers larger than or equal to the reported maximum sequence value of the endpoint.

As noted in Section 4.10.2 a host may use smaller data payloads per packet than the reported maximum packet size as a measure to improve PER, when the function endpoint reports that it supports data packet size adjustments. For function endpoints that do not support the data packet size adjustment, the host must always use the reported *wMaxPacketSize* with transactions to the function endpoint. For function endpoints that do support data packet size adjustment, the host may only use allowed data packet sizes less than or equal to the reported maximum packet size for the endpoint. For example, a bulk endpoint reports a *wMaxPacketSize* of 1536 bytes; a host may only use packet sizes in the set {512, 1024, 1536}. An IN transaction token (W_{DT}CTA, see Section 5.2.1.2) always includes the packet size the function endpoint should use for the data phase data packets. A host must always specify to use a data packet size supported by the function endpoint; otherwise the behavior is not defined. The data packet size selected for each bulk transaction is called the 'active' packet size. On OUT transactions, the function bulk endpoint (that supports packet size adjustments) must be prepared for the host to use any valid 'active' packet size in each transaction.

A bulk function endpoint must always transmit data payloads with data fields less than or equal to the transaction's active packet size. If the bulk transfer has more information than will fit into the active packet size for the transaction, all data payloads in the data burst are required to be active packet size except for the last data payload in the burst, which may contain the remaining data. A bulk transfer may span multiple bus transactions. The host is allowed to adjust the active packet size (when the device supports it) on every contiguous burst. See Section 4.10.2 for the definition of a contiguous burst. A bulk transfer is complete when the endpoint does one of the following:

- Has transferred exactly the amount of data expected
- Transfers a data packet with a last packet flag set to one in its Wireless USB header (see Section 5.1).

4.5.2 Bulk Transfer Channel Access Constraints

As with USB 2.0 a bulk function endpoint has no way to indicate a desired bus access frequency for a bulk pipe. Bulk transactions occur on the Wireless USB channel only on a bandwidth available basis; i.e. if there is Wireless USB channel time that is not being used for other purposes, bulk transactions will be moved. Wireless USB provides a "good effort" delivery of bulk data between client software and device functions. Moving control transfers over the channel has priority over moving bulk transactions. When there are bulk transfers pending for multiple endpoints, the host will provide transaction opportunities to individual endpoints according to a fair access policy, which is host implementation dependent.

All bulk transfers pending in a system contend for the same available bus time. Because of this, the USB System Software at its discretion can vary the bus time made available for bulk transfers to a particular endpoint. An endpoint and its client software cannot assume a specific rate of service for bulk transfers. Bus time made available to a software client and its endpoint can be changed as other devices are inserted into and removed from the system or also as bulk transfers are requested for other function endpoints. Client software cannot assume ordering between bulk and control transfers; i.e., in some situations, bulk transfers can be delivered ahead of control transfers. The host may determine that the sum of pending bulk transfers could achieve better throughput by making the Wireless USB channel larger. A host may reserve more MAC Layer channel time for the Wireless USB channel (i.e. enlarge the Wireless USB channel) for a short period of time in order to provide better throughput service to the pending bulk transfers. The decision to enlarge the Wireless USB channel for this purpose is host implementation dependent.

The host can use any burst size between 1 and the reported maximum in transactions with a bulk endpoint to more effectively utilize the available Wireless USB channel time. For example, there may be more bulk transfers than channel time available, so a host can employ a policy of using smaller data bursts per transactions to provide fair service to all pending bulk data streams.

When a bulk endpoint delivers a flow control event (as defined in 5.5.4) the host will remove it from the actively scheduled endpoints. The device must transmit an Endpoint Ready device notification to the host to notify it that the associated bulk endpoint has bulk data or bulk buffer space available and is ready to resume data streaming.

4.5.3 Bulk Transfer Data Sequences

Bulk transactions use the standard burst sequence for reliable data delivery protocol defined in Section 5.4. Bulk endpoints are initialized to the initial transmit or receive window condition (as defined in Section 5.4) by an appropriate control transfer (*SetConfiguration*, *SetInterface*, *ClearEndpointFeature*). A host likewise assumes the initial transmit or receive window state for bulk pipes after it has successfully completed the appropriate control transfer as mentioned above.

Halt conditions for a Wireless USB bulk pipe have the identical side effects as defined for a USB 2.0 bulk endpoint. Recovery from halt conditions are also identical to the USB 2.0 specification, see Section 5.8.5 in the USB 2.0 specification. A bulk pipe halt condition includes a STALL handshake response to a transaction or exhaustion of the host's transaction retry policy due to transmission errors (see Section 4.10).

4.6 Interrupt Transfers

The purpose and characteristics of Interrupt Transfers are similar to those defined in USB 2.0 (Section 5.7 of the USB 2.0 Specification). The Wireless USB interrupt transfer types are intended to support devices that want a high reliability method to communicate a small amount of data with a bounded over-the-air service interval. Chapter 5 of this specification describes the details of the packets, bus transactions and transaction sequences used to accomplish Interrupt transfers. The Wireless USB Interrupt transfer type nominally provides the following. :

- Guaranteed maximum service period.
- Guaranteed retries during the service period if delivery failures occur.
    - Up to 5 retries for a Wireless USB interrupt endpoint.

Note: Retries are only guaranteed if the host detects that there was some attempt by the device to send data. If the device is completely non-responsive the host may postpone further attempts to the next service interval.

- Guaranteed retry of transfer attempts the next service period in the case of multiple transfer failures during a service interval.

Interrupt transfers are attempted each service interval for an interrupt endpoint. Bandwidth is reserved to guarantee a transfer attempt and a certain number of retries each service period. Once a transfer is successful, another transfer attempt is not made until the next service period. The requested service interval for the endpoint is described in its descriptor. Subsequent sections describe the possible service intervals for Wireless USB interrupt endpoints.

Wireless USB retains the following characteristics of interrupt pipes:

- No data content structure is imposed on communication flow for interrupt pipes
- An interrupt pipe is a stream pipe, and therefore is always unidirectional.

4.6.1 Low Power Interrupt IN

Wireless USB provides explicit support for function endpoints (devices) that move data infrequently, have a low latency requirement on the delivery of the data and a keen requirement to significantly save power. In order to accomplish the low-latency, the host must poll the endpoint for data at the required poll rate (based on the value of *bInterval*). To maximize power savings, the error tolerance rules/policies of the host are relaxed so that the function endpoint is not required to be listening for every IN token. In general, a low power interrupt function endpoint only needs to respond to IN tokens when it has data to move. All of the general operational characteristics and rules of an interrupt transfer type as described above, apply for a low power interrupt IN. The exceptions to the general rules and characteristics are described below.

A host will provide up to 3 attempts for a low power interrupt IN function endpoint. A retry may only be provided by the host if the host detects that there was some attempt by the device to send data. If the host determines the device was non-responsive, it may postpone further transaction attempts until the next service interval.

A low power interrupt IN function endpoint must NAK at least every *TrustTimeout* period or risks being either STALLED or disconnected by the host (depending on whether the function endpoint is the only ‘active’ endpoint on a device). A host must not use the Keepalive IE with a device where the low power interrupt IN function endpoint is the only active endpoint on the device.

4.6.2 Interrupt Transfer Packet Size and Signaling Rate Constraints

An endpoint for interrupt transfers specifies the maximum data packet payload size that the endpoint can accept from or transmit to the Wireless USB channel. The allowable maximum data payload size for interrupt endpoints is 1024 bytes. The allowable maximum data payload size for low power interrupt endpoints is 64 bytes. The maximum allowable burst size for interrupt endpoints of any type is one. The equivalent of wired USB high bandwidth interrupt endpoints are not supported by Wireless USB. Wireless USB interrupt endpoints are only intended for moving small amounts of data with a bounded service interval. The Wireless USB protocol does not require the interrupt data packets to be maximum size. If an amount of data less than the maximum packet size is being transferred it does not need to be padded.

A host may use any of the device’s reported PHY signaling rates for data packets transmitted during the data phase of a Wireless USB transaction. For OUTs (host to device) transactions to an interrupt endpoint, the host may use any supported PHY signaling rate for data packets. For interrupt INs (device to host) the host may direct the device to use any one of the supported PHY signaling rates for data packets transmitted during the data phase.

A host is required to support any Wireless USB interrupt endpoint. A host must support maximum packet sizes from 0 to 1024 bytes for a Wireless USB interrupt endpoint. A host must support maximum packet sizes from 0 to 64 bytes for a Wireless USB low power interrupt endpoint. No host is required to support larger maximum packet sizes.

As noted in Section 4.10.2 a host may use smaller data payloads per packet than the reported maximum packet size as a measure to improve PER, when the device reports that it supports data packet size adjustments. This is an optional feature for an interrupt endpoint. For function endpoints that do not support the data packet size adjustments, the host must always use the reported *wMaxPacketSize* with transactions to the function endpoint. For function endpoints that do support data packet size adjustment, the host may only use allowed data packet sizes less than or equal to the reported maximum packet size for the endpoint. For interrupt endpoints that support maximum packet size adjustment, any packet size smaller than the reported *wMaxPacketSize* can be used. For example, with a *wMaxPacketSize* of 512 bytes; a host may only use packet sizes between 1 and 512. An IN transaction token (W_{DT}CTA, see Section 5.2.1.2) always includes the packet size the function endpoint should use for the data phase data packets. A host must always use a data packet size supported by the function endpoint; otherwise the behavior is not defined. The data packet size selected for each interrupt transaction is called the 'active' packet size. On OUT transactions, the function interrupt endpoint (that supports packet size adjustments) must be prepared for the host to use any valid 'active' packet size in each transaction.

An interrupt function endpoint must always transmit data payloads with data fields less than or equal to the transaction's active packet size. If the interrupt transfer has more information than will fit into the active packet size for the transaction, all data payloads in the transfer are required to be active packet size except for the last data payload in the transfer, which may contain the remaining data. An interrupt transfer may span multiple bus transactions. The host is allowed to adjust the active packet size (when the device supports it) on every contiguous burst. See Section 4.10.2 for the definition of a contiguous burst.

An interrupt transfer is complete when:

- Exactly the amount of data expected has been transferred
- A data packet is transferred with a last packet flag set to one in its Wireless USB header.

4.6.3 Interrupt Transfer Channel Access Constraints

Periodic endpoints can be allocated at most 80% of the Wireless USB channel time. A Host is allowed to temporarily use the 20% channel time reserved for Bulk and Control to attempt to prevent periodic stream failures.

An endpoint for an interrupt pipe specifies its desired service interval bound. An interrupt function endpoint can specify an interval from 4.096 to 4194.304 milliseconds.[2] The interval is reported as an integer value (x) from 6 to 16 in the *bInterval* field of an interrupt endpoint descriptor. The service interval is 2^{x-1} units of 128 microseconds. Table 4-2 shows the requested service interval in milliseconds for each *bInterval* value. The shaded values in the table show the intervals that can be achieved in a Wireless USB system. The service interval encoding is slightly different than the encoding used in the USB 2.0 specification, due to the MAC layer time base using units of 256 microseconds.

Table 4-2 Interrupt Endpoint Service Intervals

bInterval **Value**	**Requested Service Interval (milliseconds)**
1	0.128
2	0.256
3	0.512
4	1.024
5	2.048
6	4.096*
7	8.192

[2] The Wireless USB channel does not allow an absolute service interval guarantee smaller than 8.192 milliseconds. Once every 65.536 milliseconds the service interval gap could be this long. The typical interval bound will still be 4.096 milliseconds if 4 is the requested interval.

Table 4-2 Interrupt Endpoint Service Intervals (cont.)

bInterval Value	Requested Service Interval (milliseconds)
8	16.384
9	32.768
10	65.536
11	131.072
12	262.144
13	524.288
14	1048.576
15	2097.152
16	4194.304

The Wireless USB host may access the endpoint at any point during the service interval. The service interval provided could be smaller than the interval requested. The interrupt endpoint should not assume a fixed spacing between transaction attempts. The interrupt endpoint can assume only that it will receive a transaction attempt (and guaranteed retries) within the service interval bound. Note that errors can prevent the successful exchange of data within the service interval bound.

An interrupt IN function endpoint is required to provide a NAK response if it receives a request and has no data to send. A low power interrupt IN function endpoint is not required to NAK in this case. If an interrupt OUT function endpoint receives a transaction and does not have room to store the data it is required to NAK.

4.6.4 Interrupt Transfer Data Sequences

Interrupt transactions use the standard burst sequence for reliable data delivery protocol defined in Section 5.4. Interrupt endpoints are initialized to the initial transmit or receive window condition as defined in Section 5.4 by an appropriate control transfer (*SetConfiguration*, *SetInterface*, *ClearEndpointFeature*). A host sets the initial transmit or receive window state for interrupt pipes after it has successfully completed the appropriate control transfer.

When an interrupt endpoint delivers a flow control event (as defined in 5.5.4) (e.g. NAK), the host must automatically resume transaction attempts with the function endpoint in the next service interval. Interrupt function endpoints do not use endpoint ready DNTS notifications.

Halt conditions for a Wireless USB interrupt pipe have the identical side effects as defined for a USB 2.0 interrupt endpoint. Recovery from halt conditions are also identical to the USB 2.0 specification, see Section 5.7.5 in the USB 2.0 specification. An interrupt pipe halt condition includes a STALL handshake response to a transaction or exhaustion of the host's transaction retry policy due to transmission errors (see Section 4.10).

4.7 Isochronous Transfers

The purpose of Wireless USB Isochronous Transfers are similar to those defined in USB 2.0 (Section 5.6 of the USB 2.0 Specification). However, the characteristics of Wireless USB Isochronous Transfers are significantly different from the characteristics of wired USB isochronous endpoints. This section provides a concise summary of purpose and characteristics of Wireless USB isochronous endpoints. A more detailed discussion of factors that led to the different characteristics, operation, and design for Wireless USB isochronous endpoints is contained in Section 4.11. The Wireless USB isochronous transfer type is intended to support streams that want to perform constant rate, error tolerant, periodic transfers with a bounded service interval. Chapter 5 of this specification describes the details of the packets, bus transactions and transaction sequences used to accomplish Isochronous transfers. The Wireless USB Isochronous transfer type provides the following:

- Guaranteed bandwidth for transaction attempts on the Wireless USB channel with bounded latency.

- Guaranteed average constant data rate through the pipe as long as data is provided to the pipe.
  - Note – this guarantee is subject to the reliability of the wireless media. The Wireless USB isochronous model allows endpoints to operate with reliability similar to wired USB isochronous endpoints under most conditions.
- Guaranteed retries during the service period if delivery failures occur.
  - At least 30% of actual average throughput needs of the stream for potential use for retries see Section 4.11.9.

Note: A minimum of one guaranteed retry per service interval must be provided.

- Additional reliability during short term error bursts by adding delay to the stream. The amount of delay that can be added is a function of the buffering provided by the isochronous endpoint.

Isochronous transactions are attempted each service interval for an isochronous endpoint. Bandwidth in the Wireless USB Channel is reserved to guarantee a transaction attempt and a certain number of retries each service period. All reserved bandwidth is used each service interval until the isochronous endpoint provides a flow control response indicating that it is unable to send or receive additional data. The requested service interval for the endpoint is described in its descriptor. The Wireless USB isochronous transfer type is designed to support sources and destinations that produce and consume data at the same average rate. It is not required that software using the Wireless USB isochronous transfer type actually be isochronous in nature.

A Wireless USB isochronous pipe is a stream pipe and is always unidirectional. An endpoint description identifies whether a given isochronous pipe's communication flow is into or out of the host. If a device requires bi-directional isochronous communication flows two isochronous pipes must be used, one in each direction.

4.7.1 Isochronous Transfer Packet Size and Signaling Rate Constraints

An endpoint for isochronous transfers specifies the maximum data packet payload size that the endpoint can receive from or transmit to the Wireless USB channel. The allowable maximum data payload size for an isochronous endpoint is 3584 bytes. The maximum allowable burst size for isochronous endpoints is 1 to 16. A Wireless USB isochronous endpoint also reports its desired service interval bound. Together, the maximum packet size, maximum burst size, and desired service interval specify the average bandwidth needed by the isochronous endpoint. An isochronous endpoint must not report a greater average bandwidth than it will actually consume. The host and system software automatically provide additional reservation time for retries for Wireless USB isochronous endpoints.

A Wireless USB isochronous endpoint is limited to requesting a bandwidth requirement of no more than 40 Mb/s. An isochronous endpoint with a service interval of 4.096 milliseconds with a bandwidth requirement of 40 Mb/s will move 21475 bytes per service interval. Wireless USB isochronous endpoints have some flexibility in choosing maximum burst and packet sizes. Table 4-3 shows the requirements for maximum packet size and maximum burst for isochronous endpoints.

Table 4-3 Maximum Packet Sizes for Isochronous Endpoints

Maximum Burst Size	Max Packet Size	Total Data Payload Per Service Interval (bytes)
1	1 to 3584	1-3584
2	257 to 3584	513-7168
3	342 to 3584	1025-10752
4	385 to 3584	1537 – 14336
5	410 to 3584	2049 – 17920
6	427 to 3580	2561 – 21475
7	439 to 3068	3073 – 21475

Table 4-3 Maximum Packet Sizes for Isochronous Endpoints (cont.)

Maximum Burst Size	Max Packet Size	Total Data Payload Per Service Interval (bytes)
8	449 to 2685	3585 – 21475
9	456 to 2387	4097 – 21475
10	461 to 2148	4609 – 21475
11	466 to 1953	5121 – 21475
12	470 to 1790	5633 – 21475
13	473 to 1652	6145 – 21475
14	476 to 1534	6657 – 21475
15	478 to 1432	7169 – 21475
16	481 to 1343	7681 – 21475

For Example, an endpoint that needs to move 4000 bytes every service interval can choose maximum burst sizes from 2 to 8. This device may choose to have multiple alternate settings. For low quality links, a burst size of 8 and max packet size of 500 may provide the best reliability. For high quality links, a burst size of 2 and max packet size of 2000 will provide the best efficiency.

Wireless USB isochronous endpoints are only intended for moving data at a constant rate with a bounded service interval. The Wireless USB protocol does not require that individual isochronous data packets to be maximum size. If an amount of data less than the maximum packet size is being transferred it does not need to be padded.

All device default interface settings for Wireless USB isochronous endpoints must not include any isochronous endpoints with non-zero data payload sizes (specified via *wMaxPacketSize* in the endpoint descriptor). Alternate settings may specify non-zero data payload sizes for isochronous endpoints. If the isochronous endpoints have a large bandwidth requirement, it is recommended that additional alternate configurations or interface settings be used to specify a range of data payload sizes. For example, these settings might correspond to different resolutions of video streams for a wireless USB camera. Providing alternate settings with different bandwidth requirements increases the chance the device can be used with other Wireless USB devices present. In addition, wireless isochronous devices are recommended to support dynamically switching between their alternate settings. There are a variety of events that may cause sudden changes in the size of the Wireless USB Channel. These events include a variety of sources of interference and requirements to follow coexistence policies in a crowded channel with other UWB devices. A device will be able to continue and begin operation under more circumstances if it is capable of dynamically switching to modes of operation (alternate settings) that require less bandwidth.

USB system software uses the maximum data payload size, maximum burst size, and bus access period reported by an isochronous endpoint to ensure that there is sufficient bus channel time to accommodate the endpoint with the current Wireless USB channel. If there is sufficient Wireless USB channel time for the maximum data payload (which may mean the host may grow the size of the Wireless USB channel by allocating more of the MAC Layer channel) the configuration is established, if not, the configuration is not established. Determining if there is sufficient Wireless USB channel time for the isochronous endpoint may depend on which data rate the endpoint will be able to operate at for data transmissions. Section 4.7.4 describes the admission decision process for wireless isochronous and interrupt function endpoints. Isochronous endpoints requiring a large amount of data may only be able to operate when they are close enough to the host to allow higher transmission data rates. System software must make admission decisions where the endpoint may only receive enough bandwidth to operate successfully if it can work at a high data rate. In addition, there are coexistence policies for hosts to fairly interact and interoperate with other UWB devices. Admitting an isochronous endpoint may take the overall MAC Layer channel usage above a per device limit set by a coexistence policy for channel reservations. Operating above the limit makes the Wireless USB host susceptible to being forced to relinquish bandwidth to other UWB devices. A Wireless USB isochronous endpoint reports whether it is able to dynamically switch to lower bandwidth alternate settings to help system software make admission decisions in these cases.

A host may use any of the device's reported PHY signaling rates for data packets transmitted during the data phase of a Wireless USB transaction. For OUT (host to device) transactions to an isochronous endpoint, the host may use any supported PHY signaling rate for data packets. For isochronous INs (device to host) the host may direct the device to use any one of the supported PHY signaling rates for data packets transmitted during the data phase.

A Wireless USB host is required to support any Wireless USB isochronous function endpoint that meets the requirements illustrated in Table 4-3.

During configuration, a host will read a device's configuration. The configuration contains each isochronous endpoint's maximum data payload size, max transaction burst size, and maximum sequence count value. The host ensures that no data payload of any data packet in a transaction will be sent to the endpoint that is larger than the reported maximum packet size, it will not send more data packets than the reported maximum burst size and it will not use sequence numbers larger than or equal to the reported maximum sequence value of the endpoint.

As noted in Section 4.10.2 a host may use smaller data payloads per packet than the reported maximum packet size as a measure to improve PER, when the device reports that it supports data packet size adjustments. This feature is not supported for isochronous endpoints in this version of specification. Isochronous endpoints must not report support for data packet size adjustments. For isochronous endpoints, which do not support the data packet size adjustments, the host must always use the reported *wOverTheAirMaxPacketSize* with transactions to the function endpoint.

4.7.2 Isochronous Transfer Channel Access Constraints

Periodic endpoints can be allocated at most 80% of the Wireless USB channel time. A host is allowed to temporarily increase this allocation to attempt to prevent stream failures.

An endpoint for an isochronous pipe specifies its desired service interval bound. A Wireless USB isochronous endpoint can specify an interval from 4.096 to 4194.304 milliseconds.[3] The interval is reported as an integer value (x) from 6 to 16 in the *bOverTheAirInterval* field of an isochronous endpoint descriptor. The service interval is 2^{x-1} units of 128 microseconds. The service interval encoding is slightly different than the encoding used in the USB 2.0 specification, due to the MAC layer time base using units of 256 microseconds. The Wireless USB host may access the endpoint at any point during the service interval. In other words, the service interval provided could be smaller than the interval requested. The isochronous endpoint should not assume a fixed spacing between transaction attempts. The isochronous device can assume only that it will receive a transaction attempt (and guaranteed retries) within the service interval bound. Note that errors can prevent the successful exchange of data within the service interval bound.

An isochronous IN endpoint is required to provide a NAK response if it receives a request and has no data to send. If an isochronous OUT device receives a transaction and does not have room to store the data it is required to NAK.

Note: An isochronous OUT endpoint is required to process received data quickly, such that it never NAKs unless the buffering associated with the endpoint (and its reported *wMaxStreamDelay*) is nearly full and can not store the current burst.

4.7.3 Isochronous Transfer Data Sequences

Isochronous transactions use the standard burst sequence for reliable data delivery protocol defined in Section 5.4. The Wireless USB isochronous protocol must use handshaking for error detection and perform retries because of the potentially unreliable wireless media. Isochronous endpoints are initialized to the initial

[3] The WUSB channel does not allow an absolute service interval guarantee smaller than 8.192 milliseconds. Once every 65.536 milliseconds the service interval gap could be this long. The typical interval bound will still be 4.096 milliseconds if 4 is the requested interval.

transmit or receive window condition as defined in Section 5.4 by an appropriate control transfer (*SetConfiguration*, *SetInterface*, *ClearEndpointFeature*). A host sets the initial transmit or receive window state for isochronous pipes with the appropriate control transfer.

When an isochronous endpoint delivers a flow control event (as defined in 5.5.4) (e.g. NAK), the host must automatically resume transaction attempts with the function endpoint in the next service interval. Isochronous function endpoints do not use endpoint ready DNTS notifications.

Data may be discarded by the transmitter under the Wireless USB isochronous protocol. Each isochronous data packet contains an isochronous header that includes a presentation time. Gaps in the presentation time allow the receiver to detect when data has been discarded and how many packets have been discarded. The host uses an isochronous discard information element in the MMC to report discards to isochronous OUT endpoints. For some types of isochronous OUT endpoints the isochronous discard IE provides sufficient discard information and each presentation time does not need to be monitored, see Section 4.11 for a more detailed discussion of isochrony and some implementation examples. The host must discard data that is no longer usable by an isochronous OUT function endpoint. An isochronous IN function endpoint will be required to discard data if the channel quality has degraded and there is no buffering available (on the function) to store additional data.

Isochronous endpoints report the buffer size that is associated with the function endpoint. An isochronous IN endpoint device discards the oldest data in its buffer if this buffer size is exceeded. The host is responsible for not sending old data to an isochronous OUT endpoint based on presentation time. Section 4.11 provides examples and more detailed discussions of how buffering is used to add delay and increased short term error reliability to a wireless USB isochronous stream.

Wired isochronous endpoints never halt because the wired isochronous protocol does not include handshaking to report a halt condition. Wireless USB isochronous endpoints also must not return a STALL handshake.

4.7.4 Isochronous Endpoint Host System Admission Decisions

Isochronous endpoints in the default interface setting are required to consume no bandwidth by reporting a maximum packet size of zero. The interface will have one or more alternate settings that provide additional settings for the isochronous endpoint. Each alternate setting contains endpoint descriptors that report a maximum transaction burst size, a maximum packet size and a service interval for the isochronous endpoints in that alternate interface setting. These values inform the host how much bandwidth each isochronous endpoint will need reserved in an error free environment to function when that alternate setting is selected.

When the host receives a request to enable an alternate setting with an isochronous endpoint – it will need to determine whether it can support the bandwidth required by the alternate setting. Specifying an exact policy for how hosts make this decision is outside the scope of the Wireless USB base specification. However, this section provides an informative discussion of some of the factors that may be considered by the host in these decisions.

A host may use LOOPBACK_DATA_WRITE and LOOPBACK_DATA_READ configuration requests (described in Sections 7.3.1.7 and 7.3.1.8) to estimate the error rate for the link at one or more data rates and packet sizes. In the case of an interface with an isochronous OUT endpoint, the host may repeatedly use the Data Loopback Write request to send packets with the same size as the isochronous OUT endpoint's maximum packet size at a data rate the host wants to evaluate. The success rate of these data loopback write requests (successful acknowledge received) will give the host information to use in deciding whether to enable the alternate setting with the isochronous OUT endpoint. The host may also use the information to decide to schedule additional time for retries if it allows the alternate setting to be configured. In the case of an interface with an isochronous IN endpoint, the host may use a Data Loopback Write request to send a packet with the isochronous IN endpoint's maximum packet size. The host may then repeatedly perform Data Loopback Read requests of the same size at a data rate the host wants to evaluate.

Alternatively, the host may consider link quality information from previous traffic to the device (other interfaces or the default control endpoint) in making admission decisions, if this information is available.

In crowded environments, the overall amount of bandwidth that a host may keep reserved under contention is limited by coexistence policies. If a host is required to expand its bandwidth reservation to more bandwidth than it is allowed to keep under contention, it may consider whether the endpoints in the alternate setting support dynamic switching. If dynamic switching is supported, the endpoint supports dynamically being reconfigured to a different bandwidth mode (represented by a different alternate setting).

4.7.5 Isochronous Data Discards and Use of Isochronous Packet Discard IE

When link conditions degrade for an extended period of time an isochronous transmitter may be required to discard data. The Wireless USB protocol contains mechanisms to provide the receiver with information about the number of discarded packets. This section describes the discard mechanism for isochronous OUT and isochronous IN function endpoints and the requirements for the host and function endpoint in each situation. Each isochronous data packet includes an isochronous specific header that includes a presentation time associated with the packet. The presentation time is used in making discard decisions and processing data, see the protocol chapter for specific details on the isochronous header format.

An isochronous IN function endpoint only discards data when it runs out of physical storage. When data must be discarded, the isochronous IN function endpoint discards the oldest data stored in the buffer. If the isochronous IN function endpoint has already tried to transmit the discarded data, it attempts to transfer the oldest available non-discarded packets using the same burst sequence number(s) as the discarded packet(s). The host must examine presentation times on received packets and place data into buffer locations based on the presentation time. This behavior ensures that the host will correctly process data that is transmitted out of order when an isochronous IN function endpoint discards data.

A host only discards data to an isochronous OUT function endpoint if the presentation time for any of the isochronous segments in the packet is earlier than the current Wireless USB Channel time. A host must discard the entire packet (all segments) if the presentation time is earlier than the current Wireless USB Channel time. When the host discards a packet to an isochronous OUT function endpoint it does not reuse the burst sequence number associated with the discarded packet. When the host discards a packet it must notify the isochronous OUT function endpoint of the discard. The host performs this notification by transmitting an Isochronous Packet Discard IE. The Isochronous Packet Discard IE indicates the number of packets discarded, the expected state of the isochronous OUT endpoint receive window after the discards, the first position in the receive window, and an ID number for the discard IE. The host must put the Isochronous OUT discard IE before W_XCTAs in the MMC. The host must include at least one W_{DT}CTA for the isochronous IN function endpoint in the same MMC. If the host receives a transmission from the endpoint in the W_{DT}CTA time slot it must not send the Isochronous OUT discard IE again until another discard occurs. If the host does not receive a transmission, it must continue to send the Isochronous OUT discard IE in subsequent MMCs. If the host must discard additional packets to the isochronous OUT function endpoint before it has received a transmission from the device it must update the discarded packet count and expected device receive window in the Isochronous Packet Discard IE and continue to send the IE until it receives a transmission from the endpoint confirming that it received an MMC transmitted with the Isochronous Packet Discard IE. The host must use the same ID value for all (re)transmissions of an Isochronous packet discard IE until it receives a transmission from the endpoint. When the host transmits a new isochronous packet discard IE to an endpoint it must increment the ID value. When a device containing an Isochronous OUT function endpoint receives an Isochronous Packet Discard IE for that endpoint it must update its receive window to the expected window specified in the IE.

Note: With some isochronous OUT function endpoint implementations it is not necessary to process Isochronous Packet Discard IEs. See implementations examples in Section 4.11.7 for more details.

Use of the dynamic switching mechanism for interfaces with multiple isochronous endpoints is beyond the scope of this specification. It is recommended that isochronous endpoints supporting dynamic switching be placed in separate interfaces if possible.

4.8 Control Transfers

The purpose and characteristics of Control Transfers are identical to those defined in USB 2.0 (Section 5.5 of the USB 2.0 Specification). Chapter 5 of this specification describes the details of the packets, bus transactions and transaction sequences used to accomplish Control transfers. Chapter 9 of the USB 2.0 specification and Chapter 7 of this specification define the complete set of standard command codes used for devices.

Each device is required to implement the Default Control Pipe as a message pipe. This pipe is intended to be used for device initialization and logical device management. This pipe is to be used to access device descriptors and to make requests of the device to manipulate its behavior (at a device-level). Control transfers must adhere to the same USB data structure definitions described in USB 2.0.

The Wireless USB system will make a "best effort" to support delivery of control transfers between the host and devices. As with USB 2.0, a function and its client software cannot request specific channel access frequency or bandwidth for control transfers. System software may restrict the channel access and bandwidth a device may desire as defined in Section 4.8.1and 4.8.2.

4.8.1 Control Transfer Packet Size and Signaling Rate Constraints

Control endpoints have a fixed maximum control transfer data payload size of 512 bytes and have a maximum burst size of one (1). These maximums apply to all data transactions during the data stage of the control transfer. Refer to Section 5.5.2 for information on optimizations beyond the USB 2.0 standard provided in Wireless USB for the Setup and Status stages of a control transfer.

A Wireless USB device must report a value of 255 (FFH) in the *bMaxPacketSize* field of its Device Descriptor. The host must ignore this field in the Device Descriptor for Wireless USB devices and assume a *wMaxPacketSize* of 512 (200H) for the Default Control Pipe. A host must always use the PHY base signaling rate for Wireless USB Standard request transfers on the Default Control Pipe. This is in order to make these requests as reliable as possible. A host may choose to use other signaling rates for all other control transfers, based on the same criteria it uses to select signaling rates on other endpoint transfer types (see Section 4.4). The Default Control Pipe has a maximum sequence value (*bMaxSequence*) of 2 (i.e. only sequence values in the range [0-1] are used). Note, the special control requests DATA_LOOPBACK_READ and DATA_LOOPBACK_WRITE are not required to use the PHY base signaling rate for packets transmitted during the request's data phase. These requests may require packet sizes larger than 512 bytes to be used. These exceptions are described in detail in Sections 7.3.1.8 and 7.3.1.7.

The requirements for data delivery and completion of device to host and host to device Data stages are generally not changed between USB 2.0 and Wireless USB (see Section 5.5.3 in the USB 2.0 Specification). A control pipe may have a variable-length data phase in which the host requests more data than is contained in the actual data structure. When all of the data is returned to the host, the function must indicate that the Data stage is ended by returning a packet with the last packet flag set to one. This rule applies regardless of the size of the last data packet.

4.8.2 Control Transfer Channel Access Constraints

A device has no way to indicate a desired bus access frequency for a control pipe. A host balances the bus access requirements of all control pipes and pending transactions on those pipes to provide a "best effort" delivery between client software and functions on the device. This policy is unchanged with regards to USB 2.0.

Wireless USB requires that part of the Wireless USB Channel be reserved to be available for use by control transfers as follows:

- A control transfer comprises multiple transactions and may therefore span more than one transaction group. The individual transactions of a control transfer may, or may not be scheduled onto contiguous transaction groups.
- The transactions of a control transfer may be scheduled co-incident with transactions for other function endpoints of any defined transfer type.
- Retries are not required to occur in contiguous transaction groups and may be scheduled with equal priority to all new control and bulk transactions.
- Control transfers that are being frequently retried should not be expected to consume an unfair share of channel time.
- If there are too many pending control transfers for the available channel time, control transfers are selected to move through the channel as appropriate.

- If there are control and bulk transfers pending for multiple endpoints, control transfers for different endpoints are selected for service according to a fair access policy that is Host Controller implementation-dependent.
- When a control endpoint delivers a flow control event (as defined in 5.5.4), the host will remove the endpoint from the actively scheduled endpoints. The device must transmit an Endpoint Ready device notification to notify the host that it is ready to resume data streaming on the flow controlled endpoint.

These requirements allow control transfers between a host and devices to regularly move through the Wireless USB Channel with “best effort”. The basic System Software discretionary behavior defined in USB 2.0 (Section 5.5.4) applies equally to Wireless USB Control Transfers.

4.8.3 Control Transfer Data Sequences

Wireless USB preserves the message format and general stage sequencing of control transfers defined in USB 2.0 (see Section 5.5.5). The Wireless USB protocol defines several optimizations for the Setup and Status stages of a control transfer, however all of the sequencing requirements for normal and error recovery scenarios defined in USB 2.0 Section 5.5.5 directly map to the Wireless USB Protocol. The only necessary clarification is in regards to recovery from a halt condition, because Wireless USB does not utilize a Setup PID.

After a halt condition is encountered or an error is detected by the host, a control endpoint is allowed to recover by accepting the next Setup stage control as defined in Section 5.5.2; i.e. recovery actions via some other pipe are not required for control endpoints. For the Default Control Pipe, a device reset may be ultimately required to clear a halt or error condition if the next Setup control is not accepted.

4.8.4 Data Loopback Commands

Wireless USB requires that all devices support a pair of control loopback requests: Data Loopback Write and Data Loopback Read. The Data Loopback requests provide a standard method for accomplishing loopback capability on all devices. This capability is utilized for a variety of purposes, including compliance testing and link quality estimation. These requests require exceptions to several of the standard rules for control requests. Detailed requirements for the loopback requests are described in this section.

All devices are required to support the loopback requests in the **UnAuthenticated** device state and in the **Default** and **Address** device states. Any device that contains one or more isochronous endpoints in any of its configurations (isochronous device) must also support the loopback commands in the **Configured** state. Non isochronous devices are not required to support the request in the configured state. Behavior for a non isochronous device is not specified if it receives one of the loopback requests in the configured state.

The amount of data that any device must be able to store for the loopback commands is dependent on the configuration(s) of the device. A device must be able to store a data payload equal in size to the largest *wMaxPacketSize* or *wOverTheAirPacketSize* value from any of the device’s endpoints in any of its configurations (devMaxPacketSize). The device must be able to store a data packet of devMaxPacketSize or less, received with a Data Loopback Write request.

Note: The data packet sent in the data stage of a Data Loopback Write can be up to devMaxPacketSize in bytes. This is an exception to the normal 512 byte limitation imposed by the control endpoint maximum packet size. The host can not send more than one data packet in the data stage of a Data Loopback Write. Device behavior if the host includes multiple packets in the data stage is undefined. Loopback requests may use any data rate supported by the device. The device must support a Data Loopback Write request with any data rate that the device indicates it supports. The device must perform the data stage of a control loopback read request at the specified data rate.

After a device receives a Data Loopback Write request it is required to store the data payload. The required behavior for a device with stored loopback data including how long the data must be stored is described for each possible case in the list below:

- The device receives another Data Loopback Write request. The device is required to overwrite any currently stored data with the data from the current Data Loopback Write Request.

- Any control request other than a Data Loopback Read or Data Loopback Write occurs. The device is not required to continue to store loopback data.
- A Data Loopback Read Request occurs. If the device has stored loopback data it is required to return that data (up to the requested length) in a single packet in the data stage of the Data Loopback Read request. If the requested length is longer than the length of the stored data, device behavior is undefined. The device must use the specified power level and data rate in the data stage of the request. After a Data Loopback Read request occurs, the device must continue to store the loopback data until one of the events described in the preceding bulleted items occurs.

4.9 Device Notifications

Device Notifications are a standard method for a device to communicate asynchronous, device and Bus-level event information to the host. This communication method is a bus-level feature that does not map to the pipe model defined for the standard transfer types. Device Notifications are always initiated by a device and the flow of data information is always device to host. Each packet transmitted is called a Device Notification or simply Notification.

Notifications are message-oriented data communications that have a specific data format structure as defined in Section 7.6 and a specific media access mechanism as defined in Section 5.2.1.3 which describes Device Notification Time Slots (DNTS). These data communications do not use Wireless USB data transactions as defined in Section 5.3 to accomplish data transfers. The maximum allowable data payload for a Device Notification message is 32 bytes and the messages must always be transmitted at the PHY base signaling rate.

Device Notification time slots are scheduled by the host on an “as-needed” basis. The amount of channel time scheduled by the host depends on the service intervals of the pending events (flow control, keep alives, etc.) and implementation specific policies of the host. A host may schedule a maximum of one device notification time slot per MMC.

4.10 Media Reliability Considerations

Wireless is an unreliable media, as compared to most non-wireless technologies (i.e. copper, optical, etc). There are many different forms of interference that contribute to unreliability and a commensurate quantity of interference mitigation techniques. In Wireless USB, interference mitigation is managed by the host of the Wireless USB Channel which has the bits of information and set of characteristics to control listed below.

Information

Host-centric information	The host can maintain statistical information on every device (PER, etc.), it also has physical information about what its view of the MAC Layer channel is like (LQI, etc.).
Device-centric information	The host can query the device for its view of the MAC Layer channel. The information returned includes: LQI, etc.

Controls

Transmit Power Control	(TPC). Wireless USB offers manipulation of transmit power levels via transaction level control attributes and device-level management commands. See Section 4.10.1 for details.
Transmit Bit Rate Control	The transmit bit rate can be adjusted on a per-transaction basis.

Controls (cont.)

Data payload Size Control	The nominal size of the data payload of transmitted packets is established by the *wMaxPacketSize* of the function endpoint. A commonly defined wireless tool for mitigating some forms of interference is the ability to change the size of data packets, so that they are statistically less likely to be corrupted during transmission. See Section 4.10.2 for details. This is an optional feature for devices and hosts.
Transfer Burst Size	A device exports a maximum level of bursting capability, and the host (depending on transfer type) can choose to utilize the bursting capability up to the level required for fair and efficient service for the devices connected to the Wireless USB Channel.
PHY Channel Change	A host can choose to move the Wireless USB channel and associated cluster to an alternative PHY channel. See Section 4.10.4 for details.
Host Schedule Control	The host may temporarily use time allotted for asynchronous (bulk/control) transactions to provide additional retries for streams that are experiencing significant error rates.
Dynamic Bandwidth Interface Control	Some interfaces containing isochronous endpoints may support being dynamically switched to other bandwidth modes. The host may be able to switch an interface to address a change in available bandwidth.

The following sections provide overview descriptions of the controls that are available to a host controller, the application of these controls is a host implementation issue and beyond the scope of this specification.

USB 2.0 specifies that a transaction for a non-isochronous endpoint will be attempted at most 3 times before the associated pipe is put into a Stalled state (isochronous gets one try). Wireless USB provides a host a much larger number of attempts in order to allow a reasonable number of transaction opportunities over which it can employ the controls described below to optimize the Wireless USB channel for data communications between the host and a Function Endpoint. Wireless USB requires a host to try a transaction at least seven (7) times before stalling a non-isochronous pipe. Retries are applied at the transaction level, not the data packet level. For example, if any data packet of a burst succeeds (i.e. the data stream advances), then the host's pipe retry counter should be reset to zero. Isochronous pipes don't stall and have specifically defined mechanisms for advancing the data stream when data cannot make it from source to sink and data expires see Section 4.11. Note that low-power Interrupt IN endpoints must be managed with a different retry policy. These types of function endpoints are explicitly allowed to not respond if they do not have data. Therefore, there is no practical maximum retry count that is applicable. However, it is still necessary to determine whether the function or function endpoint has failed.

4.10.1 Transmit Power Control

Wireless USB devices report the TPC levels and adjustments they support (see Section 7.4.1.1). A host may set the TPC level(s) of a device. The power levels for beacons/directed transmissions and notifications for the device are managed using the device-level set WUSB data request (see Section 7.3.1 for details). The host can query the current device level transmit power control settings by issuing a GetStatus() request on the Default Control Pipe for the device-level features. A host specifies the power level to be used by the device for data and handshake packets transmitted by the device. Note, the device resets these parameters to their default values (i.e. highest power setting) whenever the device returns to the **UnConnected** device state.

The transmit power control settings for the device are only valid for data communications over the associated Wireless USB channel. For example if a device implementation supports multiple protocols (For example, Wireless IP and Wireless USB), the transmit power settings only affect data communications for the Wireless

USB Channel. If a device is capable of simultaneous operation on more than one Wireless USB channel, the transmit power adjustments only apply to the channel in which the adjustment is made.

The host can make power transmit level adjustments to device beacons/directed transmission and to Device Notifications or for individual protocol time slots at any time. The device is required to activate the change in transmit power for the next packet transmission from the device (after the new power setting has been received from the host). All devices are required to support transmit power control. The power control settings allow 8 power levels to be specified. A device reports the size of the steps between power levels, and number of power levels that are supported for operation on TFI and FFI channels. This version of the specification requires devices to support a power level step size of 2 dB and specifies accuracy requirements on supported power levels that allow both baseband and RF implementations of transmit power level control. There are two parts to the accuracy requirements:

- Absolute allowed ranges for each power level.
- Additional constraints that guarantee monotonically decreasing power levels for each increasing power control setting.

Table 4-4 shows the nominal power levels with a 2 dB step for each Wireless USB Power Control Setting and the absolute accuracy requirements for each level. The shaded power levels in the table must be supported by all devices.

Table 4-4 Nominal Transmit Power Level Values and Accuracy Requirements for Each Level

Power Control Setting	TFI Channel Power Level		FFI Channel Power Level	
	Nominal values	Accuracy requirement	Nominal values	Accuracy requirement
0	TFI_BASE	TFI_BASE	FFI_BASE	FFI_BASE
1	TFI_BASE – 2 dB	TFI_BASE – (1 to 3) dB	FFI_BASE – 2 dB	FFI_BASE – (1 to 3) dB
2	TFI_BASE – 4 dB	TFI_BASE – (3 to 5) dB	FFI_BASE – 4 dB	FFI_BASE – (3 to 5) dB
3	TFI_BASE – 6 dB	TFI_BASE – (4.8 to 7.2) dB	FFI_BASE – 6 dB	FFI_BASE – (4.8 to 7.2) dB
4	TFI_BASE – 8 dB	TFI_BASE – (6.4 to 9.6) dB	FFI_BASE – 8 dB	FFI_BASE – (6.4 to 9.6) dB
5	TFI_BASE – 10 dB	TFI_BASE – (8 to 12) dB	FFI_BASE – 10 dB	FFI_BASE – (8 to 12) dB
6	TFI_BASE – 12 dB	TFI_BASE – (9.6 to 14.4) dB	FFI_BASE – 12 dB	FFI_BASE – (9.6 to 14.4) dB
7	TFI_BASE _ 14 dB	TFI_BASE – (11.2 to 16.8) dB	FFI_BASE – 14 dB	FFI_BASE – (11.2 to 16.8) dB

The absolute accuracy requirements allow the power levels to overlap.

As mentioned above, there are requirements to ensure that the absolute power levels decrease when power control settings are increased. These requirements constrain the amount that the absolute power level can change for a change of N Power Control Settings. Table 4-5 shows the required range of the power change for each possible size of change for the Power Control Setting.

Table 4-5 Required Range Of Power Level Change For Changes In Power Control Setting

Change In Power Control Setting Value	Required Change In Absolute Power Level
1	1 to 3 dB
2	3 to 5 dB
3	4.8 to 7.2 dB
4	6.4 to 9.6 dB
5	8 to 12 dB
6	9.6 to 14.4 dB
7	11.2 to 16.8 dB

For example, any change of 2 power control setting value must correspond to a power level change of at least 3 dB and no more than 5 dB. Therefore, if the absolute power level for setting 4 was TFI_BASE – 8 dB then the absolute power levels for settings 2 and 6 would have to be within the ranges TFI_BASE – (3 to 5) dB and TFI_BASE – (11 to 13) dB respectively.

If instead the absolute power level for setting 4 was TFI_BASE – 6.4 dB then the absolute power levels for settings 2 and 6 would have to be within the ranges TFI_BASE – (3 to 3.4) dB and TFI_BASE – (9.6 to 11.4) dB respectively. These values may be derived as follows: A change from setting 0 to setting 2 must be at least 3 dB. Therefore setting 2 can not be less than 3 dB below the TFI_BASE (regardless of the value for setting 4). A change of 6 settings must be at least 9.6 dB. Therefore setting 6 can not be less than 9.6 DB below TFI_BASE (regardless of the value for setting 4).

4.10.2 Adjustments to Data Phase Packet Payload Sizes

Large data packets are more efficient for moving data except when interference causes the packet error rate to increase and retries dominate the transaction traffic on the channel. Packets with larger payloads have a statistical higher probability of encountering an uncorrectable error. Therefore, under certain circumstances, throughput efficiency can be increased by decreasing the size of transmitted data packets.

A host can adjust the payload sizes of data phase data packets as one method of managing the packet error rate on data streams. The effective data stream is illustrated in Figure 4-17.

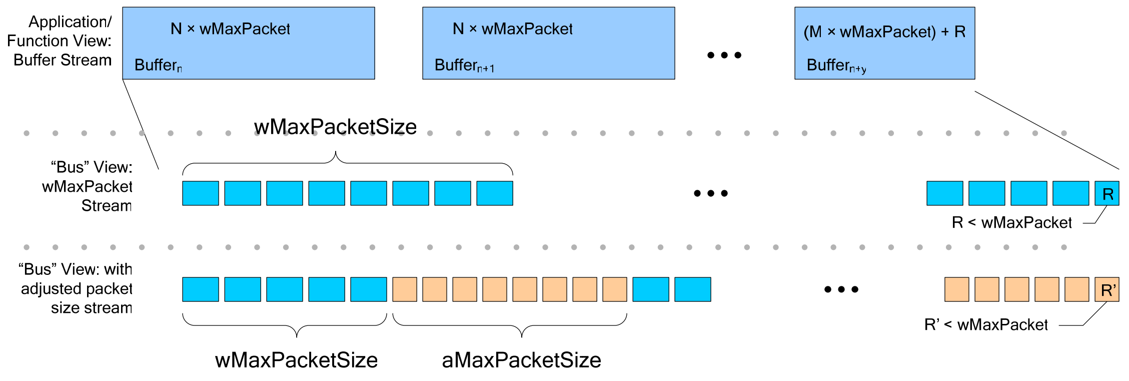

Figure 4-17. Transfer to Transmitted Packet Mapping for USB 2.0 and Wireless USB

USB 2.0 data communications require (for Bulk, Control and Interrupt transfer types) endpoints to always transmit data payloads with a data field less than or equal to the function endpoint's declared *wMaxPacketSize*). When a transfer request has more data than can fit in one maximum-sized payload, all data payloads are required to be maximum size except for the last payload, which will contain the remaining data. A transmitter may complete a transfer by moving exactly the data expected, or more generically, the transmitter may delineate the data stream by transmitting a short packet (i.e. less than *wMaxPacketSize* bytes in a data payload). Note that zero bytes in the data packet payload is still considered a short packet. Short packet semantics must be preserved for Wireless USB Bulk, Control and Interrupt function endpoints. Figure 4-17 illustrates this behavior in the top and middle rows, where the top row is a buffer stream (which may be one or more buffers) that represents a transfer of some application-specific unit of data. The middle row illustrates, the 'bus' view of the data payloads, each of *wMaxPacketSize*. 'R' represents the residual or remaining data which is less than *wMaxPacketSize*. The bottom row of Figure 4-17 illustrates the 'bus' view of data payloads where, under control of the host, the payload sizes of data packets in the data stream may be *wMaxPacketSize* intermixed with adjusted payload sizes (*aMaxPacketSize*). The limits about how and when a host is allowed to adjust payload sizes in a data stream are described below. The mix is controlled and managed by the host's implementation policy.

To preserve short packet semantics and support adjustments to data payload sizes less than the reported *wMaxPacketSize*, the short packet occurrences in the data stream must always be explicitly marked by the transmitter (regardless of whether the host or device supports data packet size adjustments). The *bmStatus.Flags.Last Packet Flag* field in the Wireless USB data header (see Section 5.1) is used to mark the last packet in a transfer. This field must be set by the transmitter on any data packet that would satisfy the 'short' packet semantics of a data stream as defined in the USB 2.0 specification.

The allowable adjustments depend on several runtime characteristics of the pipe for which the adjustments are being made. The rules for making these adjustments are:

- Changes to the data payload size can occur only on transaction boundaries.
- Changes to the data payload size can occur on contiguous-burst boundaries. This means that a host cannot adjust the burst size if there is out-of-order data packets in the data stream (i.e. burst sequence) that need to be retried.

An OUT function endpoint must always use the configured *wMaxPacketSize* as the basis for reporting buffer availability in the acknowledgement bit vector (*bvAckCode*, see Section 5.1) portion of the handshake packet. When adjusting the data packet payloads, the host must not violate the function endpoints' declared burst and buffering capabilities.

4.10.3 Adjustments to Transmit Bit Rate

The host can adjust the transmit bit rate for data packets transmitted during the data phase of a transaction on a per transaction basis. For OUT (host to device) data phases, the host may change the transmit bit rate as often as every data packet in the data burst. However, it must not do any adjustments in transfer bit rate that would violate the Wireless USB channel protocol time slot. For IN (device to host) data phases, the host will allocate a Wireless USB Channel protocol time slot for the data phase of the transaction based on an expected transmit bit rate, which is communicated to the device in the *PHY_TXRate* field of the W_{DT}CTA. The host must use a *PHY_TXRate* value that is supported by the device.

4.10.4 Changing PHY Channel

A host establishes and maintains a Wireless USB channel instance within a single PHY channel but can decide to move the Wireless USB Channel and thus the devices in the Wireless USB Cluster to another PHY channel. The criteria and process used by a host to decide to change to an alternate PHY channel is beyond the scope of this specification. This section describes the mechanisms provided for a host to communicate a PHY channel change to the members of a Wireless USB Cluster. Note that Wireless USB does not provide a generic mechanism to move a subset (one or more) of cluster members to a different channel. Rather, it describes a method for moving an entire cluster.

Wireless USB channel time is not synchronized to the underlying MAC Layer timing structure, so the host's Wireless USB Channel time is continuous across any PHY channels where the host decides to locate the Wireless USB Channel. Since the time base is continuous, the host simply needs to notify the cluster members that MMCs will be available on a different PHY channel soon. The host accomplishes this by including a Channel Change announcement (see Section 7.5.3) in the MMCs which announces that a PHY channel change will occur at a specific Wireless USB channel time. The host must continue to obey the MMC transmission requirements, regardless of the PHY channel where it has located the Wireless USB channel. Figure 4-18 illustrates an example PHY channel change sequence.

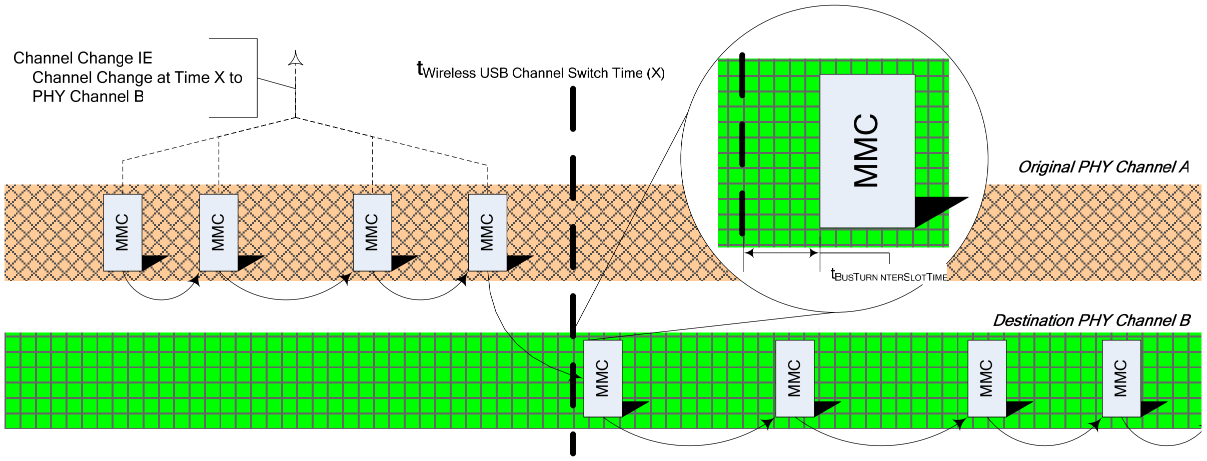
Channel Change IE
Channel Change at Time X to
PHY Channel B
$t_{\text{Wireless USB Channel Switch Time (X)}}$
MMC
Original PHY Channel A
$t_{\text{BusTurnInterSlotTime}}$
Destination PHY Channel B

Figure 4-18. Example Wireless USB Channel Change

The host must announce a PHY channel change in at least three (3) consecutive MMCs before the channel change event time. The host must specify a channel change event time that occurs after the transmission of the last MMC in the original PHY channel and at least $\mathbf{t_{BusTurnInterSlotTime}}$ (see Section 5.3) before the next MMC transmission in the destination PHY channel. If the host schedules protocol time slots between the two MMC transmissions of different PHY channels, the channel switch time must be after the protocol time slots.

A Wireless USB device must simply move to the specified destination PHY channel at the specified Wireless USB channel time if it intends to remain a member of the host's Wireless USB Cluster. If a device does not change channel with the host, then it will be detected as disconnected using the standard disconnect mechanism. If a device transitions out of the Sleep state and observes a Channel Change announcement, it should remain in the Active state until after the channel switch time has elapsed and it has completed a keep-alive notification on the new PHY channel. Note, Directed Beaconing Devices that have the Transmit Packet feature enabled, must disable this feature when transitioning to the new PHY channel. The intent of this requirement is that the device does not transmit the directed packet on the new PHY channel. The host may, at a later time re-enable the transmit packet feature on the device.

A host may have devices in the Sleep device state when it determines that is moving the Wireless USB cluster to a different PHY channel. When this is the case, the host may:

- Wait for all *Sleeping* devices to return to the *Active* device state before initiating a channel change, or
- Initiate and complete the channel change without waiting for devices in the *Sleep* state. Devices that miss the channel change due to being in the *Sleep* state simple need to locate their host and reconnect if they want to remain available for future data communications with the host.

The host may be able to accomplish the channel change without perturbing active data streams; however, this is beyond the scope of this specification. In general a host will use the *Directed Packet Transmission and Reception* feature with Directed Beaconing cluster members and GetStatus() and *SetWUSBData*() with Self-beaconing cluster members to establish a neighbor-friendly reservation on the new PHY channel.

The host should change to a channel number that is supported by all current Wireless USB Cluster member devices.

4.10.5 Host Schedule Control

USB 2.0 (High-speed) requires that at least 20% of the available bus time be reserved for asynchronous data streams. Wireless USB preserves this allocation rule, but it allows a host to temporarily use/intrude on the asynchronous channel time in order to temporarily resolve reliability problems for periodic streams.

4.10.6 Dynamic Bandwidth Interface Control

Wireless USB defines an optional dynamic switching mechanism for interfaces containing isochronous endpoints. If an interface supports dynamic switching, the endpoints in the interface must support dynamically being reconfigured to a different bandwidth mode (represented by a different alternate setting). The process for performing a dynamic switch is described in detail in this section.. For example, the host may be able to switch an interface that supports dynamic switching to a lower bandwidth setting in a case where the PHY channel bandwidth has degraded significantly.

The process for performing a dynamic switch involves two steps. In the first part of the process the host sends a Set Interface DS (Dynamic Switch) control request to the device with an active interface that supports dynamic switching. The Set Interface DS request specifies a Wireless USB Channel time (switch time) for the dynamic switch to take place and an alternate setting that will be used after the switch. It is not possible to specify more than one switch time for different endpoints in an interface.

At the switch time an isochronous IN endpoint must start generating data in the format corresponding to the interface specified in the Set Interface DS request. When presentation times are applied, all data with presentation times after the switch time must use the format corresponding to the new alternate interface setting. The isochronous IN endpoint does not discard data that is currently buffered when it receives a Set Interface DS request. It continues to respond to IN requests following the characteristics of the current alternate interface. The device will be explicitly notified by the host when the host expects data transmitted over the air by the isochronous IN endpoint to correspond to the new alternate interface settings (maximum packet size, etc). This notification occurs during the second step in the dynamic switch process, which is described later in this section.

An isochronous OUT endpoint must handle data according to the appropriate format after the device has received a Dynamic Switch DS request. When the endpoint function processes data with presentation times after the specified switch time it must assume the data characteristics correspond to the alternate setting in the Dynamic Switch DS command. An isochronous OUT endpoint does not discard any data currently buffered when a Dynamic Switch DS request occurs. The isochronous OUT endpoint continues to process data according to the current interface settings as long as the presentation times associated with the data are before the switch time.

Note: There are a variety of implementations that a function endpoint could use to process data with presentation times before and after the switch time according to different formats. Some implementations may not rely on direct observation of presentation times.

In the second step of the dynamic switch process the host sends a Set Interface command to the device that previously received the Set Interface DS command. The Set Interface request must specify the same interface alternate setting that was specified in the Set Interface DS command (unless the host is selecting a different alternate setting with a traditional SetInterface() request). After the Set Interface request, over-the-air communication to and from all endpoints (of all types) in the specified interface must conform to the characteristics of the new alternate setting. After the Set Interface request is successfully completed the host will send all data to isochronous OUT endpoints in the switched interface using the over-the-air characteristics of the endpoint in the new alternate setting. After the Set Interface command the host expects an isochrones IN endpoint in the switched interface to respond with the over-the-air characteristics of the endpoint in the new alternate setting

There are several situations that can occur in the Dynamic Switch process. This section describes possible situations and host and device responsibilities in these cases.

- If a device that supports dynamic switching receives a Set Interface request without receiving a Set Interface DS request or receives a Set Interface request specifying a different alternate setting than the last Set Interface DS request it must treat the Set Interface request as it normally would in a situation that did not involve dynamic switching.

- If a device receives a Set Interface DS request and does not receive a Set Interface request for an extended period of time it must follow the rules outlined previously in this section. If the buffer for an isochronous OUT endpoint empties or the buffer for an isochronous IN endpoint starts to overflow the device may undertake vendor specific error reporting/handling steps at any time. The host must attempt to prevent this situation from occurring. However, a dynamic switch will typically be an attempt to avoid problems due to link degradation and these errors may occur.
- A host must not attempt to make a dynamic switch to an interface with larger bandwidth requirements if it does not already have allocated bandwidth to support the change.
- A host must not attempt to send a Set Interface DS request with a switch time earlier than the current Wireless USB Channel time.
- A host must not send data in the pre-switch format to an isochronous OUT endpoint with presentation times after the switch time.
- A host must not send data in the post-switch format to an isochronous OUT endpoint with presentation times before the switch time.
- A host must not make any requests for data from an isochronous IN endpoint that only has data in the post-switch format buffered before performing the Set Interface step in the switch process.

4.11 Special Considerations for Isochronous Transfers

This section begins with an overview of the key features of wired USB isochrony. It then describes the challenges with wireless media that prevent this model from being used with Wireless USB. The section concludes with a high level overview of the Wireless USB isochronous model.

4.11.1 Summary Of Key Features Of USB Wired Isochrony

This section presents a summary of the key features of the wired USB isochronous transfer model. The Wireless USB approach to isochronous transfers has significant differences from the wired model.

Consider a full speed wired USB device with an isochronous endpoint. When an isochronous stream is started the device has a contract guaranteeing the opportunity attempt to send or receive the requested amount of data each service interval. The amount of data to be moved is part of the endpoint descriptor for the isochronous endpoint. The delivery itself is not guaranteed (wired isochronous traffic does not use handshakes or retries), however wired USB bit error rates are required to be 10^{-9} or better. Therefore, the loss of data (when the requested amount of data is not sent or received in a service interval) is rare. The USB specification does not make any guarantees on the location of the service it provides to a wired USB isochronous endpoint in the service interval. The USB host sends Start of Frame (SOF) packets at the beginning of each frame. The SOF contains a frame index that rolls over every 2048 frames. The host is responsible for sending the SOF packets at regular 1 millisecond intervals. Figure 4-19 shows the worst case variation of service location for a FS wired USB device with a service interval of 1 millisecond (1 USB frame).

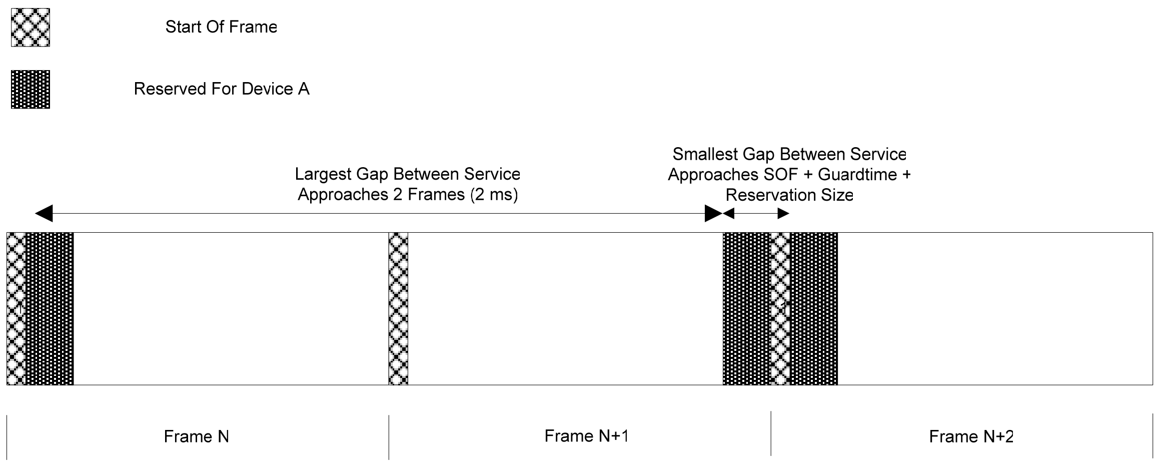
Start Of Frame
Reserved For Device A
Largest Gap Between Service Approaches 2 Frames (2 ms)
Smallest Gap Between Service Approaches SOF + Guardtime + Reservation Size
Frame N
Frame N+1
Frame N+2

Figure 4-19. Worst Case Service Interval Jitter For FS Isochronous Endpoint

In this case device A has an isochronous endpoint that has been admitted to the bus. The endpoint is guaranteed an amount of time on the bus each frame as shown in Figure 4-19. There is no guarantee on the location of the service opportunity. The service interval could approach 2 frames or be as small as the duration of the SOF, reservation and associated guard band. These extremes could happen in successive frames as shown in Figure 4-19. In typical operation the service attempts are evenly spaced as shown in Figure 4-20.

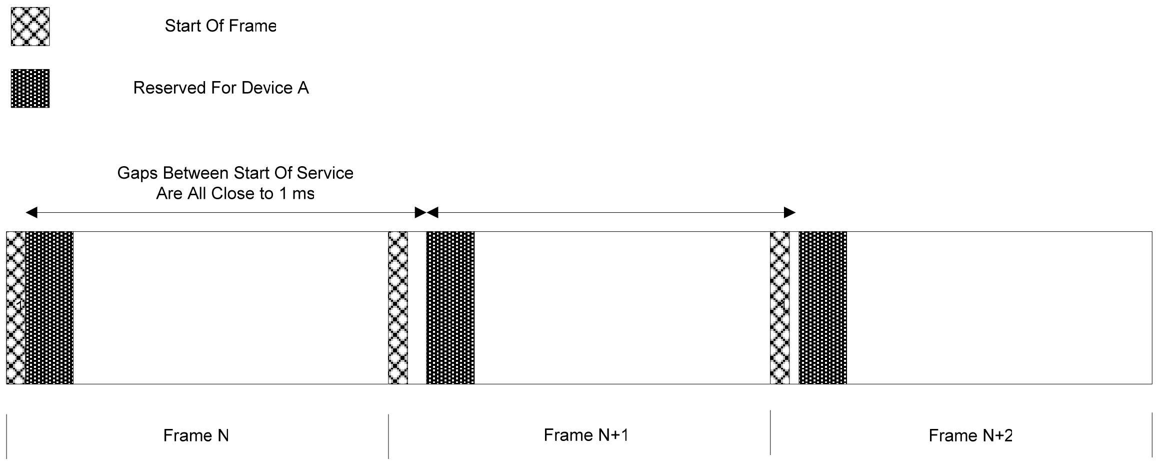

Figure 4-20. Typical Service Interval For FS Isochronous Endpoint

The basic operation of a HS wired USB isochronous endpoint is very similar to the FS case. The frame size changes to 125 microseconds. These divisions are known as microframes. There is still an indexed SOF packet sent by the host at the beginning of each microframe. Figure 4-21 shows the typical and worst case service gaps for a HS isochronous device with a service interval of 1 microframe.

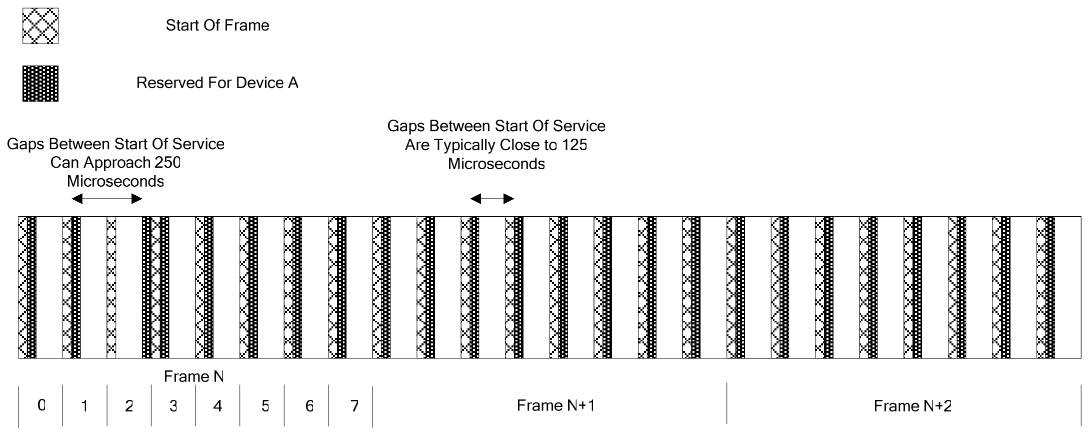
Start Of Frame
Reserved For Device A
Gaps Between Start Of Service Can Approach 250 Microseconds
Gaps Between Start Of Service Are Typically Close to 125 Microseconds
Frame N
0
1
2
3
4
5
6
7
Frame N+1
Frame N+2

Figure 4-21. Typical and Worst Case Service Intervals For HS Isochronous Endpoint

The typical gap between the start of service opportunities for the device will be 125 microseconds. However, the service can occur at any point in the service interval.

In wired USB the indexed frames (or microframes) provide a bus clock. Isochronous streams can be specified to start in specific frames (microframes). Typically, data that is produced in frame (microframe) X-1 by a wired isochronous IN device is sent across the bus in frame X. A typical wired USB isochronous endpoint needs only two frames (microframes) of buffering.

4.11.1.1 Wireless Service Intervals

As with wired USB, Wireless USB isochronous endpoints can receive service anywhere in the service interval. Furthermore, if a Wireless USB isochronous endpoint requires multiple packets per service interval (a maximum burst size bigger than one) the packet transmit/receive opportunities may be distributed as bursts from size 1 to the maximum bust size in any fashion throughout the service interval.

4.11.2 UWB Media Characteristics

4.11.2.1 Superframe Layout

The Wireless USB isochronous model is designed to work with a specific worst case superframe layout. The model does not assume that the Wireless USB host can reserve all of the time in the superframe. This section defines some terminology that divides the superframe into smaller structures. A superframe is shown Figure 4-22 divided into 16 regions called sections.

The sections are numbered from 0 to 15 from left to right in the superframe. Each section contains 16 MAS time slots of 256 microseconds each.

One section is reserved for transmitting beacons. This section is shown in Figure 4-22.

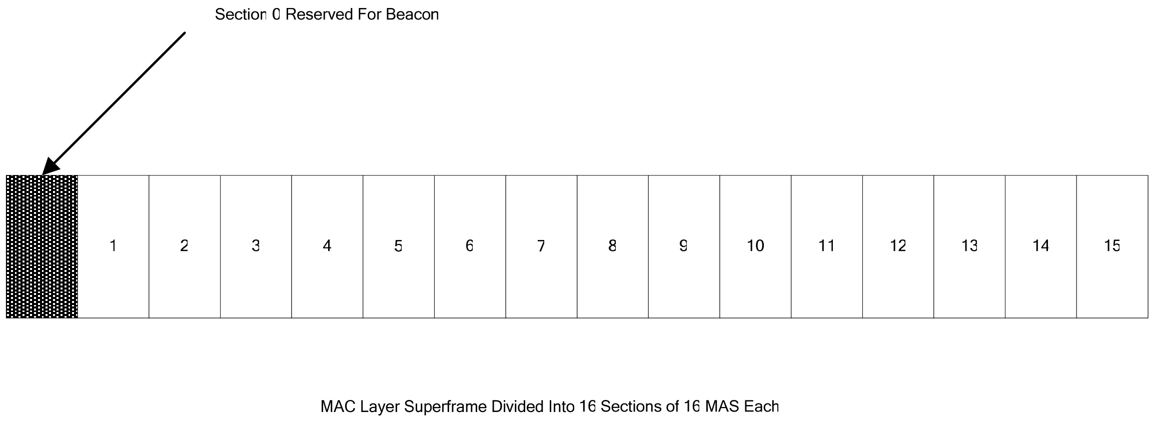
Section 0 Reserved For Beacon
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
MAC Layer Superframe Divided Into 16 Sections of 16 MAS Each

Figure 4-22. Beacon Reservation In MAC Layer Superframe

A Wireless USB device will not get service during the reserved beacon period. It also will not be able to get service during times when other Wireless USB devices have reservations. If the Wireless USB host is sharing the channel with other UWB devices (other Wireless USB hosts or non-USB UWB devices) there will be additional times when a Wireless USB isochronous device can not receive service. An isochronous device must account for service gaps to provide functionality in a variety of PHY channel conditions.

4.11.2.2 Worst Case Superframe Layout – Service Interval Bounds.

In some cases, the Wireless USB host will need to share the channel with other UWB devices that have already established their reservations. If there are no policies on the way UWB devices take reservations, the largest service interval bound approaches the size of the superframe. Figure 4-23 shows a situation where the service interval approaches the superframe size:

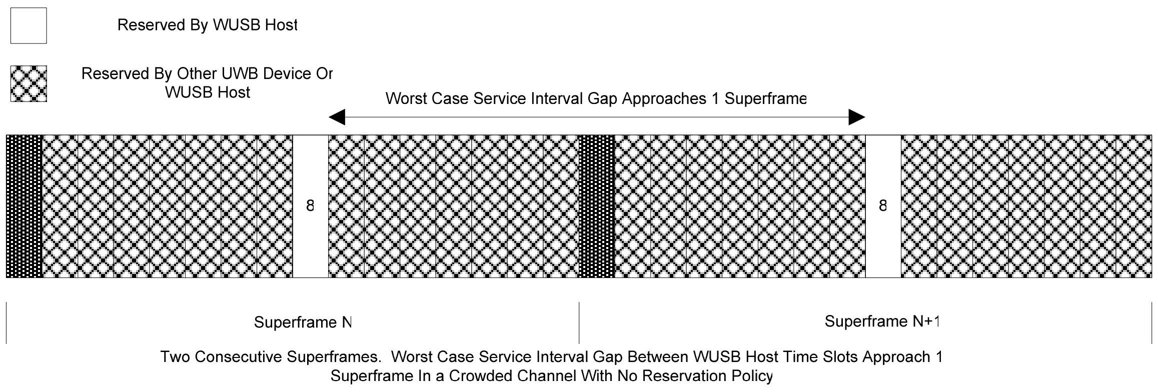
Reserved By WUSB Host
Reserved By Other UWB Device Or WUSB Host
Worst Case Service Interval Gap Approaches 1 Superframe
8
8
Superframe N
Superframe N+1
Two Consecutive Superframes. Worst Case Service Interval Gap Between WUSB Host Time Slots Approach 1 Superframe In a Crowded Channel With No Reservation Policy

Figure 4-23. Large Wireless USB Host Service Interval Gap

Under coexistence policies, a device that is admitted by the host can expect to receive service with a worst case service interval of 8.192 milliseconds. The worst case service interval occurs between the section at the end of one superframe and the section after the beacon period in the next superframe. Figure 4-24 shows a typical allocation for a Wireless USB host:

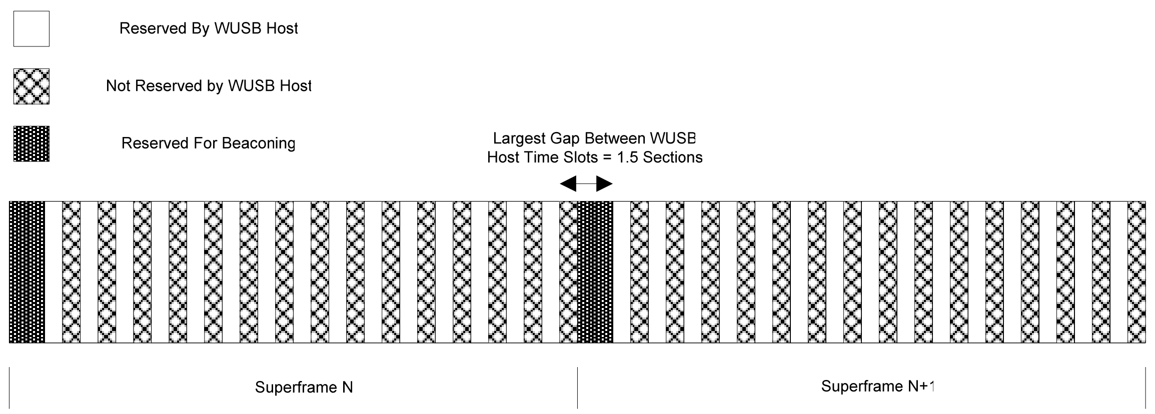
Reserved By WUSB Host
Not Reserved by WUSB Host
Reserved For Beaconing
Largest Gap Between WUSB Host Time Slots = 1.5 Sections
Superframe N
Superframe N+1

Figure 4-24. Typical Reservation for Wireless USB Host

In Figure 4-24 the Wireless USB host has reserved roughly half of time available for reservations in the superframe. The largest gap between Wireless USB host time slots occurs between the reservations before and after the beacon period. The gap is 6.144 milliseconds. The gaps between the other Wireless USB host time slots are 2.048 milliseconds. In cases where the Wireless USB host reserves a smaller percentage of the superframe the largest gap approaches 8.192 milliseconds for the time slots before and after the beacon. The largest gap between other Wireless USB host time slots (not across the beacon reservation) approaches 4.096 milliseconds.

4.11.2.3 Wireless Packet Error Rates

Error rates for wireless USB transfers can be much higher than wired USB error rates. Typical wireless average bit error rates can be as high as 10^{-4} with short term spikes to much higher error rates. For 1000 byte packets, this translate into an average packet error rate (PER) of 10%.

Wired USB bit error rates are required to be 10^{-9} or better. The wired USB protocol can ignore the possibility of errors (no handshaking to indicate if data was successfully received in the wired USB isochronous protocol) and still provide PER of 10^{-6} or better for 1000 byte packets. If the wireless USB isochronous protocol were to continue to not use handshaking, it would have to send each packet on the order of 6 times to guarantee matching wired reliability. From an efficiency standpoint for the wireless USB bus, this approach is not feasible. Therefore, the wireless USB isochronous protocol uses handshaking.

4.11.3 Wireless USB Isochronous Transfer Level Protocol

At the transaction level, the wireless USB isochronous protocol is almost identical to the wireless USB bulk protocol. The protocol is defined in the protocol chapter. Wireless USB defines mechanisms for isochronous function endpoints to allow error reporting when data is discarded and to allow backwards compatibility with some existing class drivers and applications that support wired USB isochronous endpoints.

4.11.4 Wireless USB Isochronous IN Example

This section walks through a simple high level example to illustrate the basic steps that occur in the operation of a wireless USB isochronous IN endpoint. When a device with an isochronous IN endpoint exchanges configuration information with the wireless USB host, it reports the amount of buffering that it has for use with each isochronous IN endpoint. The buffering amount is reported in the *wMaxStreamDelay* field in the Endpoint Companion descriptor, see Section 7.4.4. As shown in Figure 4-25, the host must set aside a larger amount of buffering for working with the endpoint. The buffer set aside by the host must be at least one max packet size larger than the buffer reported by the device for the isochronous IN endpoint. Note: Wireless USB Host refers generically to the system containing the wireless USB host controller.

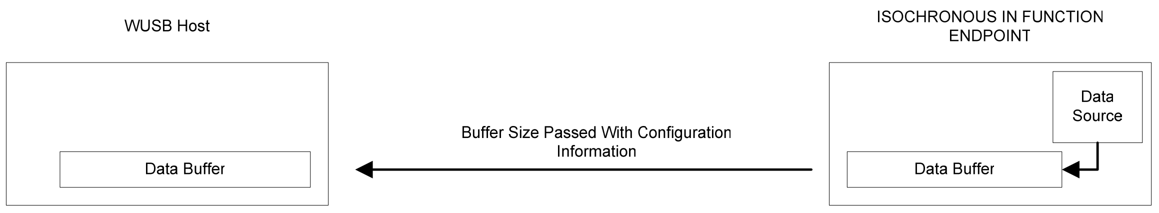
WUSB Host
Data Buffer
Buffer Size Passed With Configuration Information
ISOCHRONOUS IN FUNCTION ENDPOINT
Data Source
Data Buffer

Figure 4-25. Configuring Wireless USB Isochronous IN Endpoint

The isochronous IN endpoint in this example produces an average of 30 1024 byte packets every 65.536 milliseconds. The device contains enough buffering to store 8 of these packets. The endpoint descriptor requests a service interval of 4.096 milliseconds, a maximum packet size of 1024 bytes, a maximum burst size of 2, and a maximum stream delay of 16.384 milliseconds. Note that a 4.096 millisecond service interval provides service only 15 times every 65.536 milliseconds due to the reserved period for beacons.

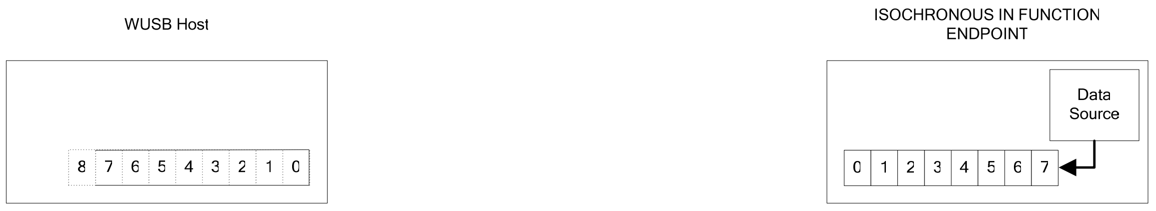

Figure 4-26. Isochronous IN Endpoint and Host Buffering

Figure 4-26 shows the size of the isochronous IN endpoint and corresponding host buffers. The buffer positions are indexed for reference throughout the example. The example isochronous IN endpoint produces data continuously when powered.

When the device buffer fills the oldest data in the buffer is thrown away to make room for the new data. When the first request for data from the wireless host occurs, the endpoint buffer is full.

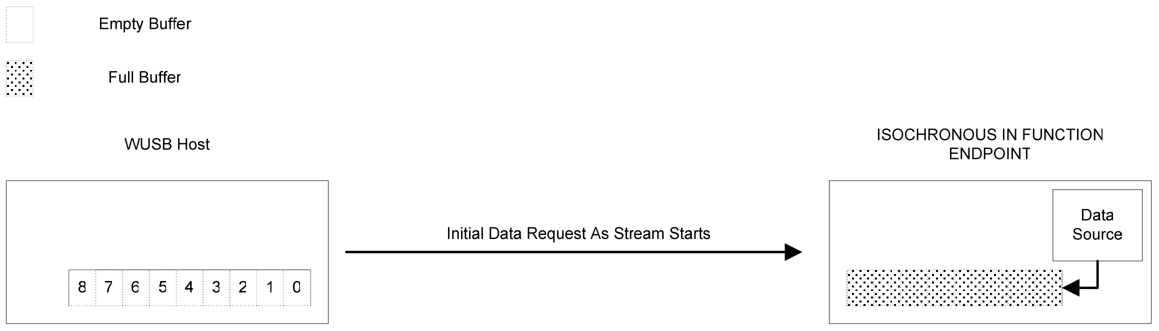
Empty Buffer
Full Buffer
WUSB Host
8 7 6 5 4 3 2 1 0
Initial Data Request As Stream Starts
ISOCHRONOUS IN FUNCTION ENDPOINT
Data Source

Figure 4-27. Initial Data Request From Isochronous IN Endpoint

Figure 4-27 shows the initial request for data. When the first request for data comes from the host, the device responds with data from its buffer. There are several ways that the isochronous endpoint could handle the initial response for data. In Figure 4-28, the isochronous IN endpoint sends the oldest data in its buffer in response to the first request for data when the stream starts.

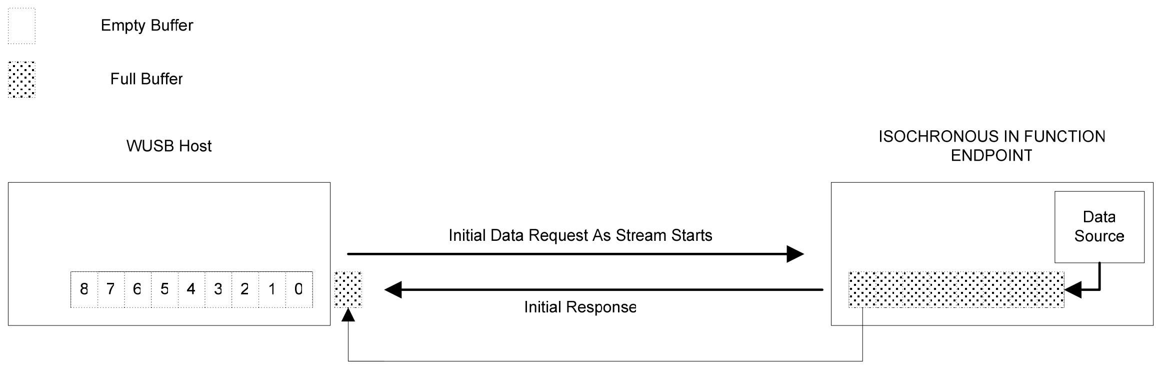

Figure 4-28. Isochronous IN Endpoint Sends Oldest Data In Buffer In Response To Initial Request

Responding with the oldest data in the buffer leaves the function endpoint susceptible to errors early in the stream. If the initial packet is successfully transferred – but several errors occur shortly thereafter, the function endpoint may have to throw away data. An alternate approach is shown in Figure 4-29.

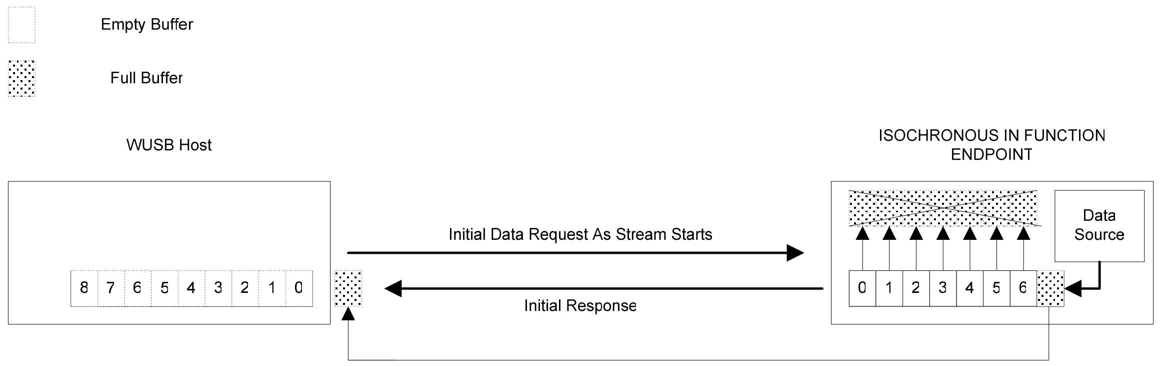

Figure 4-29. Isochronous IN Endpoint Sends Newest Data In Response To First Request And Clears Buffers

In this case the function endpoint sends the newest data in its buffer in response to the first request and then discards all other data. Subsequent data is stored normally. The function endpoint may now buffer up to 8 packets before discarding data – even during the initial startup of the stream.

Note: There are a variety of options for when an isochronous IN endpoint starts storing data in its buffer and how it responds to the first isochronous IN request. A device designer should keep in mind that some options will provide better error tolerance as the stream starts than others.

The host will continue to request data. In this implementation, when the host has received enough data to fill its initial buffer – the application will begin to consume data. Figure 4-30 shows the state of the system when data consumption begins.

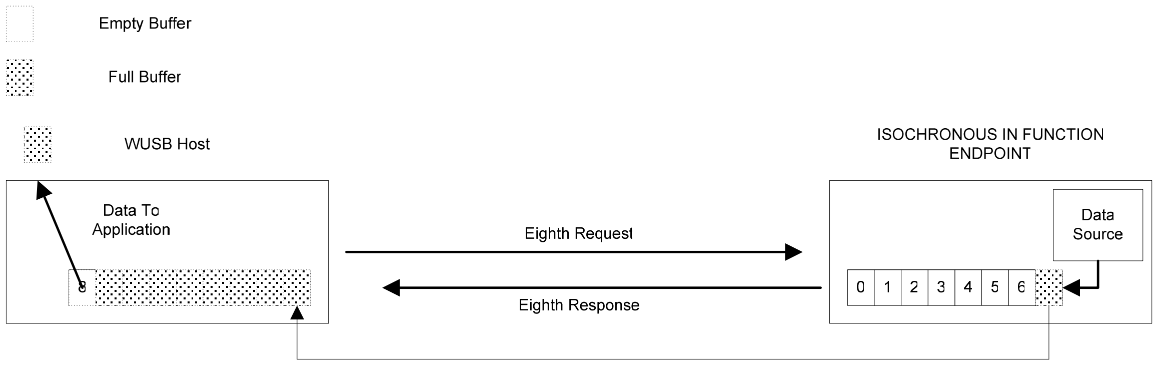

Figure 4-30. Data Consumption Begins

The host buffer fills and the oldest data in the buffer is sent to the application. Each data packet sent by the isochronous IN device contains header information that indicates the Wireless USB Channel time when the data was produced. These presentation times are used by the host to apply the data to specific locations in application buffers.

Unless there have been significant errors during the start up of the stream, the device buffer will be close to empty. A delay of approximately 16 milliseconds has been added to the system by the buffering and stream startup conditions. This delay provides tolerance to short term errors and glitches in the stream as operation continues. If the device had desired greater tolerance to short term error bursts and glitches it could use additional buffering. The amount of buffering to use is a device decision. Deciding on the amount of buffering is a tradeoff between cost, error tolerance, and the amount of acceptable delay/latency in the stream. The tradeoff is discussed in more detail in Section 4.11.6.

During normal operation the isochronous IN endpoint buffer will stay relatively empty and the host buffer will stay relatively full. If there is a prolonged period where the error rate is high the endpoint buffer will begin to fill and the host buffer will begin to empty. As long as the error rate decreases again before the endpoint buffer overflows, the system will recover because the host allocates guaranteed time for retries each service interval as part of the bandwidth reservation for the endpoint. However, if the errors continue the endpoint buffer will eventually overflow and the host will be unable to provide data to the application.

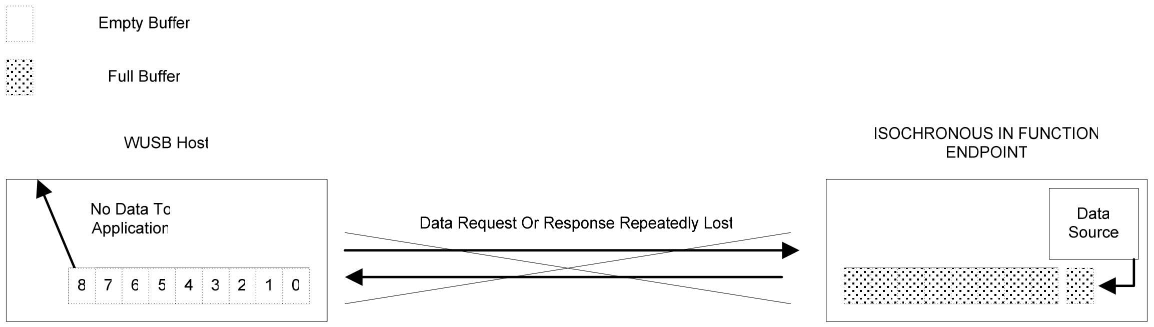

Figure 4-31. Data Must Be Discarded By Isochronous IN Function Endpoint

Figure 4-31 shows the case where the error rate has been significant for a prolonged period. The endpoint produces its next data packet but has no place to store it. At this point the isochronous IN endpoint must discard its oldest data to store the new data. This is the only case where data is discarded with an isochronous IN stream. The device will continue to try to send data until it is forced to discard data when its buffer overflows. Section 4.11.9 examines error handling in more detail. When an isochronous IN endpoint discards data it must re-use the burst sequence number associated with the discarded packet. The host processes data packets based on their presentation times and will still place data in the correct locations in application buffers. There are methods for communicating the amount of information that has been discarded and options for attempting to prevent a buffer overflow from occurring.

4.11.5 Wireless USB Isochronous OUT Example

This section walks through a simple high level example to illustrate the basic steps that occur in the operation of a wireless USB isochronous OUT endpoint. The wireless USB isochronous OUT example is similar to the isochronous IN example in reverse. When a device with an isochronous OUT endpoint exchanges configuration information with the wireless USB host, it reports the amount of buffering that it has for use with each isochronous OUT endpoint. The buffering amount is reported in the *wMaxStreamDelay* field in the endpoint companion descriptor described in 7.4.4.

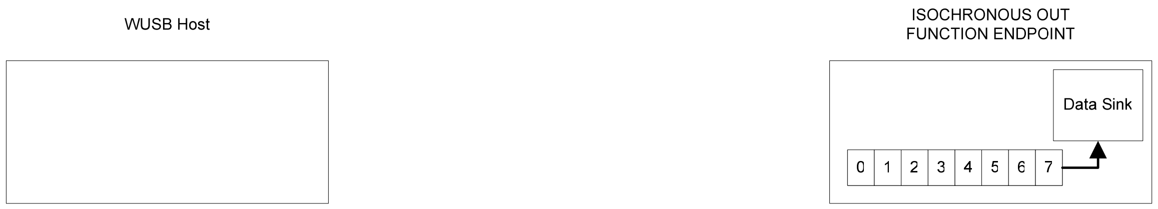

Figure 4-32. Configuring Wireless USB Isochronous OUT Endpoint

The isochronous OUT endpoint in this example produces an average of 30 1024 byte packets every 65.536 milliseconds. The device contains enough buffering to store 8 of these packets. The endpoint descriptor reports a maximum packet size of 1024, a maximum burst size of 2, a service interval of 4.096 milliseconds, and a *wMaxStreamDelay* of 16.384 milliseconds.

Figure 4-32 shows the size of the isochronous OUT endpoint. The buffer positions are indexed for reference throughout the example. The host buffering must be able to store enough data for the isochronous OUT stream that the host will only discard data when it is no longer usable by the device. The rest of this example describes exactly when the host will discard data. Both buffers are empty before the stream starts. When the Wireless USB host first receives data for the stream it attempts to send it to the isochronous OUT endpoint.

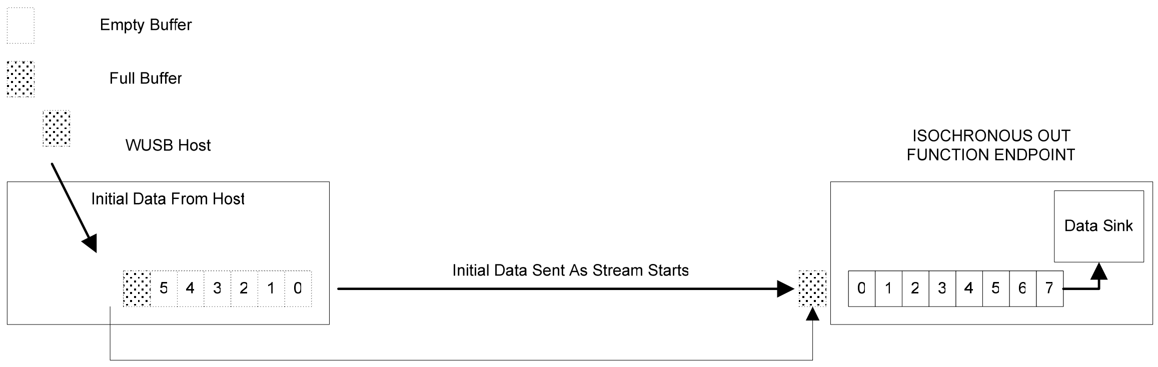

Figure 4-33. Initial Data Sent To Isochronous OUT Function Endpoint

Figure 4-33 shows the initial data as it is received by the host and sent to the Wireless USB isochronous OUT endpoint. The wireless USB host will continue to send data as it receives data from the application. Each data packet sent to the isochronous OUT function endpoint contains header information that indicates the Wireless USB Channel time when the data is intended for consumption. These presentation times can be used by the device to place data in the proper buffer locations and determine when data should be consumed by the function endpoint. When the isochronous OUT function endpoint has data with the current presentation time it will consume the data. The endpoint function must examine the presentation time in the packet with the first sequence number when the stream starts. The endpoint function begins consuming data when the Wireless USB channel time reaches the presentation time of the initial data. The host must attempt to apply initial presentation times and schedule the stream start up such that the isochronous OUT function endpoint buffering will be full when it begins to consume data based on the initial presentation time. This allows the isochronous output buffering to provide the maximum delay and short term error tolerance for the stream. Figure 4-34 shows the state of the system when data consumption by the function on the isochronous OUT function endpoint begins.

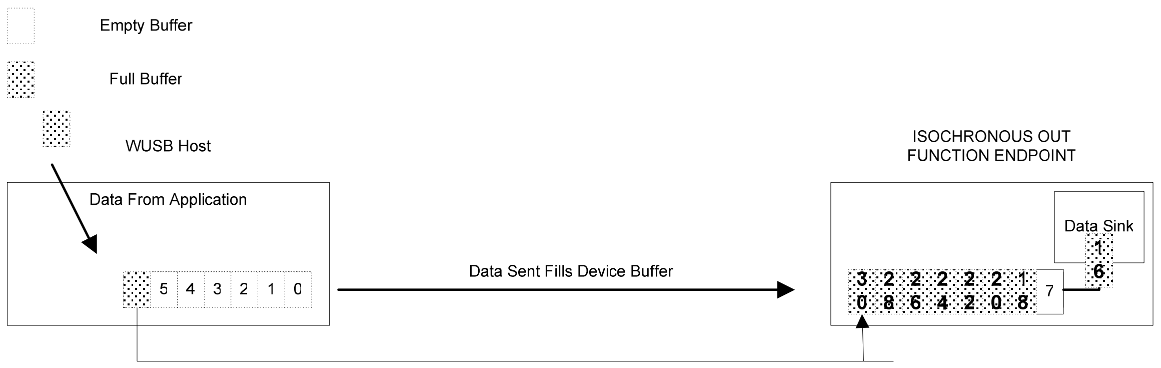

Figure 4-34. Data Consumption Starts For Isochronous OUT Function Endpoint

When the stream starts the host starts sending the first packet to the isochronous OUT function endpoint at Wireless USB channel time zero. The packet is marked with the presentation time 16 milliseconds (presentation times are rounded to even milliseconds for simplicity in the example). The isochronous OUT function endpoint buffer fills. When the Wireless USB channel time reaches 16 milliseconds the endpoint function begins to consume data starting with the first packet received with presentation time 16. Unless there have been significant errors during the start up of the stream, the isochronous OUT function endpoint buffer will be full when it begins consuming data. A delay of approximately 16 milliseconds has been added to the system by the buffering and stream startup conditions. This delay provides tolerance to short term errors and glitches in the stream as operation continues. If the isochronous OUT function endpoint had desired greater tolerance to short term error bursts and glitches it could have used additional buffering. The amount of buffering to use is a device designer's decision. It is a tradeoff between cost, error tolerance, and the amount of acceptable delay/latency in the stream. The tradeoff is discussed in more detail in section 4.11.6.

During normal operation the isochronous OUT endpoint buffer will stay relatively full. If there is a prolonged period where the error rate is high the presentation time for the oldest un-transmitted data on the host will get closer to the current Wireless USB channel time and the isochronous OUT endpoint buffer will begin to empty. As long as the error rate decreases again before the isochronous OUT endpoint buffer underflows, the system will recover. However, if the errors continue the device buffer will eventually underflow. When errors occur the isochronous OUT endpoint will not have data to consume and the host will discard data if it has not been able to transmit the data and the Wireless USB channel time reaches the presentation time for the data.

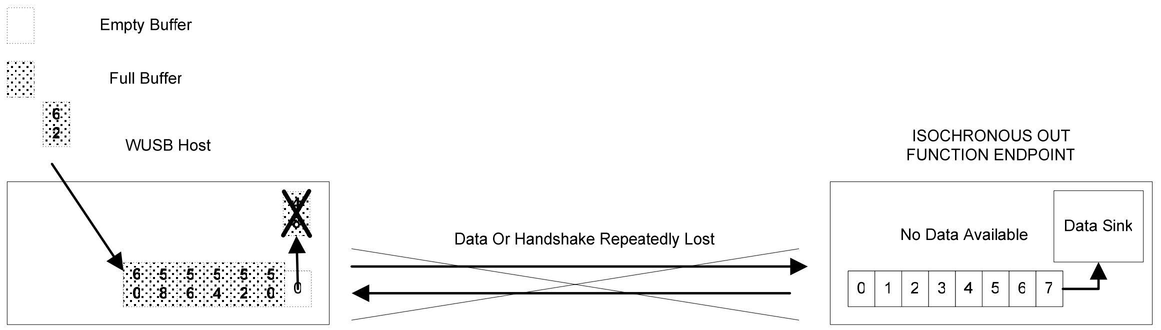

Figure 4-35. Buffer Overflows and Host Must Discard Data For Isochronous OUT Function Endpoint

Figure 4-35 shows the case where the error rate has been significant for a prolonged period. The Wireless USB channel time reaches 48 milliseconds and the host must discard the packet with a 48 millisecond presentation time. At this point the host must discard data. The host only discards data when the data is late based on its presentation time. The host will continue to try to send data until it is forced to discard data when the presentation time for data that has not been transmitted has expired. The host must discard data to an isochronous OUT function endpoint if the presentation time for the data has expired. Section 4.11.9 examines error handling in more detail. There are methods for communicating the amount of information that has been discarded and options for attempting to prevent host discards from occurring.

4.11.6 Choosing an Isochronous IN or Isochronous OUT Endpoint Buffer Size

Buffer size is an application specific decision involving the following factors:

1. Desired short term error tolerance.
2. Cost.
3. Acceptable stream delay/latency.

At a minimum an isochronous OUT function endpoint must have enough buffering associated with it to tolerate the longest possible over the air latency between service attempts – 8.192 milliseconds. Additional buffering provides additional short term error tolerance.

Warning: If only 8.192 milliseconds of buffering is provided the stream may fail with only a single error on an over the air transmission if it occurs immediately before or after the longest gap in over the air service.

Additional error tolerance is provided by using additional buffering to provide delay in the stream. If the stream has latency requirements, the amount of buffering that can be added is limited.

Note: An isochronous stream may have different latency requirements in different use situations. An isochronous device can provide alternate settings that report buffer sizes less than the physical buffering available. The endpoint must only use the amount of buffering (delay) reported in the selected alternate setting.

When there are no latency requirements, or the latency requirements allow a large amount of latency, the implementer must make a trade-off between short term error tolerance and buffering cost.

The endpoint buffer size is reported as *wMaxStreamDelay* in the Wireless USB Endpoint Companion descriptor, see Section 7.4.4. The *wMaxStreamDelay* parameter is reported in units of time. For an isochronous IN endpoint the value indicates the smallest amount of time it will take to fill the buffer from empty and cause the first discard if no data is being drained from the endpoint buffer by the host. For an isochronous OUT endpoint buffer, *wMaxStreamDelay* represents the smallest range in presentation times for data that can completely fill the endpoint buffer if it is not passing data to the data sink. The time is measured specifically from the first (smallest) presentation time in the buffer to the presentation time of the first packet that could not be accepted due to lack of buffering.

4.11.7 Isochronous OUT endpoint receiver implementation options

A host must discard any packets whose presentation time has expired. When a host discards data packets during a burst to an isochronous endpoint it must not reuse the same sequence numbers to transmit new data packets and must start including in its MMC a Discard Packet IE (refer to Section 7.5.10).

How an Isochronous OUT endpoint receiver determines which packets have been discarded by the host is implementation specific. This section presents some informative examples of receiver implementations.

Figure 4-36 shows the transmit and receive window characteristics for an example receiver burst engine that will be used as the basis for discussing different implementation options.

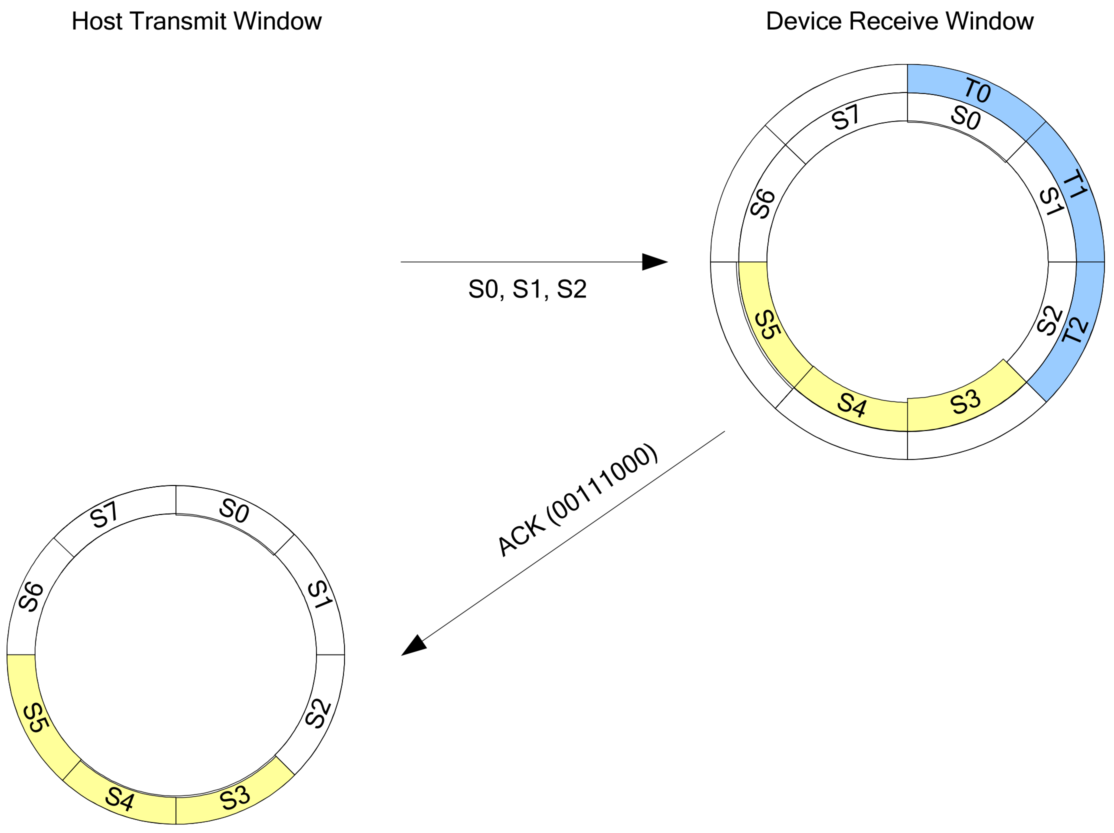

Figure 4-36 - Receive window for isochronous OUT function endpoint with a burst size of 3 and maximum burst sequence number of 8.

Each isochronous data packet is associated with a burst sequence number S_N, and a presentation time T_N. Burst sequence-numbers are used to keep transmit and receive windows synchronized as described in Section 5.4. Presentation-time is used by the host for determining when data packets should be discarded and can be used by the device for determining when the data should be consumed. A host must discard isochronous packets when their presentation time has expired. When a discard occurs the host must start transmitting a Wireless USB Isochronous Packet Discard IE (WISOCH_DISCARD_IE) containing the first sequence number of the isochronous data packet that the host will transmit for that endpoint (*bFirstReceiveWindowPosition*), together with the new value for the transmitter's window (*bmDeviceReceiveWindow*), the number of isochronous data segments that were discarded (*wNumberDiscardedSegments*), and the number of isochronous data packet that were discarded (*wNumberDiscardedPackets*). Every time the host starts a new sequence of discards (i.e. after it receives a valid handshake packet from the device's endpoint) it must increment the identifier for the discard sequence (*bDiscardID*).

The WISOCH_DISCARD_IE is then used by the device to determine if one or more data packets were discarded at the host's side and to reconstruct the gaps between received packets.

In Figure 4-36 the device has correctly received the data packets associated with S_0, S_1 and S_2 and notified the host that it is ready to receive that next burst of data (S_3, S_4, S_5).

After the next burst the data packet associated with S_3 is not received. The other two packets associated with S_4 and S_5, are received successfully as shown in Figure 4-37.

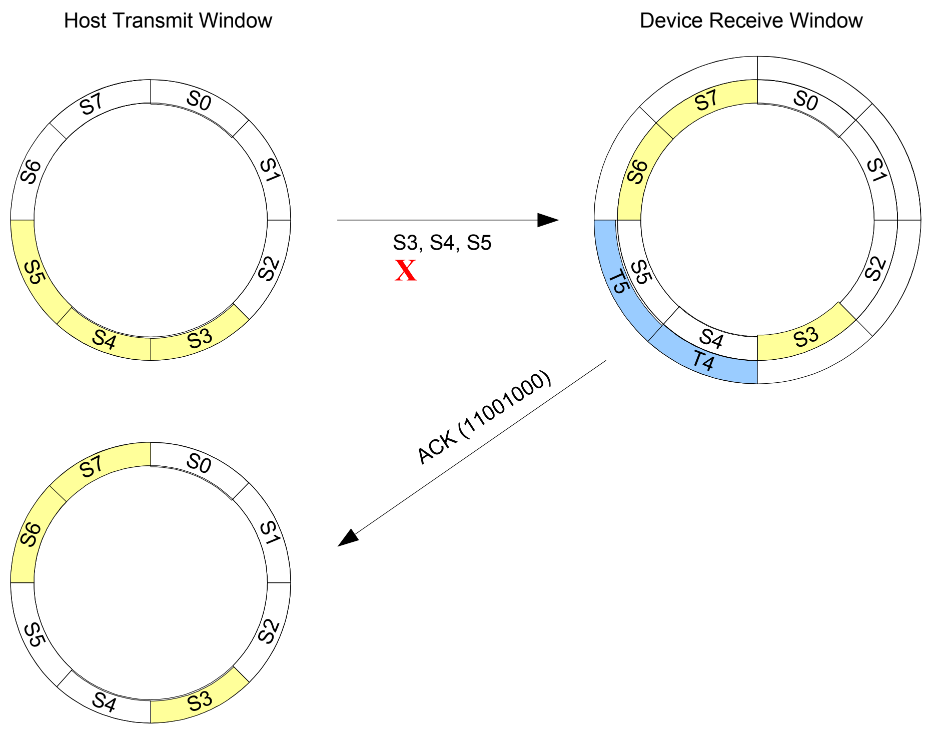

Figure 4-37 – Receive window after 2 of 3 packets are received successfully in second burst.

At this point, the presentation time T_3 expires and the host discards the packet associated with S_3.

It will construct a WISOCH_DISCARD IE containing the following values:

- *bFirstReceiveWindowPosition* = 6
- bmDeviceReceiveWindow = 11000001
- wNumberDiscardedPackets = 1
- bDiscardID = x (must be a different value than last WISOCH_DISCARD_IE sent to the function endpoint)

The host will transmit the WISOCH_DISCARD IE in the next MMC together with a W_{DR}CTA (for transmission of data packets S_6, S_7 and S_0) and a W_{DT}CTA for handshake.

4.11.7.1 Presentation Time aware implementation

This section describes an implementation of an Isochronous OUT endpoint receiver for a device capable of processing the presentation time associated with each isochronous data packet. In this implementation, after receiving the WISOCH_DISCARD IE the device examines the *bFirstReceiveWindowPosition* field to determine the first packet that will be transmitted by the host.

This information will allow the burst engine to properly update the pointer for its receive window.

The device will also reset the window to the state specified in the *bmDeviceReceiveWindow* field, before the start of the next W_{DR}CTA in which the endpoint will receive the next burst of data. The data sink will have retrieved all the data available in the buffer (in the example considered: the data packets associated with S_4 and S_5) and will process the data at the appropriate presentation time (in the example considered: T_4 and T_5). If the stream is periodic, the data sink will see a gap between the presentation times of received packets and will be able to determine how many data packets were discarded by the host (in the example considered: the data packet associated with T_3).

The next burst transmitted by the host will contain data packets containing timestamps T_6, T_7 and T_8 associated respectively with S_6, S_7 and S_0.

If the packets are all received correctly, the receive window will look like the one depicted in Figure 4-38.

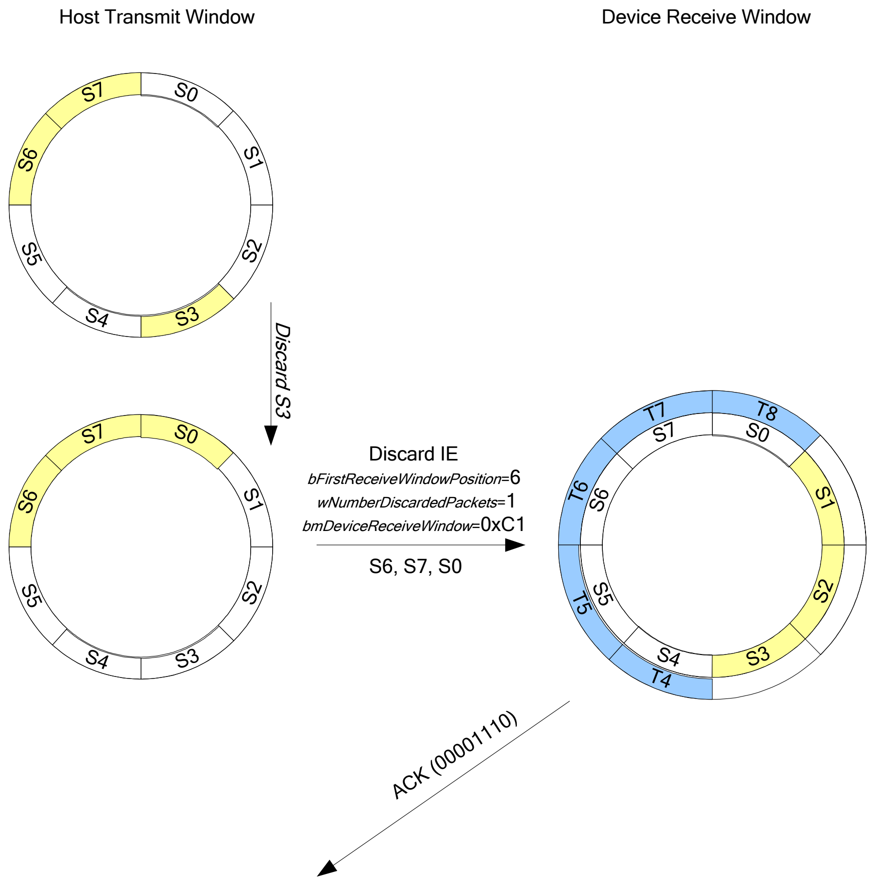

Figure 4-38 Receive window after receiving WISOCH_DISCARD IE and burst for sequences S_6, S_7, and S_0 correctly.

4.11.7.2 Presentation time aware implementation with "false" acknowledgement

An ISO receiver endpoint implementation may produce "false" acknowledgments in anticipation of a discard notification by the host.

In the case where the data packet associated with S_3 is not received and the two packets associated with S_4 and S_5 are received correctly, this type of implementation will attempt to acknowledge successful reception of all three packets before the expected presentation time for the original packet associated with S_3 expires (and therefore before the host starts including a WISOCH_DISCARD IE in its MMC). The updated receive window for this case is shown in Figure 4-39.

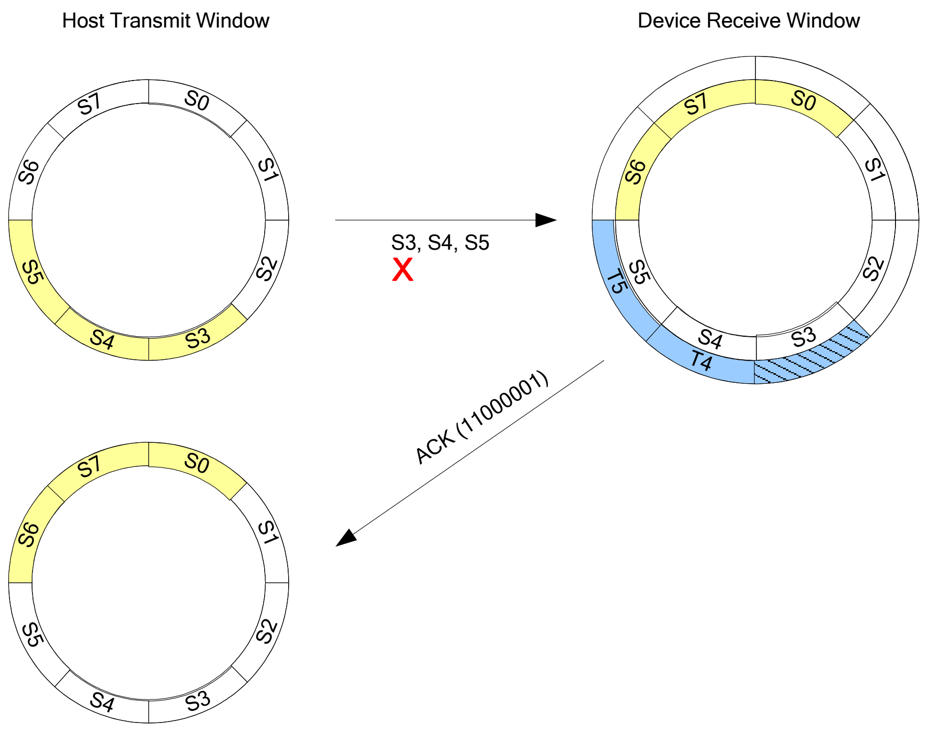

Figure 4-39 Receive window after false acknowledgement of S_3.

Given the periodic nature of the traffic under consideration, the device may know the presentation time that will be associated with each burst sequence number. This information may be used by the function endpoint to transmit “false” acknowledgements for stuck packets before they expire to keep the flow going. With this type of implementation a host will not receive accurate packet error rate information for the link. However, the host may still have knowledge of the link condition by tracking how close the packet presentation times are to the Wireless USB channel time before the packets are acknowledged. This type of implementation may not be possible if data does not follow predictable patterns. It could also lead to some amount of unnecessary data discard in error scenarios.

Although it is not guaranteed that the host will not begin a discard sequence (the host could fail to receive the false acknowledge handshake), with this type of implementation a device will still be able to re-synchronize its receive window with host’s transmissions without processing the WISOCH_DISCARD IE.

4.11.7.3 Presentation time unaware implementations

A device’s ISO endpoint receiver implementation may not be capable of processing the timestamp associated with each Isochronous data packet. In this case, the device must be able to resynchronize with the host based on the information contained in the WISOCH_DISCARD IE and according to the mechanism described in Section 4.7.5.

The data sink may delay the consumption of received packets by as many periods as the number of packets discarded (as indicated in the WISOCH_DISCARD IE).

During a prolonged discard sequence the device may receive data packets subsequently discarded by the host and accounted for in the WISOCH_DISCARD IE (the host did not receive the handshake packets from the device).This can be dealt with in several ways.

In a basic implementation, like the one illustrated in Figure 4-40, when receiving a WISOCH_DISCARD IE the device would flush all data which has not been consumed by the data sink, and reset its receive window as indicated by the discard IE.

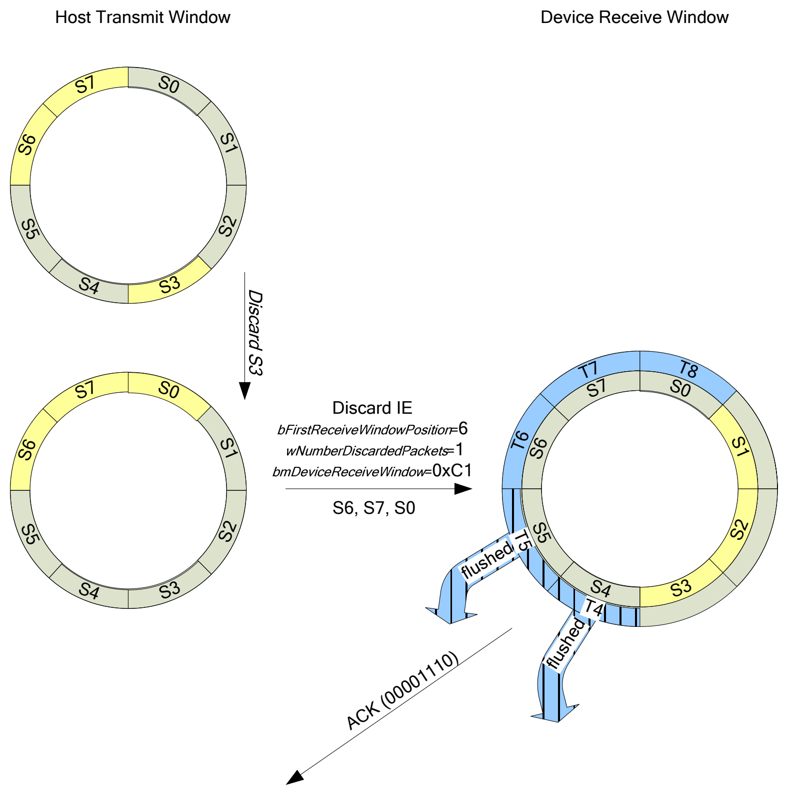

Figure 4-40 Receive window after receiving WISOCH_DISCARD IE and burst for sequences S_6, S_7, and S_0 correctly.

A more sophisticated device could implement some mechanism (e.g. a counter) to correctly determine how many of the packets indicated by the host as discarded were received in past burst transmissions. It will then update accordingly its current receive window. This type of device must know the expected presentation time for each packet based on its sequence number.

An example of how this type of device would update its receive window is depicted in Figure 4-41.

In this type of implementation data is never discarded after being received successfully.

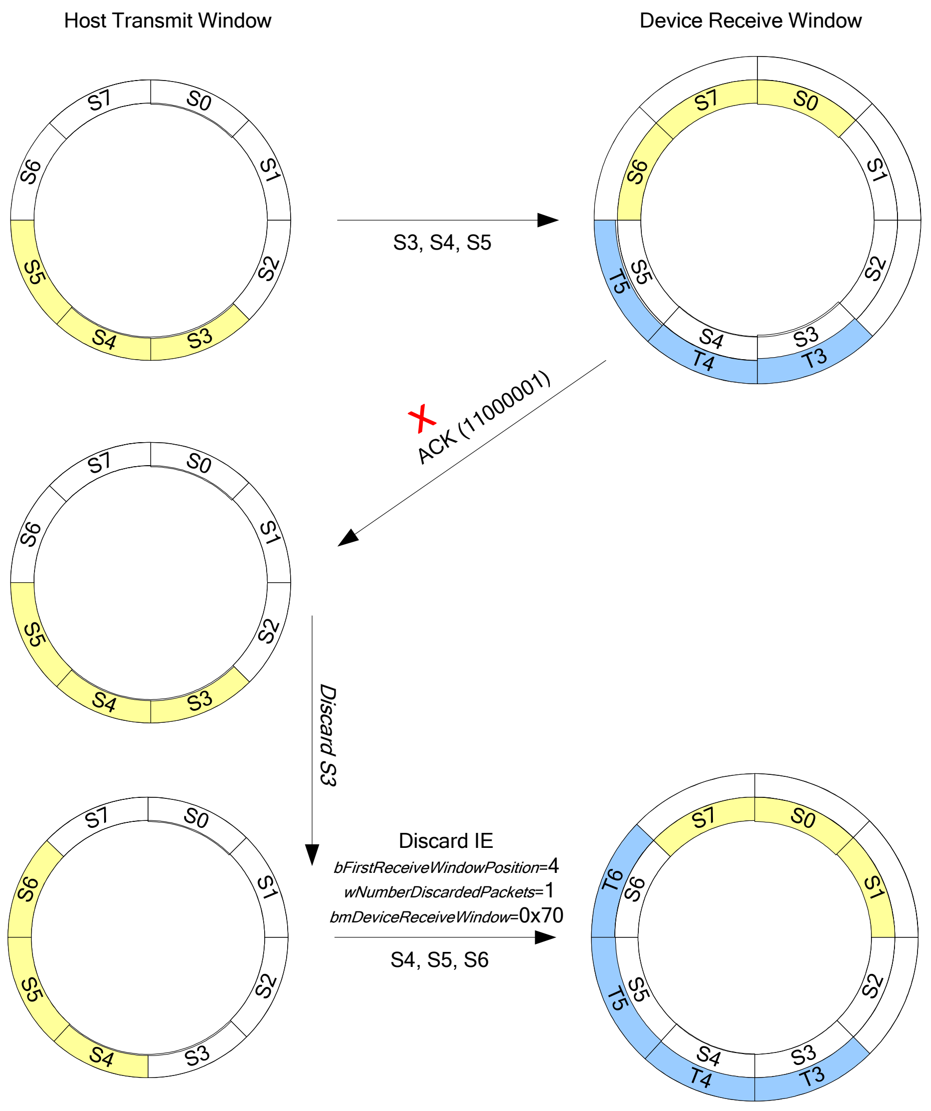

Figure 4-41 Receive window after receiving a WISOCH_DISCARD IE subsequent to a lost ACK.

4.11.8 Synchronization

To allow for synchronization of clocks between Wireless USB devices, the Wireless USB host must provide a Wireless USB channel time stamp in each MMC. This time stamp provides similar functionality for isochronous function endpoints needing synchronization as the indexed (micro)SOFs provided in USB 2.0. The time stamp format and accuracy requirements are described in Sections 4.3.1, 4.3.2 and 4.3.3.

4.11.8.1 Synchronizing a Stream Start Time

The unreliable wireless media creates a problem for stream synchronization. In wired USB the time at which an isochronous IN function endpoint starts producing data is deterministic. Consider a typical wired isochronous IN device with two (micro) frames of buffering. The (micro) frame in which the first data request is made is specified as frame (microframe) X. The data returned by the device in response to the request in frame X is produced starting at the beginning of frame X-1.

In wireless USB the time when data is first produced for a wireless USB isochronous IN stream is less deterministic. The layout of reservations in the superframe (if the Wireless USB host is sharing it with other devices) and the potential for several successive packet errors, create uncertainty in the starting time for data production by the isochronous IN endpoint. For some stream types this uncertainty is acceptable. If a stream needs to specify the exact starting time of data production an out of band mechanism can be used.

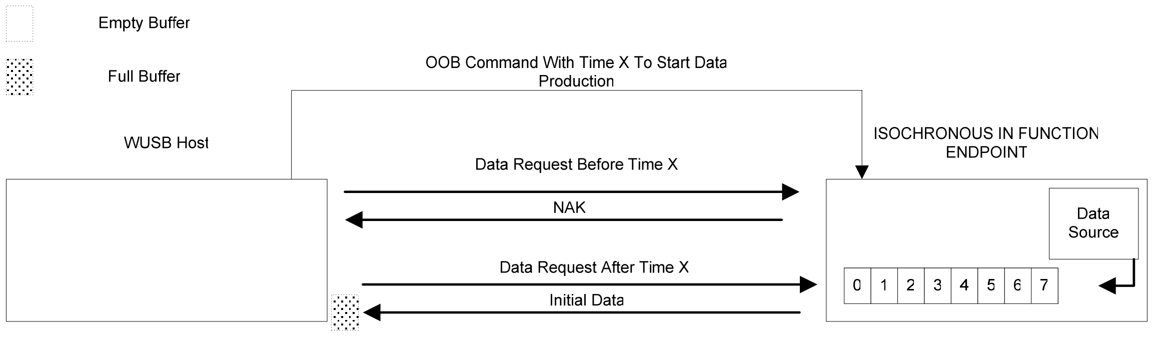
Empty Buffer
Full Buffer
WUSB Host
OOB Command With Time X To Start Data Production
ISOCHRONOUS IN FUNCTION ENDPOINT
Data Request Before Time X
NAK
Data Source
Data Request After Time X
Initial Data
0 1 2 3 4 5 6 7

Figure 4-42. Isochronous Endpoint Device Uses Out Of Band Mechanism To Specify Time For Data Production To Start

In Figure 4-42 an isochronous IN endpoint device is shown. The device does not start data production until time X. This time is specified via an out of band mechanism before the stream starts. The exact form of the out of band start time request is a device specific. If the device receives a request before time X it will NAK. At time X it will start producing data and respond normally to subsequent requests. An out of band request can also be used to synchronize multiple streams.

4.11.9 Error Handling Details

This section provides additional details on handling errors in the Wireless USB isochronous model. The primary error case in the Wireless USB isochronous model occurs when the transmitter must discard a packet. This can occur when the physical buffer overflows for an isochronous IN function endpoint or when a packet can not be transmitted to an isochronous OUT endpoint by the host before its presentation time expires. In these cases there must be a mechanism by which the receiver is informed that the overflow occurred. The receiver must also be able to determine the severity of the overflow. The following sections describe transmitter behavior when a buffer overflow occurs in more detail.

4.11.9.1 Reporting Data Discarded At the Transmitter

The basic mechanism for reporting discarded data in the isochronous model is accomplished by embedding isochronous header information in the data packets. Each isochronous data packet sent has an isochronous header that includes a presentation time and one or more data segments. If the packet contains multiple segments, the presentation time is associated with the initial data segment. The presentation time for additional segments is inferred from the *bInterval* value for the endpoint. For an isochronous IN endpoint, the presentation time represents the Wireless USB Channel time when the data was produced. For an isochronous OUT function endpoint the presentation time indicates the Wireless USB Channel time when the data is to be consumed. If an isochronous IN function endpoint discards a packet before it can be successfully transmitted it does not reuse the presentation time of the discarded packet. This allows the receiver to learn when packets are discarded and how many packets have been discarded. The specific format of the data packets is defined in the protocol chapter.

Figure 4-43 shows an example where data must be discarded by a Wireless USB isochronous IN endpoint.

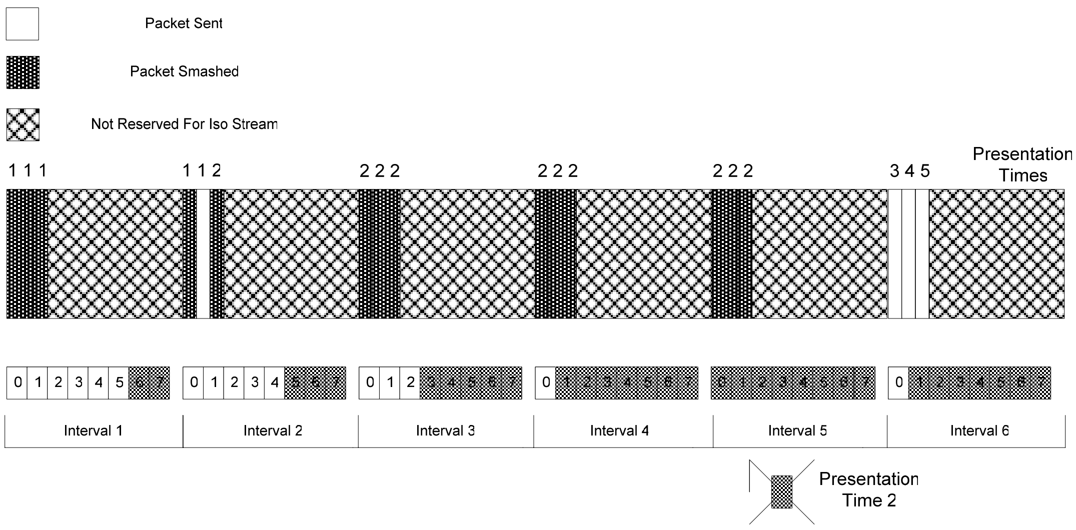

Figure 4-43. Isochronous IN Function Endpoint Transmit Buffer Overflows

The isochronous IN endpoint in this example produces an average of 2 packets every 4.096 milliseconds. The isochronous IN endpoint in this example has a maximum burst size of one. Each packet is 1 kilobyte in size. The packets are produced at regular intervals. The device contains enough buffering to store 8 of these packets. The WUSB host has reserved enough time to allow the device to send 3 packets during the slice of time the WUSB host controls each 4.096 millisecond interval. The stream has been up and running without errors until interval 1. In interval 1, all three attempts to receive a packet from the endpoint are corrupted. The presentation time for this packet is presentation time #1. The endpoint buffers the packet with presentation time #1 and the additional packet it created that interval with presentation time #2. In the next interval (interval 2), the packet with presentation time #1 is sent successfully on the second attempt. The first attempt to send the packet with presentation time #2 also fails. At the end of interval 2 the device now has three packets buffered. During intervals three and four all three attempts to send data fail. At the end of interval 4 the endpoint now has 7 packets stored in its buffers. During interval 5, all three attempts to send the packet with presentation time #2 fail. At some point during interval 5, the endpoint has produced another packet of information and has nowhere to store it. It must discard the packet of information with presentation time #2 (oldest) to store the new packet. During interval 7, all three transfer attempts are successful (packets 3, 4, and 5). The device buffer has one empty space at the end of the interval. If the link quality remains good the stream will recover and reach the state where the isochronous IN endpoint buffer is nearly empty. The receiver will be able to detect that a sample produced at presentation time #2 was not received.

4.11.9.2 Discarding Data during A Burst

An isochronous stream may use a maximum burst size greater than one. Data may be discarded when the transmitter buffer overflows or when a presentation time expires on a packet the host is attempting to transmit to an isochronous OUT function endpoint. When a packet is discarded by an isochronous IN function endpoint, it simply begins to try to send the next packet using the same burst information (burst sequence number) as the discarded packet. Only the presentation time and the actual data are different for the new packet.

When a host discards a packet for an isochronous OUT endpoint because the presentation time has expired, it must not reuse the burst information associated with the discarded packet. The host communicates the discard to the isochronous OUT function endpoint using an Isochronous Packet Discard IE. The details of the use of the Isochronous Packet Discard IE are described in Section 4.7.5.

4.11.9.3 Application Handling of Discards

As Figure 4-43 shows, if the channel conditions improve the stream may recover from a series of errors with only a minor loss of data. If the channel stays poor, large amounts of data may be lost over several intervals. It is an application specific decision as to when a stream should be terminated if errors persist. However, there are options that can be utilized by the host or endpoint to attempt to prevent discards from occurring. These options are described in Section 4.10.

4.12 Device Reset

Wired USB uses specific electrical signaling on the D+ and D- lines to signal a USB reset to the device. Upon receipt of reset signaling, the device enters the USB **Default** state and sets its USB address to the Default Address (0).

Wireless USB does not have the option of using specific electrical signaling to signal a reset to the device. Instead, the action must be initiated by sending a ResetDevice_IE which targets a particular device (see Section 7.5.11). This IE is targeted to a particular device via that device's CDID value. The device that decodes the ResetDev_ID that matches the CDID must perform an effective 'hard' or 'power-on' reset and return to the **UnConnected** device state (see Section 7.1 for definition of all device states). The intent of this form is to provide the host some mechanism to quickly reset a device when there may be some ambiguity about what state the device is in or what device address it is at.

Wireless USB also provides for a lighter form of device reset, via SetAddress (zero). For Wired USB devices, device response to Set Address with a value of 0 is undefined. For Wireless USB devices, a device receiving a Set Address with a value of 0 resets its address to the Default Address and enters the **Default** state. Any existing endpoint state is lost. Connection state is not reset. The intent of this form is to reset the function on the device without completely removing the device from the cluster.

4.13 Connection Process

USB 2.0 association is based on a device-initiated model, e.g. the device detects power on its upstream port and signals a Connect by asserting D+ (or D-), which is detected by the associated port and eventually responded to by the host with a Port Reset and subsequent information exchange over the Default Endpoint. Wireless USB implements a similar device-initiated connection model, i.e. the device finds and initiates a connection with its host. The general model of the connection process and roles each device plays are described in the remainder of this section. Details of the mechanisms used to implement this process are defined in the Protocol (Chapter 5) and Framework (Chapter 7).

Hosts include a Host Information IE in selected MMCs. The host also (via the MMCs) makes available DNTS opportunities which may be used by devices to transmit connect requests to the host.

The host reserves MAC Layer channel time across the super frame for the Wireless USB Channel, and will then randomly move the location of the DNTS window(s) for connect opportunities within the Wireless USB Channel, so that they overlay different MAS slots over time, see Figure 4-44.

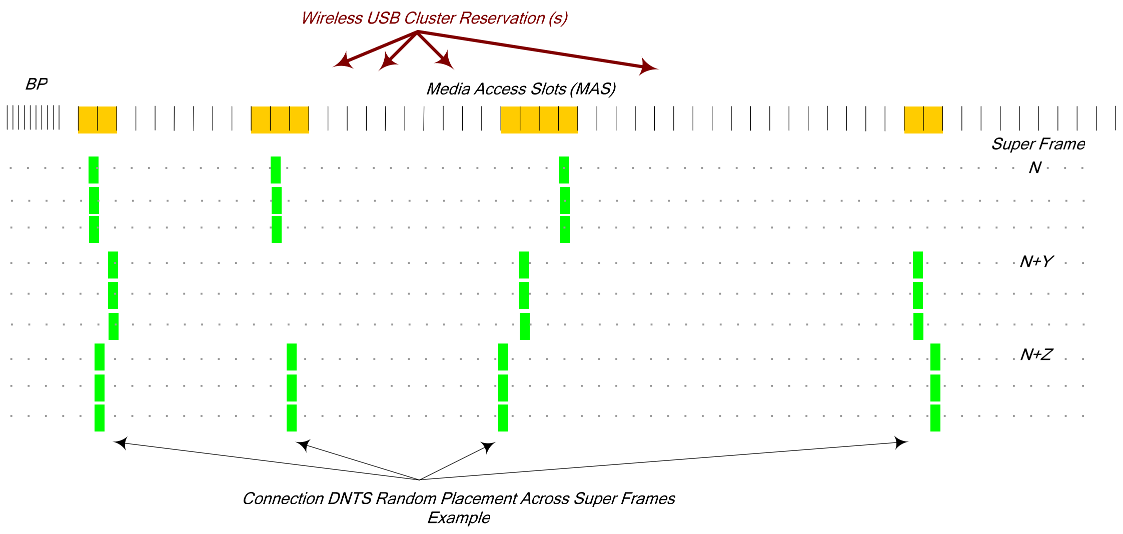
Wireless USB Cluster Reservation (s)
BP
Media Access Slots (MAS)
Super Frame
N
N+Y
N+Z
Connection DNTS Random Placement Across Super Frames Example

Figure 4-44. Example Connection 'Random Hop' DNTS Placement over Time

The security framework (see Section 6.2.8) requires that devices retain information about the host it was previously connected to (Connection Context). The Connection Context may be initialized for 'new' devices via out-of-band 'provisioning' methods which establish on a device, information about its intended host including the host's name (Connection Host ID - CHID) and a secret shared with for that host. A 'new' device may also support being provisioned via the Default Control Pipe. This is referred to as in-band provisioning. For most scenarios, whether by out-of-band provisioning or simply residual information from a previous connection, the host's identity is usually known before the device attempts to connect. See below for details on the connection process when a CHID is not known by the device prior to an association attempt.

An unconnected "provisioned" device that is looking to establish a connection with a known host locates a MAC Layer channel which encapsulates the Wireless USB Channel being maintained by its intended host. The device locates the correct Wireless USB Channel by capturing and processing MMCs in each observable Wireless USB channel looking for an MMC that contains a Host Information IE with a CHID value that matches the CHID in the device's local host context. When a match is found it then follows the MMC control stream in the Wireless USB Channel looking for a DNTS opportunity during which it will attempt to transmit a *DN_Connect* notification. A device making a connection always uses the *UnConnected_Device_Address* when transmitting a *DN_Connect* notification to the host (see Section 7.6.1). The payload of this request includes (at least) the device's name (Connect Device ID, e.g. CDID) which is 16 bytes of device unique information. The device then waits for a *Connect Acknowledge* response (see Section 7.5.1) which includes the device's name (i.e. CDID) from its *DN_Connect* notification and a Wireless USB channel device address. The device must retransmit the connect notification until it successfully receives a connect acknowledge from the host (see below). Retransmissions are implementation-specific, but should occur no more frequently than three times per 100 milliseconds. A device attempting to connect should check all Host Information IEs it encounters to ensure that the host remains open for connection requests.

When a host receives a connect notification, it allocates a Device Address from the *UnAuthenticated_Device_Address_Range* and then includes a Connect Acknowledge IE in a subsequent MMC. The host must retransmit the connect acknowledgement until it observes the device responding to control transfers to its Default Control Pipe at the assigned Wireless USB channel address. The rate of retransmission is host-implementation specific.

On reception of the connect acknowledgement, the device updates its Wireless USB channel device address and then begins listening on the Wireless USB Channel for host transactions directed to its Default Control Pipe at the new device address. If the device is a Self-beaconing device (as identified in the *DN_Connect* notification), the host can quickly determine the devices' optimal MAC Layer channel time to accomplish data communications via its MAS Availability information. The device then proceeds through the Authentication process which is driven by the host using control transfer requests over the device's Default Control Pipe.

Once the host and device have completed mutual authentication, each has the proper session key for encrypting/decrypting protocol data and handshake packets. The host completes the connection process with two final steps. First, it uses SetKey() to give the device a copy of the Wireless USB cluster group key. The device requires this key in order to authenticate MMCs packets. At this point, the device and host have a secure relationship established. The final step in the connection process is the host uses SetAddress() to set the device's device address to zero, which transitions the device into the USB 2.0 equivalent of the **Default** device state.

A device that has no valid Connection Context but is capable of being provisioned via the Default Control Pipe may request a connection with a host provided the host indicates that it is accepting connections from unknown devices. This advertisement is further described in Section 7.5.2. When the device locates a host making such an advertisement, the device makes an attempt to connect as described above, indicating the connect request is for a *New connection*. Such a device must also create a temporary CDID value to use in the connect request. This is covered in more detail in Section 6.2.10.3.

The connection process proceeds as described above and the device transitions to the **UnAuthenticated** device state. While in this state, the host will determine the provisioning capabilities of the device and attempt to provision the device with a valid Connection Context. This provisioning involves establishing a common security mechanism, e.g. public key cryptography to protect the Connection Context. Once the Connection Context has been delivered to the device, the host will initiate the mutual authentication 4-way handshake. When the 4-way handshake is successfully completed, the host and device complete the establishment of the connection as described above.

4.13.1 Reconnection Process

A device may lose contact with its host (i.e. not receive any valid host packets) for a period of time longer than the *TrustTimeout* (see Section 6.2.10.2). Whenever a *TrustTimeout* occurs, the device begins listening for its host (as it does from the **UnConnected** state). When it reacquires its host's Wireless USB channel, the device makes a *reconnect* request to the host. A *reconnect* request is simply an encrypted *DN_Connect* device notification with the *Previous Address* field set to the device's current USB address. This notification is simply asking the host to be allowed to resume operation at that address. If the host does not remember the device (cannot decrypt the notification), it will not respond to the connect notification and the device should then try to re-connect from the **UnConnected** device state (i.e. unencrypted *DN_Connect*). The host may acknowledge the *DN_Connect* with either the device's previous address or a device address in the **UnAuthenticated** device address range. The host will then initiate the authentication process. If this completes successfully, the device returns to its previous operational state (i.e. before the *TrustTimeout*). Note the host may need to restore the device's previous address via a SetAddress(previous address) command before the device is restored to its previous operational state. Note, Section 7.1 provides a detailed device state diagram and summary description of all the device states.

Note that a host has the option of not being available for connections as represented in the Host Information IE (see Section 7.5.2). When a host is reporting that it is not available for connect requests, it must remain available for reconnect requests, as defined above.

4.14 Disconnect

Wireless USB supports two disconnect models: explicit and implicit, however neither is equivalent to the USB 2.0 disconnect model. The explicit disconnect model allows the host or device to initiate a disconnect event. When the host initiates a disconnect event, it is essentially removing the device from its Wireless USB cluster (i.e. tears down the secure relationship, frees up internal resources for tracking the device, etc). When the device initiates a disconnect event, it is simply notifying the host that it is leaving the cluster so that the host can explicitly 'forget' the secure relationship, etc. Disconnected devices may attempt to reconnect at any time. A host initiated disconnect event has the following process:

- The host sends three consecutive WDEV_Disconnect_IEs or WHOST_Disconnect_IEs (to disconnect all devices in cluster).
- The device disconnects immediately and enters the Un-Connected state.

A device initiated disconnect event has the following process:

- A device sends a *DN_Disconnect* notification during a DNTS period to notify the host (see Section 7.6.2). This notification tells the host that the device is going to disconnect. The device should wait for a response from the host before disconnecting and should retry at least twice before disconnecting if no host response is observed.
- The host responds to the device *DN_Disconnect* notification in a subsequent MMC with a WDEV_Disconnect_IE, targeting the requesting device. The device disconnects immediately and enters the Un-Connected state when it sees the host response.

In wired USB, a disconnect event has significant impact on both the host and device state. The host releases buffers and the device address because it knows the device is gone and relies on the wired connect and device enumeration events to occur to return the device to operation. There are many scenarios for the wireless environment (interference, distance change, security, etc.) where it is desirable to be flexible about when to detect an actual disconnect event and trigger release of resources and force a full re-enumeration and initialization. The implicit disconnect model embodies this flexibility.

The basic model of the implicit disconnect is tied to the *TrustTimeout* threshold. In general, Wireless USB requires the host and device to keep its notion of *TrustTimeout* intact. This leads to a consistent user experience because when they attempt to use an idle device, chances are good that the device is available (or will be available in a reasonable amount of time). A host may implement a policy where a device is 'disconnected' whenever the host observes a *TrustTimeout* for that device.

The mechanisms defined to accommodate refresh of the Trust relationship are different, depending on the operational state or communications load on the device (device scenario). The mechanisms are described below, based on the individual device scenarios. Note, the host and device views are described as appropriate.

- **Active.** In this scenario, the host is actively communicating with the device. When data is flowing, both the host and device are seeing packet transmissions from each other within the *TrustTimeout* threshold.
- **Idle.** In this scenario, the host is not actively communicating with the device (i.e. the owning device driver is not actively attempting to move data to/from the device, or all active function endpoints are flow controlled). The device however, won't experience a *TrustTimeout* as long as it can successfully track the Wireless USB channel (i.e. MMCs). However, since the host is not actively scheduling transactions to any function endpoints, the host observes no encrypted packets from the device. In this situation, the host will activate a 'keep alive' poll by use of the Keepalive IE/*DN_Alive* mechanism (see Sections 7.5.9 and 7.6.7). This mechanism allows the host to specifically request one or more devices to send a *DN_Alive* or equivalent notification to the host. The successful reception of a notification from the device will restart the *TrustTimeout* period for the associated device. Note that once the host includes a device's address in a Keepalive IE, it will remain there until it either successfully receives a notification packet from the device, or it encounters a *TrustTimeout*. Similarly, a device will continue to transmit *DN_Alive* or equivalent notifications until the host removes the device address from subsequent Keepalive IEs or it experiences a *TrustTimeout*.
- **Sleep.** This scenario is described in detail in Section 4.16.1.1.

Loss of Activity is the scenario that all other scenarios degrade to when the host and device are unable to successfully receive each other's packets for at least a *TrustTimeout* period. When an active device loses an MMC, it should go into an 'open scan' mode (continuous listening) looking for an MMC transmission from its host. Looking for an MMC from a host in this context can be as simple as matching on a transmission addressed to the cluster, which includes the correct Cluster ID and Stream Index value in the MAC header, in addition to the correct frame type, etc. If the device fails to observe host activity (MMCs) for 500 milliseconds it should initiate its host detection process.[4] If the device re-acquires the Wireless USB channel before a *TrustTimeout* occurs, then the device may continue normal operations. After a *TrustTimeout* event, the device must transition to the Reconnecting device state and continue looking for the Wireless USB channel of its host. Looking for a host in this context includes finding an MMC with a Host Information_IE that has the correct CHID value. On subsequent host Wireless USB channel acquisition, the device will attempt to re-connect for at least 100 ms. If

[4] A host detection process can involve scanning other PHY channels for the host, perform a continuous open scan, etc.

the host does not respond to the re-connect attempts, the device must then transition to the **UnConnected** state and can then begin connect attempts.

When the device knows that the host is asleep and has enabled the device to perform Remote Wakeup, the device should wait at least a *TrustTimeout* (the host required ‘polling’ rate) before assuming the host has disappeared.

When a **Sleep**ing device returns to the **Awake** state, it begins looking for MMCs from its host on the same PHY channel it last saw MMCs from its host. It searches for MMCs via an open scan. Similar to above, if it cannot locate the correct MMC for 500 milliseconds, it should begin looking for its host on other PHY channels. See Section 4.16.2.2 for details on behavioral requirements for Remote Wake.

4.15 Security Mechanisms

The security mechanisms described in this specification are implemented using the security mechanisms of the MAC Layer. This section describes the mapping between Wireless USB security concepts and MAC Layer security concepts see reference [3]. Refer to Section 6 for complete security details.

Wireless USB hosts and devices operate in MAC Layer Security Mode 1. This mode allows Wireless USB devices to connect using Wireless USB Control requests encapsulated in MAC Layer data frames.

A device receives a group key from the host at the completion of a successful 4-way handshake. However, a device must be able to receive MMCs from the host in order to locate the host and start the 4-way handshake. A device is permitted to successfully receive secured MMCs if it is not yet in possession of the valid group key. When the device receives a group key from the host, it should begin validation of the MMCs as described for the MAC Layer.

4.15.1 Connection Lifetime

Wireless USB requires that data communications must occur frequently enough to keep the trust relationship intact. If a host does not receive any authenticated packets from a device or a device does not receive any authenticated packets from its host for a *TrustTimeout* period, the host (or device) must force a re-authentication (i.e. 4-way handshake) before resumption of normal data communications. The duration of *TrustTimeout* is four (4) seconds.

4.15.2 Host Security Considerations

4.15.2.1 CHID Selection

Devices use the CHID field of a connection context to locate a host. To insure uniqueness in the presence of multiple hosts, a host should develop its CHID value from other values that supply uniqueness, such as the host’s EUI-64 address.

4.15.2.2 CDID Selection

CDID values should be derived using the PseudoRandom Function PRF-128. This is described in Section 6.2.10.1

4.16 Wireless USB Power Management

Wireless USB provides mechanisms that allow hosts and devices to opportunistically and explicitly control their power consumption. Because Wireless USB protocol is TDMA-based, hosts and devices know exactly when their radios do not need to be transmitting or receiving and can take steps to conserve power during these times. Other mechanisms allow hosts and devices to turn off their radios for longer periods of time. The sections below cover power management mechanisms available for devices and for hosts and define the interactions between the two.

4.16.1 Device Power Management

Devices have three general ways to manage their Wireless USB power consumption. The first is to manage power during normal operation by taking advantage of the TDMA nature of Wireless USB protocol and opportunistically turning their radio off during periods when it isn't needed. Devices can do this at any time with the host being unaware of the efforts.

The second way to manage power is to have the device go to 'sleep' for extended periods of time but still stay 'connected'. In this case the device will not be responsive to any communications from the host. Devices must notify the host before sleeping. Details of the mechanisms for notifying the host are provided in Section 4.16.1.1.

The third way that a device can save power is to disconnect from the host. The mechanism for device disconnect is covered in Section 4.14.

4.16.1.1 Device Sleep

During periods of inactivity, a device may want to conserve power by turning off its radio and being unresponsive for an extended period of time. A device is required to notify the host before going to sleep and the host will acknowledge the notification.

A device uses the *DN_Sleep* notification sent during a DNTS period to notify the host of its intent to transition to the **Sleep** state. The format of the *DN_Sleep* notification can be found in Section 7.6.5. There are two types of Sleep notifications described by the *DN_Sleep* notification:

- Device is going to sleep. This notification tells the host that the device is going to sleep unconditionally. The device should wait for a response from the host (see below) before going to sleep and must retry at least twice if no host response is seen. After three attempts, a device may choose to go to sleep without seeing a host response after 3 MMCs have occurred after the last attempt.
- Device wants to go to sleep. This notification tells the host that the device is going to sleep if there is no pending work for the device. The device must wait for a response from the host and depending on the host response decide whether or not to go to sleep. If no response is seen, the device may retry or decide to stay awake.

In response to a *DN_Sleep* notification a host generates a Work IE acknowledgement (see Section 7.5.7) and includes that IE in three successive MMCs. Work IEs contain information indicating whether or not there is work pending for the device. If there are no operations queued for a device or if the only operations are on Interrupt IN endpoints or flow controlled (ie. inactive) IN endpoints, then the host response will indicate No Work pending. For any other situation the host response will indicate Work pending.

Table 4-6 shows the host view of the device power management state based on the different device notifications and host responses.

Table 4-6. Standard Request Availability in Wireless USB Device States

Sleep Notification	Host Response	Device State (host view)
Going to Sleep	Work	Sleep
Going to Sleep	No Work	Sleep
Want to Sleep	Work	Awake
Want to Sleep	No Work	Sleep

When the host believes a device is in the **Sleep** state, the host will not schedule any transactions with the device.

There may be cases where the host believes a device is in a different power management state than the device is actually in. For example, if the host does not see any of the Sleep notifications (maybe because of interference) and the device decides to go to **Sleep** anyway, the host will think the device is Awake, when actually it is sleeping. In this case, the host may schedule transactions for the device that will time out and the device may end up being disconnected. This is a risk a device takes if it decides to go to **Sleep** before seeing a response from the host.

Another mismatch between states could occur if the host sees the Sleep notification but the device does not see the host response. In this case, the host thinks the device is in Sleep state, while the device is still **Awake**. This state will typically resolve because the device will continue to send the Sleep notification until it sees a host response.

A device must not attempt to transition to the **Sleep** state while processing a control transfer (i.e. have not responded with an ACK to the Status stage of the control transfer). It may attempt to transition to the **Sleep** state (beginning with a DN_Sleep notification) after it has responded to the Status stage of a control transfer with an ACK (or STALL, signaling its completion of the control transfer). The device must not transition to the **Sleep** state if the host responds with either a Work_IE (Work) or another transaction (or transaction phase) addressed to the device.

4.16.1.2 Device Wakeup

After entering the Sleep state, devices may want to occasionally check with the host to find out if there is any work pending or the device may want to go back to the Awake state because the device now has data to deliver to the host (maybe for an Interrupt IN endpoint).

To check for pending work, the device sends a Sleep notification as described above and the host makes the same responses as described above. The state of the device again corresponds to Table 4-6 above. Devices must not check for pending work any more often than every 100 milliseconds. There is no maximum time limit specified for how often a device must 'check in' with the host, although devices that don't 'check in' at least once every *TrustTimeout* are likely to be disconnected. See Section 4.14 for a description of disconnect mechanisms and timings.

When a device wants to transition from the **Sleep** state to the **Awake** state the device notification transmitted by the device depends on whether the device has detected a *TrustTimeout*. If there has been a *TrustTimeout*, a device must transmit a Reconnect Request notification (see Section 7.6.1.2) to the host. The host will respond with a Connect Acknowledge IE, which returns the device to the **UnAuthenticated** state (see Section 7.5.1). After re-authentication the host will begin scheduling transactions for the device. If there has not been a *TrustTimeout*, the device will transmit DN_Alive notifications to the host. On successful reception of the DN_Alive before a *TrustTimeout*, the host will start scheduling pending transactions, if any, for the device. The host may perform a 4-way handshake at any time.

Anytime a device goes to sleep it runs the risk of the host disappearing or being disconnected from the host. Host disappearance is detected when the device cannot find the Wireless USB channel (i.e. MMCs). In this case, the device should revert to its standard procedure for finding a host. If the device can find the WUSB channel, but the host never responds to the Sleep notifications, the host may have 'disconnected' the device and the device may need to reconnect using a Reconnect Request notification.

Figure 4-45 shows a state diagram for device power states. This diagram depicts the state transitions that a device makes assuming that the device waits for a host response before making a state transition.

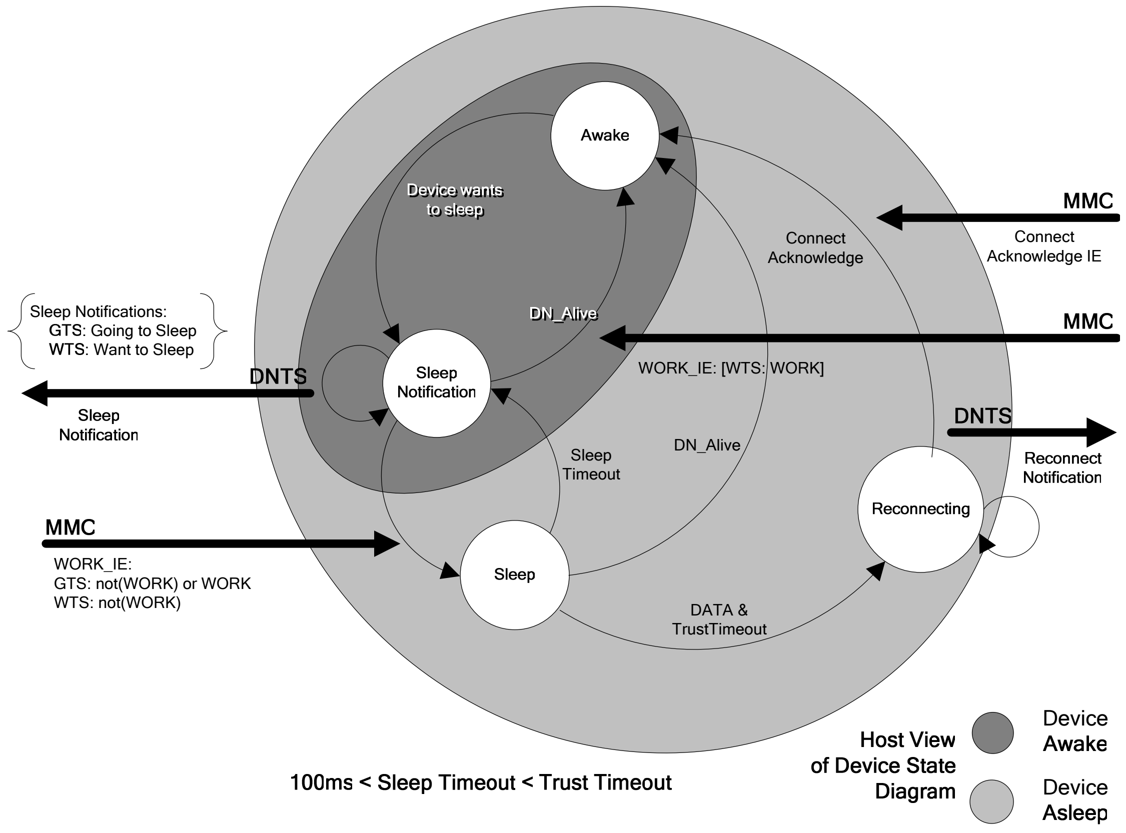

Figure 4-45. Power state diagram for devices

4.16.2 Host Power Management

A host has two general ways to manage Wireless USB power. The first can be done during normal operation by taking advantage of the TDMA nature of WUSB protocol and turning the radio off during periods when it is not needed. During times of low activity, the host can manage the Wireless USB channel to have long periods between MMCs and thereby have more time when the radio can be off. Devices are unaware of this power management, and since the Wireless USB channel is maintained, they just follow from one MMC to the next.

The second general way for a host to manage power is to interrupt the Wireless USB channel, meaning that the continuous string of linked MMCs is stopped. Some typical reasons for the host to do this include:

- The platform going to a low power state (Standby, Hibernate, …)
- The platform being shut down.
- The user disabling the radio
- Aggressive host power management

For this case, devices are made aware of the hosts actions through an explicit communication from the host. This is described in Section 4.16.2.1. A Remote Wake mechanism is defined in Section 4.16.2.2. This allows a sleeping host to be awakened by a Wireless USB device. Section 4.16.2.3 describes host and device behaviors as the host ‘wakes up’ from either being asleep or off.

4.16.2.1 Channel Stop

When a host is going to stop the Wireless USB channel, it must tell devices ahead of time. The host does this by including a Channel Stop IE in at least three consecutive MMCs immediately before the channel is stopped. See Section 7.5.8 for details on the Channel Stop IE. Information in the IE includes the Wireless USB channel time when the channel will stop. That time should match the end of the last MMC transmitted. In the last MMC before the channel is stopped, the Next MMC Time field is set to zero, and there should be no timeslots allocated.

If a host decides not to stop the channel after including Channel Stop IEs in MMCs, the host simply removes the Channel Stop IEs from subsequent MMCs. After stopping a channel, the host can restart the channel at any time.

Awake devices should not stop tracking the Wireless USB channel until after the channel stop time has been reached. Anytime a device receives a MMC without a Channel Stop IE, it must keep tracking the Wireless USB channel.

When a host stops the Wireless USB channel, it assumes that all devices have gone to the Sleep state.

When/if the host restarts the channel, devices may 'reconnect' with the host using the mechanisms described in Section 4.16.1.2 that describes what devices can do when waking up.

4.16.2.2 Remote Wakeup

Wireless USB has a Remote Wakeup mechanism that allows a Wireless USB device to wake up a sleeping host. A host that is checking for Remote Wakeup must restart the USB channel at least once every *TrustTimeout*. The restarted channel must contain at least three MMCs, all including DNTS slots. These MMCs will also typically include Channel Stop IEs to indicate that the host is stopping the channel again.

If a host will be checking for Remote Wakeup, it must set the Remote Wakeup bit in the Channel Stop IEs that it transmits prior to stopping the channel.

When a device wants to wake up a host, the device tries to find the Wireless USB channel by listening for MMCs. Ideally, the device will discover the channel within the *TrustTimeout* (the host required 'polling' rate). Devices may search for a longer time, but at some point will probably decide that the host has disappeared and will follow device specific mechanisms for finding the host.

If the device finds the Wireless USB channel and the MMCs with DNTS slots, the device will send a Remote Wakeup notification to the host. Remote Wakeup notifications are described in Section 7.6.6 of the Framework chapter. If the host successfully sees the notification, the host will remove the Channel Stop IEs from the MMCs and the channel will continue operation. The device must send a Reconnect Request notification after the channel is operating.

Note that there are other notifications that a device may transmit that may result in waking up the host. This includes analogs to the wired events of connect, disconnect, etc. The summary list of notifications available to a device from the **Sleep** state is in Section 7.6.

4.16.2.3 Channel Start

To start or restart a Wireless USB channel, a host simply begins to transmit MMCs with DNTS IEs. Note that hosts will do the appropriate MAC Layer PHY channel selection and DRP protocol before starting the Wireless USB channel. Hosts should always try to use the same PHY channel used when previously operating, if at all possible. When a USB channel restarts, the host may decide to retain some connection context from the last time the channel was running. For example, when a suspended host resumes and restarts the channel, the host may remember devices that were previously connected and not have to fully re-enumerate those devices when they reconnect. However, the host will always re-authenticate devices when restarting a channel. Devices that reconnect within a *TrustTimeout* of a host restarting a channel are assured that they will not have to be re-enumerated.

When a device detects a restarted Wireless USB channel, wants to connect, and has retained its context (configuration state, etc.) since the channel was stopped, the device sends a Reconnect Request notification to the host. If the device has not retained context, then it sends a Connect Request notification to the host. In either case, the host will respond with a Connect Acknowledge IE in a subsequent MMC and proceed with transactions to the device.

4.17 Dual Role Devices (DRD)

Wireless USB allows a device to simultaneously function as both a host and Wireless USB Device[s] on a single transceiver. Any numbers of scenarios are possible including ‘combination’ and ‘point-to-point’ scenarios. A combination scenario is one where a device is connected ‘upstream’ to one or more separate Wireless USB Channels as a device and at the same time manages its own Wireless USB Channel as a host. An example of a combination scenario is a Wireless USB printer which could be connected as a device on one or more Wireless USB Channels and at the same time provides a Wireless USB channel for connecting cameras. A point-to-point scenario is one where two Wireless USB DRD devices connect as both a host and Wireless USB device to each other.

In the “combination” scenario, Wireless USB DRD-Host and Wireless USB DRD-Device[s] are logically independent. They just operate as a Wireless USB host and Wireless USB device[s] respectively.

In the “point-to-point” scenario, two Wireless USB DRDs linking to each other by one upstream and one downstream Wireless USB link, are called paired P2P-DRDs. The Wireless USB link that is established first is called the default link; the link that is established later is called the reverse link. Paired P2P-DRDs must co-exist on the same MAC Layer channel. The pairing process between two P2P-DRDs is a two-stage establishment of the default link and the reverse link.

Both default link and reverse link share the same connection context [CHID, CDID and CK] and session context [SFC, data keys and management keys]. Therefore, P2P-DRD’s CHID on the default link is the same as P2P-DRD’s CDID on the reverse link. Only the DRD-Host of the default link can modify these contexts.

A two stage establishment process is described in the following sub-sections:

- Discover P2P-DRD Host to establish default link
- Pair P2P-DRD to establish reverse link

4.17.1 Discover P2P-DRD Host to establish default link

There are two kinds of P2P-DRD discovery processes:

- User-instructed discovery process for the first time connection
- Self-discovery process for re-connections.

During first time connection, the P2P-DRD that is powered-up first will be the DRD-Host. The second P2P-DRD will scan the medium and finds its corresponding P2P-DRD Host. It will connect to this P2P-DRD Host as a P2P-DRD Device.

A P2P-DRD capable host shall set the P2P-DRD capable bit to one in the Host Information Element in its MMCs. This allows a P2P-DRD capable device to scan its proximity and locate a P2P-DRD Host.

After both P2P-DRDs complete the connection process, they have their connection context and the default link. Then the host enumerates the device. The P2P-DRD device shall return a Device Capability Descriptor with P2P-DRD capability set to 1. Once the DRD-Host finds the device is a P2P-DRD device, it goes to establish a reverse link as specified in Section 4.17.2.

For the re-connection case, paired P2P-DRDs in the previous session keep their role as before, and re-establish the default link through the re-connection process. User intervention may be necessary if the P2P-DRDs cannot reconnect to each other within an acceptable period of time.

4.17.2 Pairing P2P-DRD to establish reverse link

After the default link succeeds in the authentication phase of the connection process, the P2P-DRD host on the reverse link shall allocate its beacon slot and wireless USB cluster on the same MAC channel, set P2P-DRD capable bit to one in the Host Information IE of its MMCs and open to re-connection.

After the P2P-DRD host on the default link finds the wireless USB device is a P2P-DRD device, as a device on the reverse link, it starts to locate the host on the reverse link whose CHID value is the same as its CDID value on the default link.

P2P-DRDs on the reverse link use the same connection context on the default link to re-connect each other; and skip the authentication phase of the connection process. They authenticate each other by using secure USB control transfer – SetAddress(0).

During enumeration phase, the P2P-DRD device on the reverse link shall return a Device Capability Descriptor with P2P-DRD capability clear to 0.

In order to self-establish the default link in the future, both successfully paired P2P-DRDs shall remember their connection context and respective roles on the default link.

Chapter 5
Protocol Layer

This chapter presents a bottom-up view of the Wireless USB protocol starting at packet format definitions inherited from the MAC Layer standard and the application-defined extension required for Wireless USB. This is followed by a detailed description of Transaction Groups and basic transaction formats followed by detailed Transaction Group timing requirements. The next section provides a detailed description of data bursting, transaction level fault recovery, link-layer data flow for each transfer type and a summary of the flow-control protocol. The last section contains PHY and MAC-specific timing and header information.

5.1 Packet Formats

Wireless USB uses the packet (Frame) formats defined in the MAC Layer standard. The general structure of a packet is that it contains a PHY Preamble, PHY Header and MAC Header followed by a data payload (MAC frame body) which can be transmitted at a signaling rate different than that of the PHY and MAC Header (see top of Figure 5-1). The PHY layer provides standard support for error correction for all bits in the logical packet (PHY/MAC Header plus frame body). The PHY also CRC checks the PHY and MAC Header. The Frame Check Sequence field, which is the CRC value for the frame body payload is managed by the MAC layer. See the MAC Layer standard for implementation requirements. Note that when the Security bit component of the Frame Control field is set to zero (0), the security-related fields are not present in the packet. These fields are *TKID*, *Rsrvd*, *Encryption Offset*, *SFN*, and *MIC*. These fields are present if the Security bit is set to one (1).

Section 5.6 summarizes the MAC Layer packet types (MAC Header field) used in the Wireless USB protocol. Wireless USB uses both the secure and non-secure packet formats defined in the MAC Layer standard.

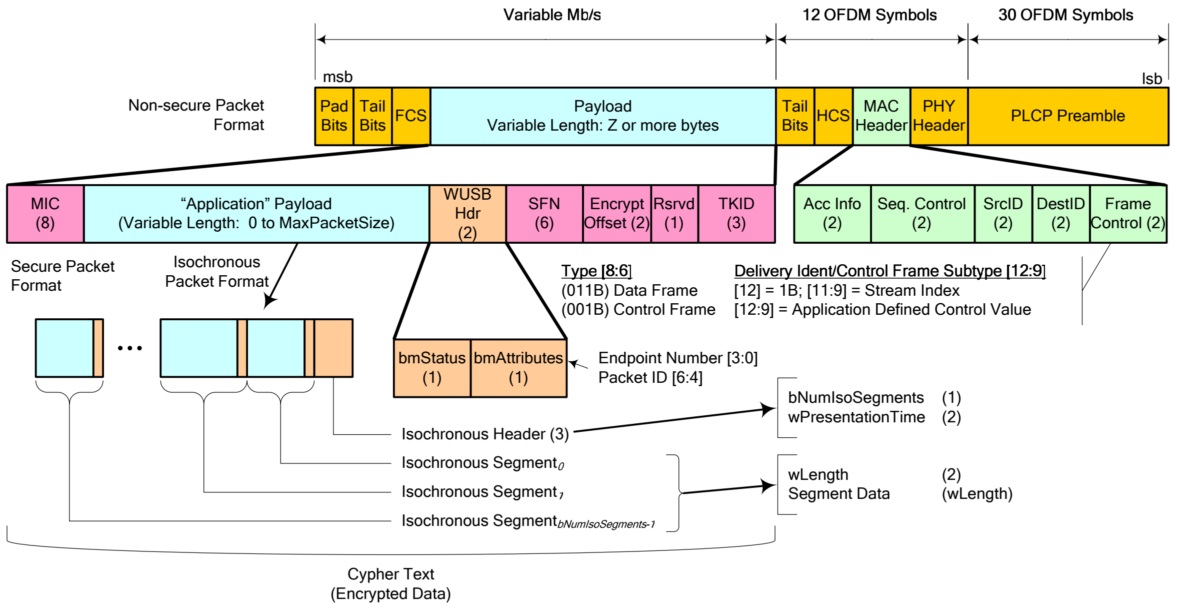

Figure 5-1. General Format of a Wireless USB Application Packet

The terminus of data communication flows on a Wireless USB device is the same as that of a wired USB device; i.e. an Endpoint. Wireless USB addressing has three basic parts:

1. All packet transmissions (using a MAC frame type of *Data Frame*) during a Wireless USB DRP use the same stream index value in the stream index field of the MAC Layer Header. The stream index

value is allocated by the host when establishing a Wireless USB Cluster. Devices learn this value from the Host Information IE.

2. Every device in the Wireless USB Cluster is assigned a unique device address (relative to the cluster) during the enumeration process. The assigned device address is equivalent to the wired USB device address. Note that the host has an effective device address which is the generated DevAddr value necessary to conform to MAC Layer addressing requirements. Every packet transmission includes the transmitter's device address in the MAC Layer MAC Header *SrcAddr* field and the targeted devices' device address in the MAC Layer MAC Header *DestAddr* field. The exception to this is the MMC packet which uses the Broadcast Cluster ID in the *DestAddr* field.

Due to artifacts of the MAC Layer operation, the host may need to change its MAC Layer device address (i.e. the value it uses in the SrcAddr field of any host-to-device or host to Cluster Broadcast Address transmission). When devices transmit packets to the host, they must use the MAC Layer device address from the MMC's SrcAddr field in the device's *DestAddr* field.

3. Wireless USB packets which originate or terminate on a function endpoint must include a Wireless USB Application header (see below for details). The Wireless USB Application Header serves several purposes one of which is to carry *Endpoint Number* addressing information.

In summary, the *DestAddr* field in the MAC Header is used by host and device MACs to determine whether received packets should be ignored. The *Stream Index* and *SrcAddr* fields in the MAC Header are used by the host and device to deliver received packets to the Wireless USB application (one of possibly several distinct applications simultaneously using the radio resource). The device uses the *Endpoint Number* field in the Wireless USB Application Header to deliver the data to the correct endpoint buffer. The host uses *Endpoint Number* and *SrcAddr* field (from the MAC Header) to deliver the data to the appropriate endpoint buffer.

The Wireless USB Application Header is located immediately after the security header fields and is included in all packets originating or terminating at an Endpoint. The format of the Wireless USB Header is detailed in Table 5-1. The length of the Wireless USB Header is different, depending on the value of the PID field. The shaded portion of the table indicates the fixed or common portion of the Header, which must be present, regardless of the value of the PID field. Isochronous data phase data packets use the IDATA PID and have an additional variable length header section following the common Wireless USB header. The additional information for headers with the IDATA PID must contain the fields for at least one data segment as shown in white in Table 5-2. Additional fields for additional data segments are optional and shown in grey in Table 5-2.

Table 5-1. Wireless USB Data Packet Header Format Details (rWUSBHeader)

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	This field has the following bit encodings: **Bits** **Description** 3:0 **Endpoint Number.** Valid values are 0-15. 6:4 **Packet ID (PID)** (see Table 5-3) 7 **Reserved/Endpoint Direction.** Value depends on PID value. If value not explicitly defined for a PID value then the value must be 0B. When Endpoint Direction, the value encodings mean (0: OUT, 1: IN).

Table 5-1. Wireless USB Data Packet Header Format Details (rWUSBHeader) (cont.)

Offset	Field	Size	Value	Description
1	*bmStatus*	1	Bitmap	This is a status byte/information byte. The field encodings are: **<u>Bits</u>** **<u>Description</u>** 4:0 **Sequence Number.** When the PID field indicates a DATA or IDATA packet, then this field is the data burst sequence number. Otherwise, this field must be set to zeros. 7:5 **Flags/Handshake Code.** When the PID field has the value DATA or IDATA this field is a set of status flags related to the data stream. Either the host or device may set these flag bits. Encodings are: **<u>Bits</u>** **<u>Description</u>** 6:5 Reserved. Must be set to zero 7 Last Packet Flag When the PID field has the value HNDSHK, the packet is a Handshake packet. This field is used by a device (only) to return information about its status to the host. The host does not use the HNDSHK PID value. .The bit fields are encoded as follows: **<u>Value</u>** **<u>Meaning</u>** 000B Reserved. 001B ACK. Endpoint observes error-free data or acknowledges command in token. 010B NAK. This value indicates the endpoint is not prepared to move data (transmit or receive). 011B STALL. Endpoint is halted or a control transfer request is not supported. 100B – 111B Reserved. This field is reserved and must be set to zero when the PID field has any other value.
2	*Isochronous Header*	Var	Record	This is a variable-length header used to describe organization of the isochronous data in the payload. This portion of the header only exists when PID equals IDATA.

Refer to Section 4.10.2 for details on the use of the *bmStatus.Flags.LastPacket* field.

The Isochronous packet header, when present is always concatenated to the standard Wireless USB packet header. Therefore, the Offsets in Table 5-2 are relative to the position of the standard Wireless USB packet header.

Table 5-2. Wireless USB Data Packet Header Details for Isochronous Packets

Offset	Field	Size	Value	Description
2	bNumIsoSegments	1	Number	This field is present in all isochronous data packets (PID value = IDATA). This field indicates the number of data segments that are contained in the data payload of the packet. There must be at least one data segment in an isochronous data packet.
3	wPresentationTime	2	Number	The presentation time on the Wireless USB channel associated with Data1. wPresentationTime has a 125 microsecond granularity. The value is application specific, but typically references a microframe time when the data was intended to be delivered to the receiver. Presentation times for subsequent data segments are implied based on the service interval characteristics of the endpoint. Zero length packets must be explicitly described as zero length data segments. Examples of the use of presentation time are provided in the data flow chapter.
5	wLength1	2	Number	The length of the data in data segment 1 (Data1) in bytes.
7	Data1	Var	Raw Data	The data for data segment one.
7+ wLength1	wLength2	2	Number	The length of the data in data segment 2 (Data2) in bytes.
9+ wLength1	Data2	Var	Raw Data	The data for data segment two.
…				
Var	wLengthN	2	Number	The length of the data in data segment N (DataN) in bytes.
Var+2	DataN	Var	Raw Data	The data for data segment N.

Table 5-3. Wireless USB PID Types

PID Type	PID Name	Value	Description
Data	DATA	000B	Data packet. Sequence numbers for packets are located in the *bmStatus* field.
	IDATA	001B	Isochronous data packet. Isochronous data packet headers must use the IDATA value in the PID field which also indicates that the Wireless USB Header includes an additional variable length isochronous packet header field. See Table 5-1.
Handshake	HNDSHK	100B	Device transmitted handshake packet. Status bit values and data payload content are used to communicate explicit handshake information to the host.
Notification	DN	101B	Device Notification
Reserved		010B-011B and 110B-111B	These PID values are reserved for future use

On reception of a data packet the receiver will strip the Security, Wireless USB Headers, the Security Checksum, and Isochronous Headers (when appropriate) before delivering the data to the appropriate application buffer (on the host or device). Note that all checksums and decryption checks must complete successfully before the receiver is allowed to commit any action related to the packet, including generating a handshake, notifying an application that data is available, etc. The application payload bytes illustrated in Figure 5-1 are the Wireless USB equivalent to the data field of a wired USB data packet. For asynchronous data streams, the rules for sending data between a Wireless USB host and device are the same as for the wired case (e.g. a multi-packet data request by a client application is expected to be delivered with data packet payloads of maximum packet size until the last data payload which may be less than a maximum packet size). This allows short-packet semantics to be used for Wireless USB data streams.

There are four basic packet types used to communicate information in the Wireless USB protocol. The use of secure or non-secure packet encapsulation depends on the context of use unless specified otherwise. The MAC Layer may provide a programmable AcK mechanism where each data packet can be marked by the transmitter by a code indicating the AcK policy. The Wireless USB protocol described here exclusively utilizes a MAC Layer no-AcK policy (e.g. no immediate MAC Layer defined AcKs used). Whenever the host is expecting any cluster device to transmit, it must be listening at least two maximum drift times ($2^*\mathbf{t}_{\mathbf{MAXDRIFT}}$) before the Wireless USB channel time it determines when the device should start transmitting, based on its local clock.

- The MMC (Micro-scheduled Management Command) packet (see Section 5.2). MMC packets use the *Application-defined Control Frame* format defined in the MAC Layer standard. MMC packets are a Cluster broadcast control packet, effectively addressing all devices in the Wireless USB Cluster, and therefore do not include a Wireless USB Header (see Table 5-1). MMC packets are the management thread for the Wireless USB Channel, so must be transmitted at the most reliable bit transfer rate (i.e. PHY Base signaling rate). MMC packets are always transmitted using secure packet encapsulation with the Encryption Offset field in the Security Header set to the length of the MMC payload. Note that the host uses the Wireless USB Group Key to generate the MIC for MMC packets.

- A protocol data packet. Protocol data packets are encoded as *Data Frame* in the MAC Header. Protocol data packets can be transmitted by either a host or a device and can only be transmitted during a W_{DR}CTA or W_{DT}CTA time slot (see Section 5.2.1). Protocol data packets must include the Wireless USB Header. In addition they must be transmitted using secure packet encapsulation unless explicitly specified otherwise. When secure packet encapsulation is present, the Encryption Offset field in the Security Header is set to a value of 2, so that the Wireless USB Header is transmitted in plain text. The entire body of the application payload is encrypted. Protocol data packets have the Wireless USB Header *bmAttributes.PID* field set to DATA or IDATA and the *bmAttributes.EndpointDirection* field must be set to 0B and should be ignored by the Receiver (see Table 5-3). The remainder of the payload portion of a protocol data packet is application-specific data. Protocol data packets can be transmitted at any of the implementation supported bit transfer rates.

- A protocol handshake packet. Protocol handshake packets are encoded as *Data Frame* in the MAC Header. Protocol handshake packets can only be transmitted by a device and only during a W_{DT}CTA time slot (see Section 5.2.1). Protocol handshake packets must include the Wireless USB Header with the *bmAttributes.PID* field set to HNDSHK and the *bmAttributes.EndpointDirection* field set to the direction of the endpoint generating the handshake packet. In addition they must be transmitted using secure packet encapsulation unless explicitly specified otherwise. When secure packet encapsulation is present, the Encryption Offset field in the Security Header is set to six (6). The entire handshake packet is transmitted in plain text but is still protected by the MIC. Protocol handshake packets are small, however important portion of the protocol, and must be transmitted as reliably as possible, so therefore must transmitted at the most reliable bit transfer rate (i.e. PHY Base signaling rate).

Table 5-4. Handshake Packet Format

Offset	Field	Size	Value	Description
0	*rWUSBHeader*	2	Record	See Table 5-1. *PID* value = HNDSHK and the *bmStatus* bits indicate the type of handshake information.
2	*bvAckCode*	4	Bitmap	When the handshake packet is an acknowledgement of a data phase data burst, this field is used to convey information about the results of the last data burst phase to the host. See Section 5.4 For the information that is required to be encoded in this field.

- A device notification packet. Device notification packets are encoded as *Data Frame* in the MAC Header. Device notification packets can only be transmitted by a device, during a W_{DNTS}CTA time slot (see Sections 5.2.1 and 5.3). Device notification packets are transmitted using secure packet encapsulation unless explicitly specified otherwise. Device notification packets must include the Wireless USB Header with the *bmAttributes.PID* field set to DN and the *bmAttributes.EndpointDirection* field must be set to 0B by the device and may be ignored by the host. Note that some device notifications are transmitted without secure packet encapsulation because they are transmitted outside of the secure relationship (i.e. like the Connect notification). Refer to Section 5.3 for details about device notifications. The data payload portion of the packet is used to convey specific notification information from the device to the host. When secure packet encapsulation is present, the Encryption Offset field in the Security Header is set to the length of the Wireless USB header plus the length of the notification payload. In short, the entire packet is transmitted in plain text in form similar to an MMC (see above).

The host may determine that it does not have any pending transactions for a reservation period and can 'release' the remainder of the period by directing all devices in the cluster for which the reservation applies to transmit a UDR control packet which allows non-Wireless USB device in range to utilize the remainder of the reservation period.

5.2 Wireless USB Transaction Groups

This section provides a general overview of how the USB transactions are accomplished via Micro-scheduling, defines general structure for the MMC (Micro-scheduled Management Command) and also defines the valid information elements for an MMC.

A Wireless USB Micro-scheduled sequence is comprised of an MMC (transmitted by the host) and the subsequent channel time which is described in the MMC. Wireless USB uses the structure of a Micro-scheduled sequence to manage the Wireless USB transaction protocol. In general, a Micro-scheduled sequence may include one or more Wireless USB transactions and is generally referred to in the remainder of this specification as a ***Transaction Group***. Figure 5-2 illustrates the general format of a transaction group. Note that a transaction group is simply a structure for running Wireless USB transactions. The host dynamically manages the contents (size) of transaction groups over time depending on the demands of the Endpoint data streams. Therefore, the number of transactions per-transaction group can be dynamic.

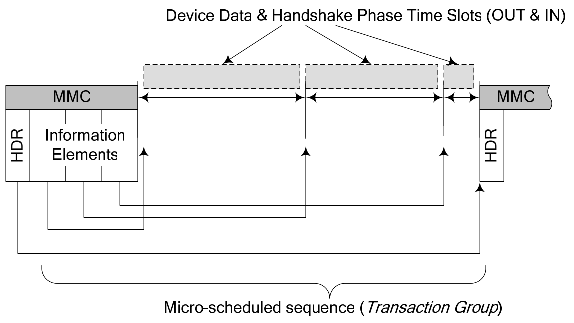
Device Data & Handshake Phase Time Slots (OUT & IN)
MMC
HDR
Information Elements
MMC
HDR
Micro-scheduled sequence (Transaction Group)

Figure 5-2. General Model of a Wireless USB Transaction Group

MMCs are used by a host to maintain and control the Wireless USB Channel. The MMC is an Application-defined Control packet (see MAC Layer specification) and is comprised mostly of specific information elements (IEs). The MMC layout and many of the MMC information elements are defined in the Framework Chapter, see Section 7.5. The Channel Time Slot Allocation IEs are not in the Framework chapter, but are included here as they are a fundamental component in describing the Wireless USB data streaming Protocol.

5.2.1 Wireless USB Channel Time Allocation Information Elements

The general form of a Wireless USB information element that describes time slot allocations is formatted as illustrated in Figure 5-3. A host may include at most one WCTA_IE in an MMC.

(lsb)					(msb)
1	1	variable	variable		Variable
bLength	IE Identifier = WCTA_IE	W_XCTA[0]	W_XCTA[1]	…	W_XCTA[n]

Figure 5-3. General Form of a Wireless USB Channel Time Allocation IE

The *bLength* field value includes the total length of the Wireless USB Channel Time Allocation information element, including the *bLength* field. A Wireless USB Channel Time Allocation IE is comprised of one or more W_XCTA blocks (Wireless USB Channel Time Allocation blocks). W_XCTA blocks describe a time slot allocation relative to the MMC. The general structure of a W_XCTA and the relationship between its information and the described time slot is illustrated in Figure 5-4.

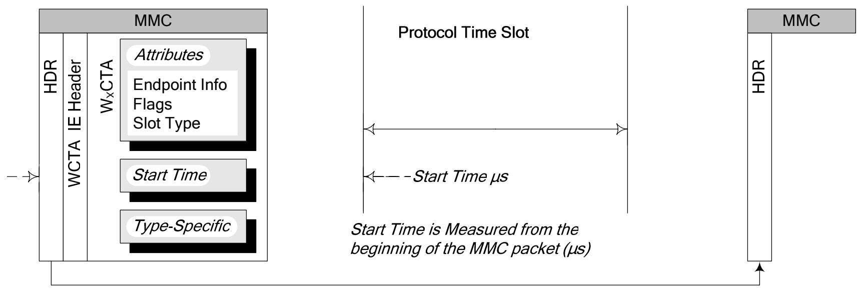

Figure 5-4. Structure of a W_XCTA and Relationship to a Time Slot

Start times for adjacent protocol time slots must be separated by enough channel time to accommodate sending the information packets at the specified signaling rate, plus packet overheads (preambles, inter-packet gaps for streaming-mode data phase) plus an inter-slot idle time (see Section 5.3 for full timing constraints).

There are several types of Wireless USB Channel Time Allocation blocks (W_XCTA) that can be used in a Wireless USB Channel Time Allocation IE, including device receive, device transmit and device notification (a management form of device transmit). All W_XCTA blocks have a common header portion, which includes an attribute field and the time slot information (see Table 5-5).

The types of W_XCTA blocks are:

- W_{DR}CTA (Device Receive). The targeted Function Endpoint must listen for packet transmissions during the described time slot.
- W_{DT}CTA (Device Transmit). The targeted Function Endpoint must transmit information during the described time slot.
- W_{DNTS}CTA (Device Notification Time Slot). This is a management time slot reserved for use by Wireless USB devices to send only device notifications to the host.

Table 5-5. Wireless USB W_XCTA Block Common Header

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	This bitmap has the following encoding: **Bit** / **Value** / **Description** 5:0 / Variable / Depends on value of W_XCTA Block Type code field 7:6 / Enum / W_XCTA Block Type code. Encodings are: **Value** / **Tag** 00B / W_{DR}CTA 01B / W_{DT}CTA 10B / W_{DNTS}CTA 11B / Reserved
1	*wStart*	2	Number	The units of this field are in micro-seconds. The value is measured from the beginning of the preamble of the MMC packet where this W_XCTA Block is transmitted.

Bits [7:6] of the *bmAttributes* field are the W_XCTA Block Type Code. The value in this field indicates the format of the entire W_XCTA Block. The rules for how the W_XCTA Block Type code affects the interpretation of the W_XCTA are provided below:

W_XCTA Block Type Code	**Interpretation of Bits [5:0]**
W_{DR}CTA or W_{DT}CTA	Time slot allocation is for a Wireless USB transaction data packet. Bits [3:0] are the USB device Endpoint number. Bit [4] is dependent on W_XCTA type, see below. Bit [5] is a flag indicating that the time slot is associated with a SETUP stage of a control transfer. When this bit is a one, then the 8 bytes immediately following the W_XCTA are the SETUP Data bytes.
W_{DNTS}CTA	This time slot allocation is for a DNTS (Device Notification Time Slot). Bits [5:0] will be set to zero by the host controller and must be ignored by devices.

The *wStart* field value is always expressed in micro-seconds. Note that the *wStart* value is expressed as an offset from the beginning of the MMC as a synchronization point. The host calculates the value for *wStart* based on the sum of: standard PHY packet overheads, data payload size, data payload bit signaling rate and appropriate inter-slot idle time. Section 5.3 details the rules for calculating minimum inter-slot idle times. Sections 5.2.1.1 through 5.2.1.3 describe the details of these different Wireless USB Channel Time Allocation IEs.

Wireless USB Channel Time Allocation IEs may contain a mix of W_XCTA block types. The Wireless USB host must construct the IE so that they are ordered in time (i.e. increasing start order). In addition, the IEs must be constructed with any W_{DR}CTAs followed by a W_{DNTS}CTA (if present) and then by any W_{DT}CTAs (i.e. OUTs followed by INs). This allows better channel utilization because it minimizes the additional overhead of transmitter/receiver switches (bus turns). See Section 7.5 for rules for ordering IEs in MMC packets.

The following sections detail the different types of channel time allocation blocks. The shading in each of the block format tables indicates the common header portion.

5.2.1.1 Wireless USB Device Receive W_{DR}CTA Block

W_{DR}CTAs (Device Receive) channel allocation blocks describe a time slot in which a device is required to listen for data packets to the addressed OUT endpoint number.

Table 5-6. W_{DR}CTA Block Format

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	This bitmap has the following encoding **Bit** / **Value** / **Description** 3:0 Variable USB Endpoint Number 4 Zero Reserved 5 Boolean Setup Flag 7:6 W_{DR}CTA W_{DR}CTA block type
1	*wStart*	2	Number	See Table 5-5
3	*bDeviceID*	1	Number	Device Address of the Wireless USB device.

5.2.1.2 Wireless USB Device Transmit W_{DT}CTA Block

W_{DT}CTA (Device Transmit) channel allocation blocks describe a time slot in which a device is required to transmit data. These time slots are used for two purposes, one is to transmit data or handshake from the addressed IN endpoint number and the other is to transmit a handshake from the addressed OUT endpoint number. The host must correctly annotate the W_{DT}CTA block to disambiguate about which endpoint should respond during the time slot.

Table 5-7. W_{DT}CTA Block Format

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	This bitmap has the following encoding: **<u>Bit</u>** / **<u>Value</u>** / **<u>Description</u>** 3:0 / Variable / **USB Endpoint Number** 4 / Variable / **Direction.** This field is used to indicate which device endpoint number should transmit during the time slot. Encodings are: 0 OUT Endpoint Number should respond with handshake packet 1 IN Endpoint Number should respond with data or handshake packet 5 / Boolean / **Setup Flag** 7:6 / W_{DT}CTA / **W_{DT}CTA block type**
1	*wStart*	2	Number	See Table 5-5
3	*bDeviceID*	1	Number	Device Address of the Wireless USB device.

Table 5-7. WDTCTA Block Format (cont.)

Offset	Field	Size	Value	Description
4	*bmTXAttributes*	4	Bitmap	These sub-fields indicate the use parameters of the data phase time slot: **Bit** — **Description** 14:0 — **Active TX Packet Size.** This field contains the maximum size of the data payloads a device must use for packet transmissions during the data phase. 15 — **ControlStatusStageFlag.** When this field has a 1B value, the associated protocol time slot is for a Status Stage handshake. 20:16 — **PHY_TXRate.** Refer to Section 5.6. 23:21 — **Transmit Power.** The value of this field selects the transmit power level the device must use to transmit data packets during the data phase protocol time slot. 28:24 — **Transaction Burst Size.** This field is used by the host to modify the configured burst size for the current transaction. The value in this field is the maximum number of data packets the device may send during the protocol time slot. Valid values are in the range [00000B – 10000B]. All other values are reserved. 31:29 — **Data Burst Preamble Policy.** This field is an encoded value used to specify how the device must use standard preambles between data packets in the burst data phase. The encoded values indicate how standard preambles must be used in the data burst. The first preamble is always a standard preamble. **Value** — **Meaning** 000B — Use only standard preambles 001B — Every 2nd packet 010B — Every 4th packet 011B — Every 8th packet
8	*bvDINAck*	4	Bit Vector	Data burst acknowledgement bit vector. Refer to Section 5.4 for detailed explanation.

The *bmTXAttributes.Active TX Packet Size* field is used by the host to instruct the device about what packet payload size the device must use for data packets transmitted during the data phase protocol time slot. The maximum value a host can put into this field is the configured *wMaxPacketSize* for the function endpoint. If the host uses a value larger, the behavior is undefined. The device does not use this field for a handshake packet and the host must set this field to a zero.

When the *bmTXAttributes.ControlStatusStageFlag* field is a 1B, it signals to the device that the current Control transfer on the endpoint is transitioning to the Status Stage. The device must transmit a handshake packet during the assigned protocol time slot. The *bmAttributes.Direction* field must be set to a 1B by the host and the device must ignore the *bmAttributes.Direction* field.

The *bmTXAttributes.PHY_TXRate* field specifies the bit signaling rate the device must use for the data payload portion of packets sent during the associated time slot. This transmit rate only applies to this time slot. The host may change the value of this field at any time. The host must only use values in this field which the device has explicitly noted that it supports. The host must set this field to 00000B when the W_{DT}CTA is used for a handshake packet.

The *bmTXAttributes.Transmit Power* field is used to specify the transmit power level the device must use to transmit all of the data packets during the associated data phase protocol time slot. In general, a value of zero (0) selects the highest power setting and a value of seven (7) selects the lowest. Refer to Section 4.10.3 for details.

The host sets the value of *bmTXAttributes.Transaction Burst Size* to reflect the number of data packets it is prepared to receive during the allocated protocol time slot. The device must transmit the requested number of data packets in the burst if the associated data is available. The device must not transmit more data packets than the requested burst size during the allocated protocol time slot. If the host uses a value in this field which is larger than the configured Maximum Burst Size for the function endpoint, the resulting behavior is undefined.

The *bmTXAttributes.Data Burst Preamble Policy* field is used by the host to instruct the device how to use streaming and standard preambles during the next DATA IN phase data burst. The encoded value is a power-of-two step function of how frequently the device must insert standard preambles between the data packets of the burst. Note that the physical layer bursting rules require that a data burst must always begin with a standard preamble. The host must set this field to 000B when this W_{DT}CTA is used for a handshake packet. The host must set this field to a value of 000B when *PHY_TXRate* is 200 Mb/s or below (see reference [4]). If the host uses this feature on a function endpoint where it is not supported, the results are undefined.

The *bvDINAck* field is the handshake to a DATA IN burst. It is a bit vector where each bit location corresponds to a data sequence value (*bvDINAck[0]* corresponds to sequence value 0, *bvDINAck[1]* to sequence value 1, and so on). Refer to Section 5.4 for a full description of the data burst sequence rules. For a W_{DT}CTA targeted at an OUT endpoint, the device does not use this field and the host must set this field to a zero.

5.2.1.3 Wireless USB Device Notification W_{DNTS}CTA Block

DNTS time slots are allocated by the host to allow individual devices to send small, asynchronous notification messages to the host. The host notifies devices in its Wireless USB cluster of a DNTS by including a W_{DNTS}CTA block in an MMC. The format of a W_{DNTS}CTA block is illustrated in Table 5-8. A host may include at most one W_{DNTS}CTA block in a WCTA_IE.

Device Notification Time Slots are logically structured as a window of uniform sized message slots. Message slots in a DNTS are large enough for a device to transmit a maximum sized device notification plus guard-band to allow for local device clock drift. The W_{DNTS}CTA describing a DNTS instance includes the number of message slots in the instance.

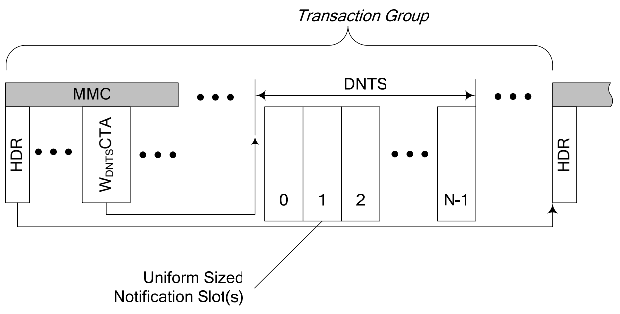

Figure 5-5. Generic model for Organization of a DNTS

The access mechanism for devices to transmit during a DNTS is Slotted Aloha. A device that has a device notification to transmit selects a message slot in a DNTS using a uniformly distributed random integer value, in the range {0, N-1} where N is the number of message slots in the DNTS instance. A device will begin transmitting the device notification packet at the point it determines the start of the message slot, measured from the beginning of the previous MMC packet, based on its local clock. Individual time slots within a DNTS have a fixed duration of $\mathbf{t}_{\text{NOTIFICATIONSLOT}}$. A W_{DNTS} must be scheduled to occur within 25ms of its associated MMC packet.

Table 5-8. W_{DNTS}CTA Block Format

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	This bitmap has the following encoding: **Bit** / **Value** / **Description** 3:0 / Zero / Reserved 4 / Zero / Reserved 5 / Zero / Reserved 7:6 / W_{DNTS}CTA / W_{DNTS}CTA block type
1	*wStart*	2	Number	See Table 5-5
3	*bNumslots*	1	Number	The value in this field is the raw number of notification message time slots available in the DNTS.

5.3 Transaction Group Timing Constraints

The host must order individual transactions within transaction groups to minimize the number of 'bus turns' (e.g. turning the hosts' radio from Transmit to Receive or Receive to Transmit). This means that a transaction group must be constructed to have all the host Transmit protocol time slots immediately after the MMC, followed by a 'bus turn' and then all Device Transmit protocol time slots. Note there is another 'bus turn' between the last Device Transmit time slot and the next scheduled MMC packet (see Figure 5-6).

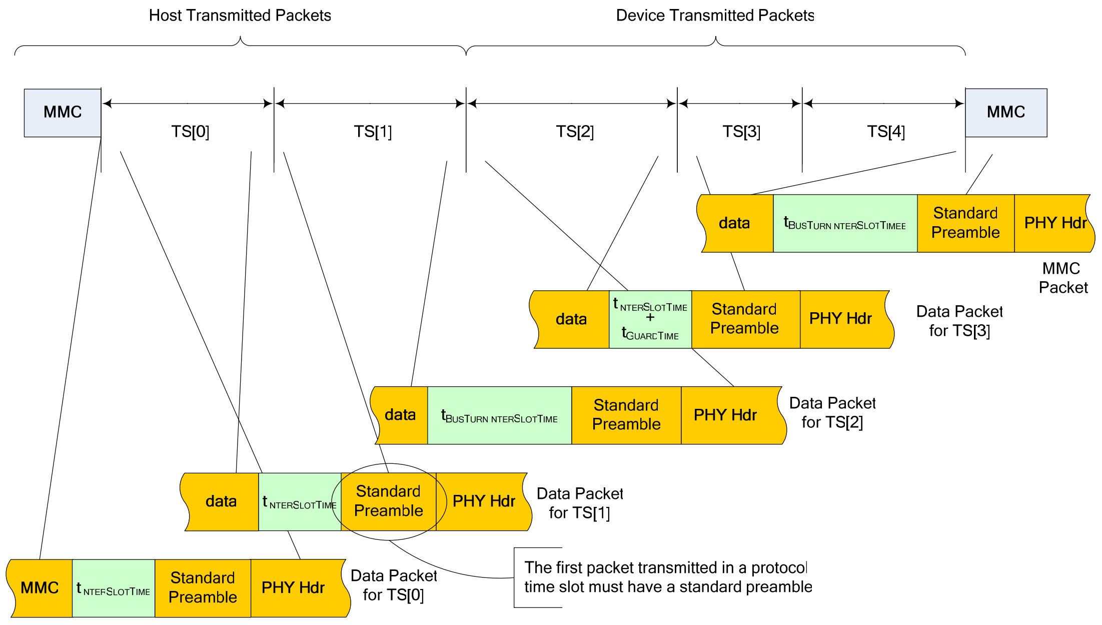

Figure 5-6. Example Wireless USB Transaction Group Organization

An inter-slot time is the time between the end of the last packet transmission of one protocol time slot to the start of the next protocol time slot (or transmission of an MMC packet). In general, the intent of the inter-slot time is to ensure that transmissions between protocol time slots do not overlap. Inter-slot times must be long enough in duration to guard against the maximum clock drift between a device's local clock and the ideal clock, see Figure 5-7 (which also provides general the method for calculating a guard time (**t**$_{\text{GUARDTIME}}$).)

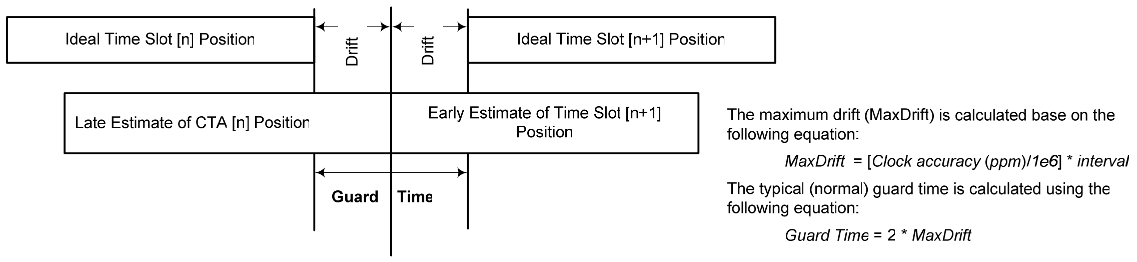
Ideal Time Slot [n] Position
Drift
Drift
Ideal Time Slot [n+1] Position
Late Estimate of CTA [n] Position
Early Estimate of Time Slot [n+1] Position
Guard
Time
The maximum drift (MaxDrift) is calculated base on the following equation:
MaxDrift = [Clock accuracy (ppm)/1e6] * interval
The typical (normal) guard time is calculated using the following equation:
Guard Time = 2 * MaxDrift

Figure 5-7. TDMA Slot Guard Time Reference

Note that the *ppm* (parts per million) term depends on the clock rate of the PHY and the maximum drift is a function of the elapsed time since the last synchronization event (i.e. the *interval*). In order to minimize the effects of Guard Time on the available bandwidth, Wireless USB uses MMCs as clock synchronization reference points.

The following discussion applies to determining the minimum timing constraints for allowable inter-slot time. Inter-slot times are, as noted above, a timing component used only by the host to calculate individual time slot durations. Actual inter-slot times are host-implementation dependent, but must meet the minimum requirements described below.

The first two protocol time slots in Figure 5-6 indicate OUT (host to device) transmissions. Protocol time slots in a transaction group must be ordered OUT then INs, so the host is the transmitter of all the packets beginning from an MMC until the first IN protocol time slot. This looks in many respects like a burst transfer (supported in many PHY standards), although the recipient devices in adjacent protocol slots may be different for this application. For adjacent OUT protocol time slots, the minimum inter-slot time must be **t**$_{\text{INTERSLOTTIME}}$. There is no need to add guard times between these consecutive OUT transactions (or between the MMC and the first OUT,) since the host is the transmitter of all these packets. The receiving device, however, must start listening at least a calculated guard time (**t**$_{\text{GUARDTIME}}$) before the anticipated packet start time. The Wireless USB standard inter-slot time (**t**$_{\text{INTERSLOTTIME}}$) is a PHY-related timing parameter, see Table 5-11.

During device to host (IN) transactions, the minimum inter-slot time between successive IN transactions must be **t**$_{\text{INTERSLOTTIME}}$ plus **t**$_{\text{GUARDTIME}}$ since the IN time slots could potentially be used by different transmitters with drifting clocks.

When an MMC or OUT protocol time slot is followed by an IN protocol time slot, or an IN protocol time slot is followed by an MMC, then the minimum inter-slot time must be equal to the calculated guard time (**t**$_{\text{GUARDTIME}}$) plus the host's bus switch time (**t**$_{\text{BUSTURNTIME}}$) which is a PHY-related timing parameter, see Table 5-11. The sum of these is the bus turn inter-slot time (**t**$_{\text{BUSTURNINTERSLOTTIME}}$). Figure 5-6 illustrates these inter-slot idle times.

The final components in calculating protocol slot time durations are the inter-packet gaps and size of preambles between packets in slots where multiple packets are transmitted (e.g. burst-mode packet transmissions). PHY standards may (or may not) define a minimum and maximum value for burst-mode inter-packet gaps and the use of streaming preambles (which can be shorter than standard preambles). When necessary, Wireless USB does define a maximum requirement for the streaming mode inter-packet gaps, see Table 5-11. The availability of streaming preambles is also a PHY-specific parameter. Section 5.3.1 summarizes the streaming-mode timing constraints for the PHY. These rules and parameters allow a host implementation to calculate protocol slot time durations with only the transmit rate, number and size of the data packets as variables in the calculation, all other slot time terms are constants.

Figure 5-8 summarizes component parts the host takes into consideration when calculating the durations of data phase time slots when the burst-mode size is greater than one. Regardless of the direction of transmit (OUT or IN), the first packet in a protocol time slot is required to have a standard preamble. Between each data packet is an allowance for a streaming-mode inter-packet gap ($\mathbf{t}_{\mathbf{STREAMIPG}}$) and either a streaming-mode or standard preamble. At the end of the time slot is an allowance for the inter-slot idle time (see above discussion around Figure 5-6).

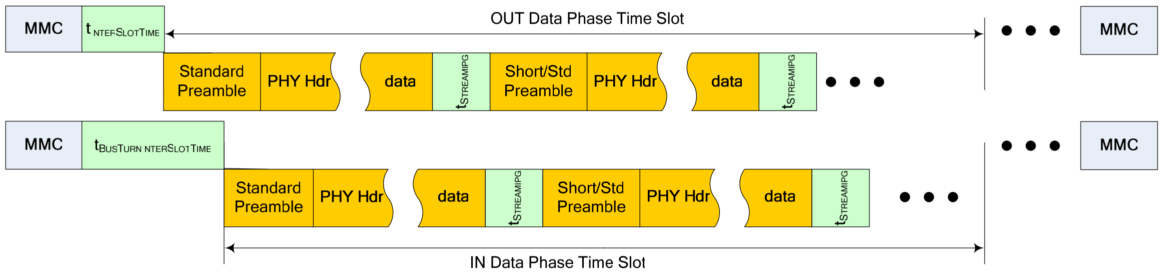

Figure 5-8. Example Wireless USB Burst Data Phase Time Slot Layout

Figure 5-8 also illustrates the timing constraints for protocol time slots immediately following an MMC packet. When the protocol time slot following an MMC is an OUT the MMC and the first data packet in the OUT time slot must be separate by $\mathbf{t}_{\mathbf{INTERSLOTTIME}}$. When the protocol time slot following an MMC is an IN, the MMC and the first data packet being transmitted by the device must be separated by $\mathbf{t}_{\mathbf{BUSTURNINTERSLOTTIME}}$.

5.3.1 Streaming-Mode Inter-packet Constraints for the PHY

The PHY standard defines a strict set of rules for implementing streaming-mode data transmissions. A summary of the rules are repeated below. The final standard authority on differences between this specification and the PHY standard is the PHY standard.

The first data packet in a Wireless USB data burst (i.e. multi-data packet data phase) must always use a standard (length) preamble. All packets in a Wireless USB data burst must be separated by $\mathbf{t}_{\mathbf{STREAMIPG}}$. For data rates of 200Mb/s and lower, all data packets of the burst must use a standard preamble. For data rates greater than 200Mb/s, the host may use streaming mode preambles for OUT data phase bursts and will instruct via the *bmTXAttributes.Data Burst Preamble Policy* W_{DT}CTA parameter the pattern of streaming and standard preambles to use for the device's burst transmission during the associated protocol time slot.

5.3.2 Protocol Synchronization

All Wireless USB Protocol timings are specified relative to the beginning of the preamble for the MMC packets. Figure 5-9 illustrates Wireless USB protocol synchronization and relative reference points.

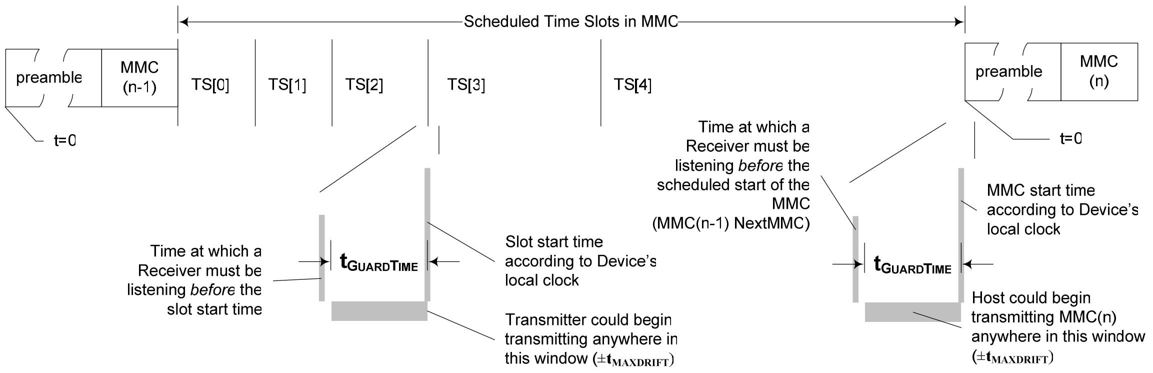
Scheduled Time Slots in MMC
preamble
MMC (n-1)
TS[0]
TS[1]
TS[2]
TS[3]
TS[4]
preamble
MMC (n)
t=0
t=0
Time at which a Receiver must be listening before the scheduled start of the MMC (MMC(n-1) NextMMC)
MMC start time according to Device's local clock
tGUARDTIME
Time at which a Receiver must be listening before the slot start time
Slot start time according to Device's local clock
Transmitter could begin transmitting anywhere in this window (±tMAXDRIFT)
tGUARDTIME
Host could begin transmitting MMC(n) anywhere in this window (±tMAXDRIFT)

Figure 5-9. Protocol Timing Relative to MMC

Devices reset their protocol clocks to zero at the beginning of an MMC preamble. All channel time offsets (*nextMMC* and W_{X}CTA time slot allocations) in the MMC are specified by the host relative to the start of the preamble for the current MMC.

A device may idle its radio after the MMC packet and the Start Time for time slots it is designated to be either a Transmitter or Receiver. When designated as a Transmitter, the device (or host) must begin transmitting its preamble at the point it determines the start of the time slot (or MMC), measured from the beginning of the last MMC packet, based on its local clock. When designated as a Receiver, the device (or host) must begin listening at least a calculated guard time (**$t_{GUARDTIME}$**) before the point it determines the start of the time slot, based on its local clock (see Figure 5-9).

It is the responsibility of the host to ensure allocated time slots are large enough to accommodate the data communications intended to occur during the time slot. Wireless USB devices must preserve the integrity of allocated time slots. For IN protocol time slots, this means that the device must not transmit before its local clock indicates the start of its time slot. For OUT protocol time slots, the device may turn off its receiver when its local clock indicates the adjacent protocol slot start time (unless the adjacent slot time is for a different endpoint on the same device). Devices derive the slot boundaries from the WCTA_IE information in the IE. The W_XCTAs must always be provided in order, which means the device can derive the slot boundaries based on its W_XCTA's *wStart* field and the *wStart* field of the next (adjacent) W_XCTA. This means that the host must always provide an 'end of list' W_XCTA in an MMC, which always provides a termination of W_XCTA.wStart fields for 'real' endpoint transaction. The 'end of list' W_XCTA must be the last W_XCTA block in a WCTA_IE. The 'end of list' W_XCTA block must not be interpreted as a W_XCTA for use with a valid Function Endpoint. To ensure this, the 'end of list' W_XCTA block has the field values specified in Table 5-9.

In the case were the last W_XCTA before the EOL W_XCTA is a W_{DR}CTA then the *nextMMC* time must be at least a calculated guard time (**$t_{GUARDTIME}$**) larger than the *wStart* value of the EOL W_XCTA. In the case of a W_{DT}CTA before the EOL W_XCTA, the *nextMMC* time must be at least **$t_{BUSTURNTIME}$** + **$t_{GUARDTIME}$** larger than the *wStart* value of the EOL W_XCTA. Note that larger in this context must accommodate appropriately for channel time rollover conditions.

Table 5-9. Required End of List W_XCTA Block Values

Field	Sub-Field	Value
bmAttributes	USB Endpoint Number	0000B
	Reserved/Direction	0B
	Setup Flag	0B
	W_XCTA Block Type	00B
wStart	N/A	Set to appropriate value for context of use.
bDeviceID	N/A	Broadcast Cluster ID

5.4 Data Burst Synchronization and Retry

Wireless USB provides a mechanism to guarantee data sequence synchronization between the data transmitter and the data receiver across multiple transactions with data bursts of different sizes. This mechanism provides for identifying required data order, guarantees that handshake information is interpreted correctly by transmitter and receiver and guarantees advancement of the data stream only after reliable data delivery has been accomplished.

Data bursting as defined here provides a mechanism for reliable delivery of data between a Transmitter and a Receiver. The Transmitter may transmit more than one data packet per data phase and the Receiver must provide information during the handshake phase acknowledging that data was received. The method for annotating the required packet sequence and acknowledgement mechanism provides a structure for efficient retransmission of lost data burst packets (i.e. only lost packets are retransmitted). This means that, from the Receiver's point of view, data packets will at times, appear to arrive 'out of order'. Packets transmitted during a data phase time slot usually have a PID field value of DATA (or IDATA). Note that a device is allowed to respond to a W_{DT}CTA "token" with a single Handshake packet. A device is not allowed to mix DATA (or IDATA) and Handshake packets in the same data phase. The Wireless USB Header in a data packet has a Sequence Number field (see Table 5-1) which is used as data sequence counter. This counter and the rules described below allow the Receiver to reconstruct the order intended by the Transmitter. The data sequencing mechanism defined below uses a relatively small range of sequence values, which allows for a small footprint overhead per data packet while providing a reasonable degree of bursting capability even in the face of errors and retries.

Wireless USB data bursting uses a simple sliding window protocol that provides support for reliable data delivery. The sliding window protocol ensures that a transmitter always uses data sequence numbers in strict ascending sequence order, so a receiver can use received data packets using the same ordering rules, thus preserving the packet ordering intended by the transmitter. Figure 5-10 illustrates the general data flow model for wireless USB data bursting.

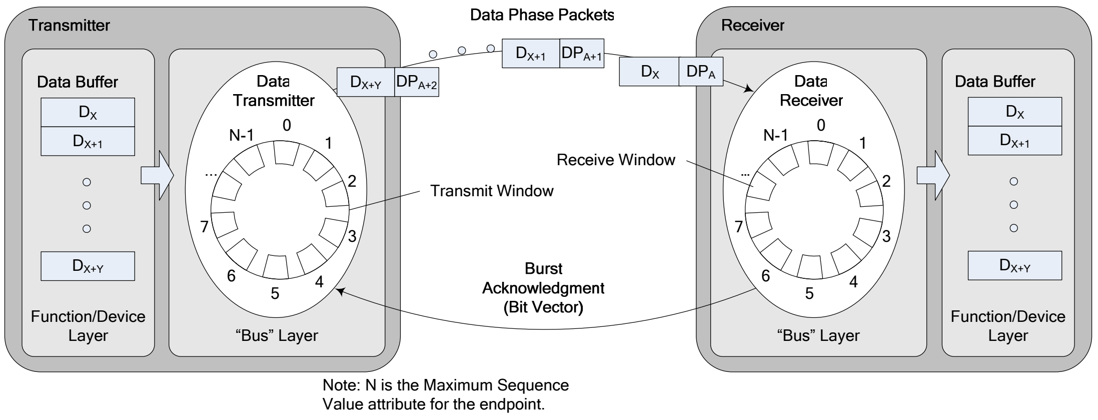

Figure 5-10. General Data Burst Data Sequencing Rules

The Transmitter has a data stream that is logically segmented into Maximum Packet sized portions (D_X through D_{X+Y}). It also maintains a sliding *transmit* window that controls how sequence numbers are associated with each data packet for the next transaction data phase. The Transmitter must associate sequence numbers with data buffer segments in strict, ascending sequence number order.

The Receiver maintains a *receive* window that identifies which data sequence numbers (and by association which data packets) it will retain for use from the next transaction. It also provides burst acknowledgment information during the handshake phase of the transaction. The Receiver must use data received in strict ascending sequence number order (except in Isochronous discard cases).

Each device endpoint has attributes to support data bursting. A device endpoint reports its bursting attributes via information in the Wireless USB Endpoint Companion Descriptor (see Section 7.4.4). The bursting attributes of each endpoint include the following:

- Maximum Packet Size – This is nominal data unit size for all data packets.
- Maximum Burst Size – This is the largest number of packets an endpoint can accommodate in a single data phase. A device must provide enough buffering to accept at least Maximum Packet Size * Maximum Burst Size bytes. A device may provide more buffering for better performance.
- Maximum Sequence – This is the range of sequence numbers that must be used when transferring data to this endpoint. The **N** value in Figure 5-10 is the Maximum Sequence value. The actual range of sequence numbers for the endpoint is zero to (Maximum Sequence – 1).

In addition to these attributes, each function endpoint uses the following parameter for data bursting control.

- Maximum Sequence Distance – This is the range of current transmit/receive window that can be maintained by the transmitter/receiver. This indicates the maximum difference between the smallest sequence counter value and the largest value of the window including both values. The Maximum Sequence Distance value is the Maximum Sequence value minus one. The Maximum Sequence Distance must never be exceeded.

If the transmitter sends a packet with a sequence number that is outside of the current receive window, the receiver must ignore the packet. At any point in time, transmit and receive windows are never larger than the Maximum Burst Size. Transmit and receive windows will occasionally be different in size, for example when the transmitter has less than Maximum Burst Size number of data packets to send. The general rules of the transmitter and receiver to maintain transmit and receive windows are:

- When a configuration event occurs (SetConfiguration, SetInterface, ClearEndPointFeature), transmit and receive data sequences are reset to start at zero (0), the transmit window is initialized to send up to the Maximum Burst Size number of data packets for the next transaction and the receive window is initialized to receive up to the Maximum Burst Size number of data packets. Note that the actual size of the transmit window for any transaction depends on two factors: the actual amount of pending transmit data and or the size of the receive window (whichever is smaller).
- During the Data Stage of a transaction, the transmitter will transmit all of the data packets in the transmit window. The receiver will advance the receive window one location for every packet it successfully receives. The advancement of the receive window must always be modulo Maximum Sequence Size.
- During the Handshake Phase, the receiver provides a burst acknowledgement. Note that when the transaction is an OUT transfer, the burst acknowledgement is in the data payload of a Handshake Packet. When an IN transaction, the burst acknowledgement is in a subsequent MMC (W_{DT}CTA information element). The burst acknowledgement value is the current receive window value, formatted as a bit-mask. The burst acknowledgement is a bit vector representation of the receive window. The '1' bits in the bit vector represent the receive window.

  The transmitter must use the burst acknowledgement data value to advance it's transmit window, in preparation for the next transaction data phase. If the transmitter does not correctly receive an acknowledgement, it does not advance it's transmit window. There are some exceptions for isochronous discard scenarios.

Figure 5-11 illustrates the generic model of data bursting for a Wireless USB transaction with some randomly selected endpoint bursting attributes. The model demonstrated below works regardless of a function endpoint's attribute values. The endpoint attributes for this example are a Maximum Burst Size of four (4) and a Maximum Sequence value of 10. Based on the rules above, the sequence value range used for the data bursting stream is [0-9]. On the left-hand side of the figure is the initial condition for transmit and receive windows. The shaded slots are part of the current 'window', and each window is initialized with burst size number of slots. During the data phase, the Transmitter sends only the packets in the current transmit window. As the Receiver lands data packets during the Data Phase of the transaction, it advances the receive window for each successfully received packet (based on the observed sequence number). In the Handshake phase of the transaction, the receiver provides a bit vector which is the current receive window. This bit vector directly indicates which sequence numbers the Transmitter is allowed to use in the next transaction.

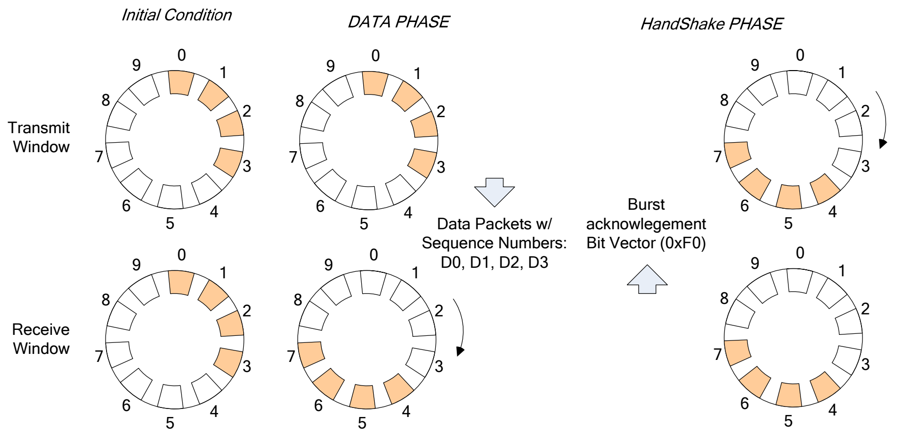
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Receive Window
Data Packets w/ Sequence Numbers: D0, D1, D2, D3
Burst acknowlegement Bit Vector (0xF0)
0
1
2
3
4
5
6
7
8
9

Figure 5-11. Example Sliding Window Tracking (Burst Size = 4; Max Sequence = 10)

The example in Figure 5-12 demonstrates sliding window sequence through an entire pass through all of the sequence numbers of the example endpoint. Note that by the end of the Transaction 1, the transmit window spans the last two sequence numbers and the first two sequence numbers. This example shows that any number for the Maximum Sequence that meets the minimum requirements (> 2 X Maximum Burst Size) will result in acceptable bursting behavior with regards to proper sequencing.

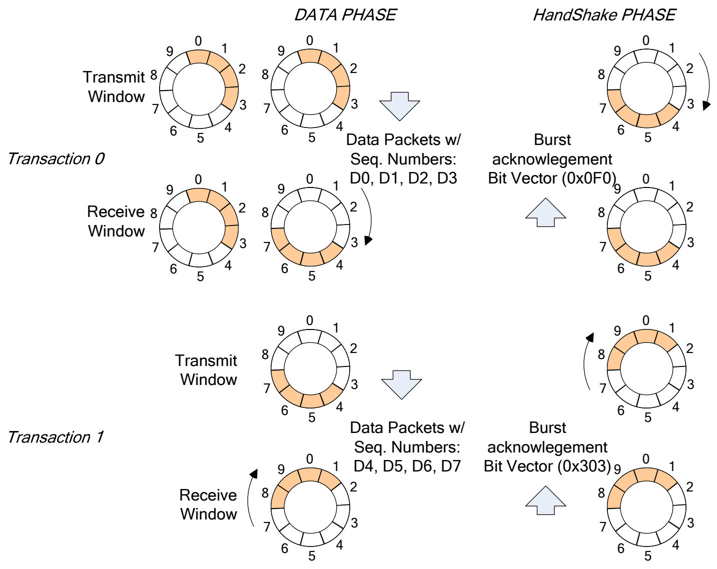
DATA PHASE
HandShake PHASE
Transmit Window
Receive Window
Transaction 0
Data Packets w/ Seq. Numbers: D0, D1, D2, D3
Burst acknowlegement Bit Vector (0x0F0)
Transaction 1
Data Packets w/ Seq. Numbers: D4, D5, D6, D7
Burst acknowlegement Bit Vector (0x303)

Figure 5-12. Example Sliding Window, Full Rotation (Modulo Max Sequence)

The example in Figure 5-13 demonstrates sliding window sequence through a scenario where packets are lost during the data phase, and subsequent retries and recovery.

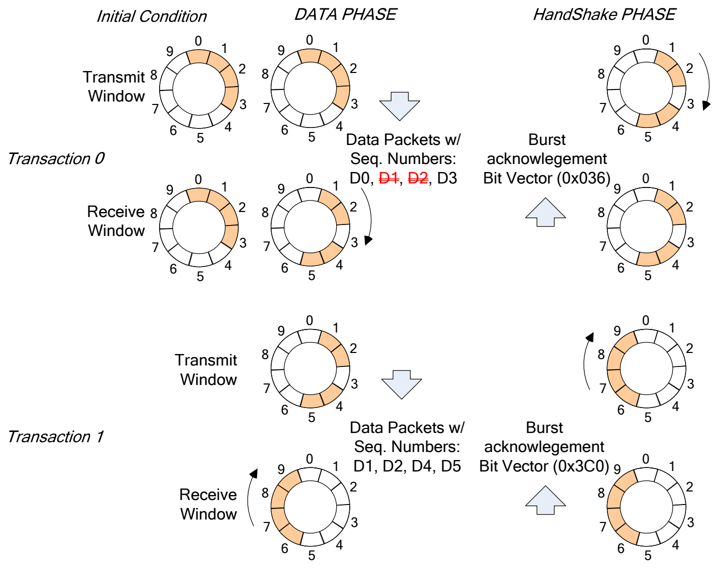
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Receive Window
Transaction 0
Data Packets w/ Seq. Numbers: D0, ~~D1~~, ~~D2~~, D3
Burst acknowlegement Bit Vector (0x036)
Transaction 1
Data Packets w/ Seq. Numbers: D1, D2, D4, D5
Burst acknowlegement Bit Vector (0x3C0)

Figure 5-13. Example Sliding Window, Smashed Packets & Recovery

In transaction 0, data packets with sequence counter values 1 and 2 are not correctly received. The Receiver advances the receive window for the sequence numbers it does see, but also retains the window over the packets it has not yet seen. The resultant window mask is returned to the Transmitter as the burst acknowledgement value during the handshake phase of the transaction. The Transmitter takes the burst acknowledgement and updates it's transmit window to match the receiver window. It then retransmits the lost data packets (with the same sequence numbers) and then transmits new data packets for the new portion of the transmit window. The Transmitter will issue retry packets before transmitting new data packets.

The example in Figure 5-14 demonstrates sliding window sequence through a scenario where the Handshake Phase encounters some corruption and the burst acknowledgement does not make it back to the transmitter.

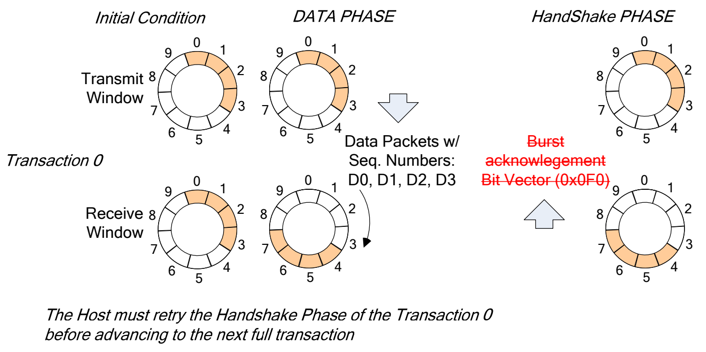
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Transaction 0
Receive Window
Data Packets w/ Seq. Numbers: D0, D1, D2, D3
~~Burst acknowlegement Bit Vector (0x0F0)~~
The Host must retry the Handshake Phase of the Transaction 0 before advancing to the next full transaction

Figure 5-14. Example Sliding Window, Smashed Handshake

During transaction 0, the receiver successfully lands all of the data packets, updates it's receive window and sends the burst acknowledgement information in the handshake phase. The handshake information does not reach the Transmitter, so it does not advance the transmit window. There is a protocol rule that a transfer cannot be advanced to the next transaction until the handshake has been received by the Transmitter. In the case of an OUT (host to device transaction) where the handshake packet gets lost, the host must retry the Handshake phase ONLY, until it receives a good handshake before advancing to the next full transaction. In the case of an IN (device to host) the handshake information is included in the MMC with the token for the next transaction. By inference if the device receives the token for the next transaction without error, it has received the handshake information for the previous transaction without error.

The example in Figure 5-15 demonstrates sliding window sequence through a scenario where the Receiver's application layer does not consume data from the bus layer at a rate that allows the system to burst a full burst size (because buffering is not available at the bus layer). In Figure 5-15, the gray stripe around the receive window represents occupied data packet buffers that are not available to receive new data packets from the Transmitter.

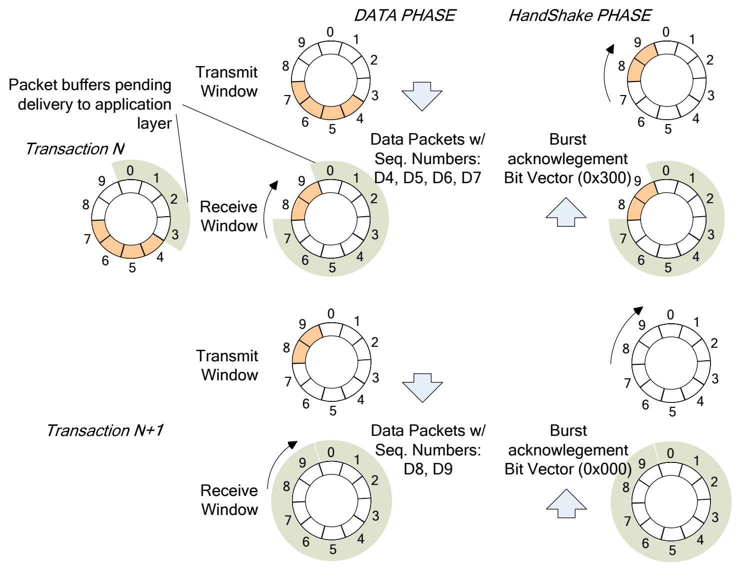
DATA PHASE
HandShake PHASE
Packet buffers pending delivery to application layer
Transmit Window
Transaction N
Receive Window
Data Packets w/ Seq. Numbers: D4, D5, D6, D7
Burst acknowledgement Bit Vector (0x300)
Transmit Window
Transaction N+1
Receive Window
Data Packets w/ Seq. Numbers: D8, D9
Burst acknowledgement Bit Vector (0x000)

Figure 5-15. Example Sliding Window, Flow Control Scenario

In transaction N, the Receiver's buffers are partially occupied, so in the burst acknowledgement, it tells the Transmitter that it can only receive two packets (D8 and D9, respectively). In transaction N+1, the transmitter sends D8 and D9, which fills all of the available buffering on the endpoint, so the burst acknowledgement value indicates that the Receiver has no buffering available. This is a flow control event. When the data direction is an OUT (host to device), the host will interpret the burst acknowledgement of all zeros as a flow control event and remove the endpoint from the actively scheduled endpoints. Refer to Section 5.5.4 for details on flow control.

The example in Figure 5-16 demonstrates sliding window sequence through a scenario where a packet keeps getting lost during the data phase, and the receiver cannot advance the window beyond the Maximum Sequence Distance. In this example, the Maximum Sequence Distance value is 9, which is the Maximum Sequence value 10 minus one.

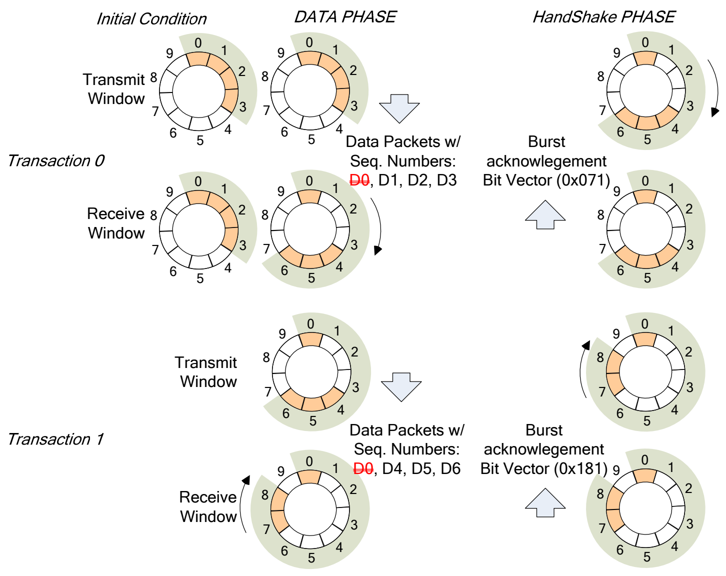
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Transaction 0
Receive Window
Data Packets w/ Seq. Numbers: ~~D0~~, D1, D2, D3
Burst acknowlegement Bit Vector (0x071)
Transmit Window
Transaction 1
Receive Window
Data Packets w/ Seq. Numbers: ~~D0~~, D4, D5, D6
Burst acknowlegement Bit Vector (0x181)

Figure 5-16. Example Sliding Window, Smashed Packets & Maximum Sequence Distance Limit

The sequence distance is defined as the difference between the smallest sequence counter value and the largest value in the current sequence. In the initial condition, the sequence distance is 4, as the smallest value is 0 and the largest value is 3. In Figure 5-16 the outside shadow illustrates the tracking of the sequence distance. In transaction 0, the data packet with sequence counter values 0 is not correctly received. The Receiver advances the receive window and sends a handshake packet (071H). The resultant sequence distance is 7 (0 to 6). In transaction 1, the data packet with sequence counter value 0 is retransmitted with some new data packets, but it gets lost again. The Receiver updates the receive window, however it cannot advance the receive window up to the sequence counter value 9 because it would increase the receive sequence range to 10, which exceeds the Maximum Sequence Distance 9. Therefore, the Receiver advances the receive window so the sequence distance is 9 and sends a handshake packet of (181H), which results in the Transmitter advancing its transmit window to match.

The maximum sequence distance rule allows the burst protocol to work through a ‘stuck-at’ sequence wrap scenario without additional in-stream flags. The example in Figure 5-17 illustrates a continuation of the example stuck-at condition that pushes the burst sequence up to the edge of the maximum sequence distance (started in Figure 5-16).

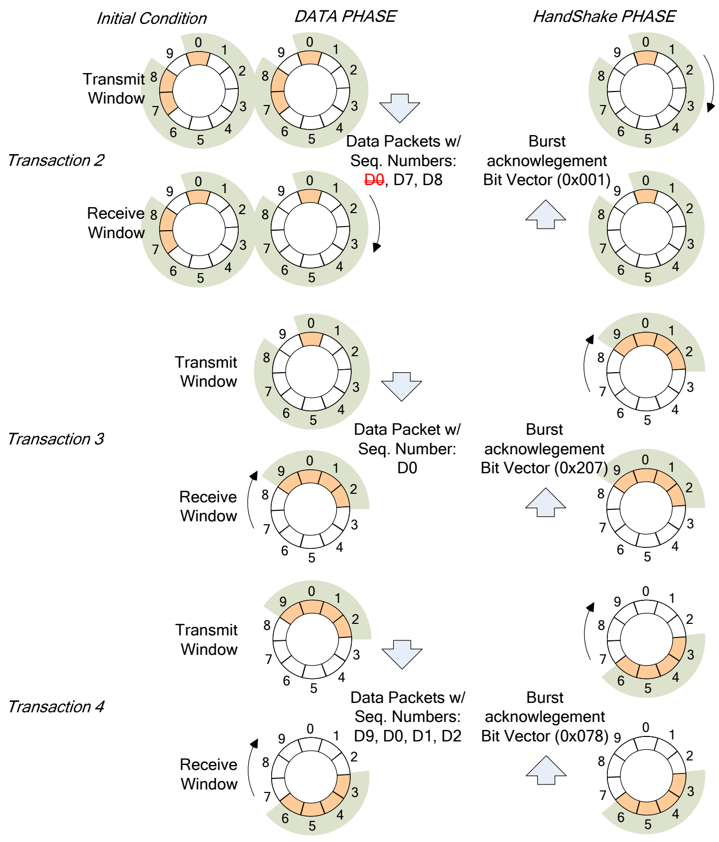
Initial Condition
DATA PHASE
HandShake PHASE
Transmit Window
Receive Window
Transaction 2
Data Packets w/ Seq. Numbers: ~~D0~~, D7, D8
Burst acknowlegement Bit Vector (0x001)
Transaction 3
Data Packet w/ Seq. Number: D0
Burst acknowlegement Bit Vector (0x207)
Transaction 4
Data Packets w/ Seq. Numbers: D9, D0, D1, D2
Burst acknowlegement Bit Vector (0x078)

Figure 5-17. Example Recovery from Stuck-at Wrap Condition

In transaction 2, the Transmitter sends a data burst including D0, D7 and D8. Again, D0 is lost which results in the Receiver recording the correctly received packets, but cannot advance its window (again) due to the D0 being stuck. So the Receiver returns a handshake indicating it is (still) ready for D0 (001H). At this point the transmit and receive windows are only one packet wide. In transaction 3, the Transmitter sends the single data packet with sequence number 0, which is successfully received by the Receiver. The Receiver is able to consume the received data virtually immediately (making the packet buffer free for another packet) which allows the device to finally shrink the sequence distance to zero. This in turn, allows the Receiver to finally advance the receive window to the maximum burst size and return a handshake of (207H). Note, that this particular response is a best-case scenario. Many implementations would not consume data that quickly and free up the receive buffers that quickly (between the end of the data phase and getting ready to transmit the handshake phase). Therefore, a common response in this situation will be a burst acknowledgement of 200H. The transmitter observes that the Receiver has advanced the receive window and can now advance the transmit window accordingly and send all new data in Transaction 4, including re-using sequence value 0 in back-to-back transactions, with no ambiguity whether the data associated with 0 is old or new.

5.5 Wireless USB Transactions

All transfer types in Wireless USB use the same basic transaction format. This format has all of the necessary components to provide for reliable delivery of data by providing means of error detection and retry. As noted in Section 4.4 Wireless USB transaction are split transactions mapped over a TDMA-based structure. Transactions are nominally three phase (Token, Data, Handshake); However under certain flow control and halt (stall) conditions; there may be only two phases to a transaction. MMC and Handshake packets must be transmitted at base rate. Data phase data packets may be transmitted at any supported bit transfer rate.

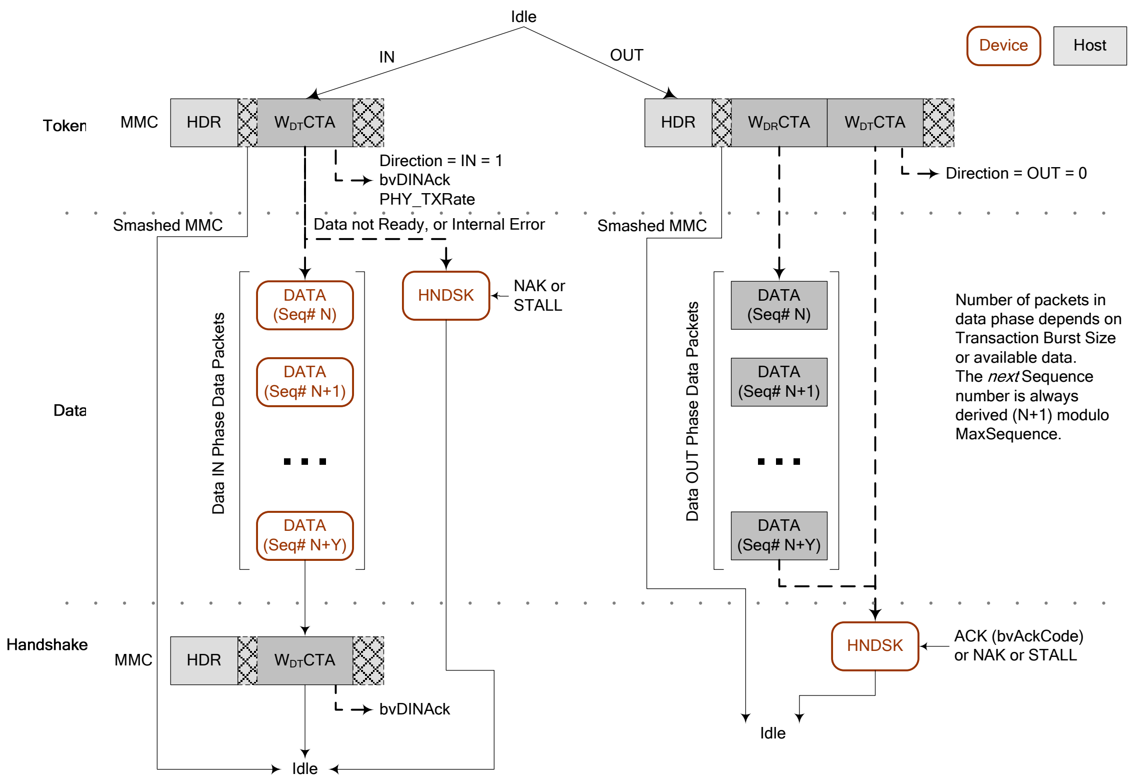

Figure 5-18. General Wireless USB Transaction Format

When a host is ready to receive data, it issues an MMC with a W_{DT}CTA block describing the channel time protocol time slot for the data phase of the transaction. If the function endpoint successfully receives the MMC, updates its transmit window based on the value of *bvDINAck* bit vector field in the W_{DT}CTA block and then responds, (beginning at the start of the assigned protocol time slot) by transmitting either a burst of data packets (one or more), or should it be unable to respond with data, it returns a Handshake packet encoded with a NAK or STALL handshake code. The function endpoint transmits the data packets at the bit transmission rate encoded in the W_{DT}CTA block by the host. If the function endpoint detects an error in the MMC it will not respond to the host at the protocol time slot. During the data phase protocol time slot, the host listens for data packets from the function endpoint. It observes the sequence numbers of received data packets and advances it's receive window accordingly. The acknowledgement of which data packets the host received without error is communicated to the function endpoint in the *bvDINAck* bit vector field in next W_{DT}CTA block addressed to the function endpoint.

Binding the acknowledgement information into the *next* token for the function endpoint saves channel time and protocol overhead and works well while the pipe is streaming data. However, at the end of a transfer (from the host perspective, all buffers provided by the application are full) the function will not receive any acknowledgement until the application provides more buffering and the host resumes transactions. In some cases this may be a long time and is significantly different sequencing requirements from USB 2.0. In order to simplify function implementations, the host has additional operational requirement to get an acknowledgement to the function endpoint as quickly as possible. For a Bulk IN function endpoint, when the host detects that it has no more data buffer, it must schedule at least one *blank* W_{DT}CTA for the function endpoint in subsequent transaction groups. For an Interrupt or Isochronous function endpoint, the host must schedule a *blank* W_{DT}CTA as the last 'transaction' in the service period after it has successfully received packets in the previous transaction and advanced its receive window. A *blank* W_{DT}CTA is one that allocates no channel time and serves only to acknowledge the packets received in the previous data phase.

When a host is ready to transmit data to a function endpoint, it transmits an MMC with two W_XCTA blocks which describe the protocol time slots required to complete the data and handshake phases of the Data OUT transaction. The host uses a W_{DR}CTA block to describe the protocol time slot for the data phase and a W_{DT}CTA block (with the *Direction* field set to OUT (0), indicating the OUT function endpoint must respond with a handshake packet) to describe the protocol time slot for the handshake phase. The function endpoint ignores the value of the *bvDINAck* field when the *Direction* field is set to OUT (0). Note if there is insufficient time in the current reservation to complete both the data and handshake stages of the transaction, the host may transmit the handshake stage W_{DT}CTA block in a later MMC. During the data phase protocol time slot, the host will transmit a burst of data packets, based on the state of the host's transmit window (see Section 5.4). If the function endpoint detects an error in the MMC it will not respond to the host at the handshake protocol time slot. Otherwise, the function endpoint will transmit a handshake packet at the handshake phase protocol time slot (described by the W_{DT}CTA block). The function endpoint will set the *Handshake Code* to one of the following values:

ACK is used to communicate the function endpoint's receive window state to the host. The receive window state is encoded into the handshake packet's *bvAckCode* field. See Section 5.4 for how ACK and the value of *bvAckCode* are interpreted by the host. Note that one or more (up to all) of the data packets transmitted during the data phase protocol time slot may be corrupted when received by the function endpoint. The function endpoint must respond with a handshake packet during the handshake protocol time slot to provide the host the current receive window state. This information allows the host to know which packets need to be retransmitted.

NAK indicates that the function endpoint did not accept any data transmitted by the host during the data stage protocol time slot. This is nominally a flow control response indicating that the function was in a temporary condition preventing it from accepting any of the data (e.g. buffer full). The host will resend the data to the function endpoint at a later time, which depends on the transfer type of the function endpoint, see Section 5.5.4.

STALL is used by Bulk and Interrupt function endpoints to indicate that the endpoint is halted and the host must not attempt to retry the transmission because there is an error condition on the function.

When the host does not successfully receive the Handshake packet, the host must retry the Handshake packet (only) before retrying the data phase of the transaction.

Note: a device must not respond to a W_XCTA that has a valid device address, but invalid endpoint number. Examples of this include: transactions addressing endpoints before the device is configured and transactions addressing endpoints not defined in the current active configuration.

The host sends zero-length Data OUT transfers by including a *blank* W_{DR}CTA in the MMC. A *blank* W_{DR}CTA is a W_{DR}CTA with no channel time allocation. The receiving device behaves as if an actual zero-length transfer has occurred. The host will schedule a subsequent W_{DT}CTA for the device acknowledgement. No such optimization can be made for Data IN transfers. Devices must send a Data packet with a zero-length data payload.

5.5.1 Isochronous Transactions

Wireless USB Isochronous transactions follow the same basic format and structure as described in Section 5.5, with a few extensions to support the Isochronous data streaming model described in detail in Section 4.11.

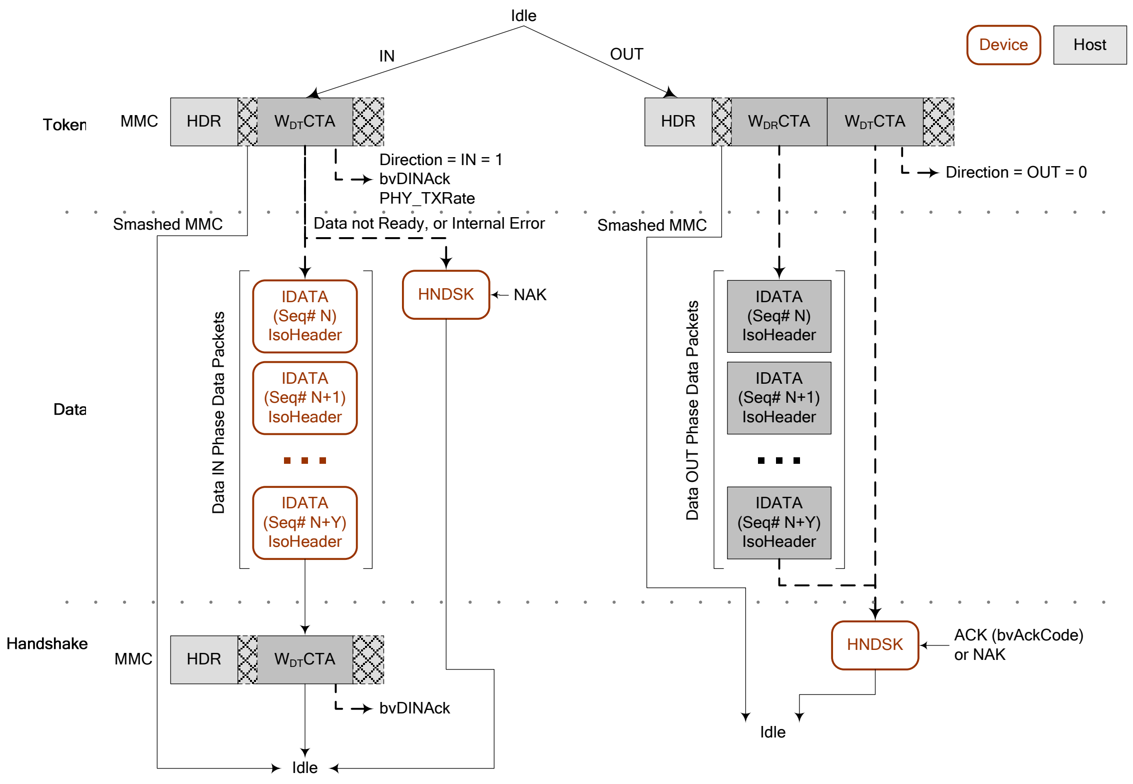

Figure 5-19. Isochronous Wireless USB Transaction Format

There are two structural differences in the data phase between the general transaction format (Figure 5-18) and the Isochronous transaction format (Figure 5-19). Data packets transmitted during the data phase of an isochronous transaction must use the isochronous version of the Wireless USB Header. The *PID* field must have the value of IDATA and must include the additional isochronous data header fields shown in Table 5-2.

5.5.2 Control Transfers

Control transfers in USB 2.0 are comprised of two or three stages of transactions that begin with a SETUP transaction followed by an optional set of transactions for a data stage and the transfer completes with a transaction for a status stage. Wireless USB preserves the same three-stage concept and semantics for control transfers, but introduces some optimizations to make control transfers easier for devices to implement and more efficient within the micro-scheduling transaction format.

The optimization is that the SETUP stage is not a separate OUT transaction. Rather, the Setup command bytes are transmitted in an MMC and are associated with the W_XCTA block that describes the transaction for the next stage of the control transfer. This is fully compatible with the USB 2.0 rule that a device must always accept a SETUP transaction (i.e. cannot NAK). This protocol is simpler because it has removed the opportunity to respond directly to a SETUP transaction. Another optimization is that the Status stage of the control transfer is always encoded as an IN transaction where the function control endpoint must respond with a handshake packet. The final optimization simplifies the data sequencing rules for transactions in the data stage. The start of a control transfer resets the data bursting state on the host and Function endpoint to the default initial state. For a control IN transfer, it means the hosts receive window is initialized to receive packets and this is communicated to the function endpoint in the data stage W_{DT}CTA block via the *bvDINAck* field. For a control

OUT transfer, the setup stage resets the function endpoint's receive window to its initial condition which the host assumes in order to begin transmitting packets for the first transaction in the data stage.

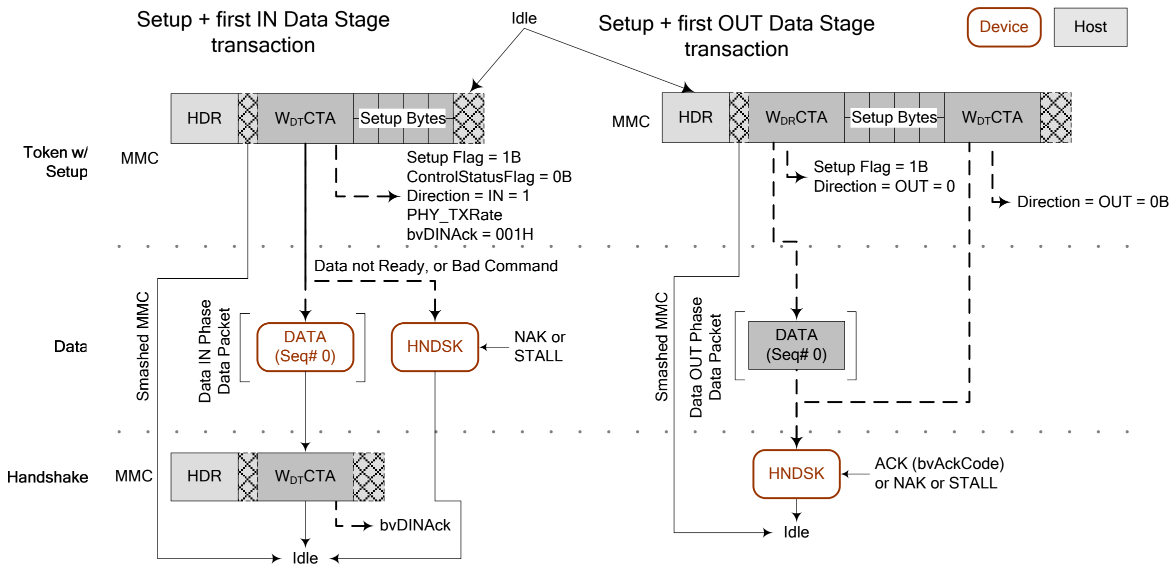

Figure 5-20. Setup Transactions with Data Stage

To start a control read transfer, the host will transmit an MMC with a W_{DT}CTA block describing the channel protocol time slot for the first IN transaction in the data phase of the transfer. The host sets the *Setup Flag* to a 1B in the W_{DT}CTA block to indicate that the eight control request bytes are appended to the channel time description block. It also initializes the *bvDINAck* bit vector to the initial condition of its receive window. If the function control endpoint successfully receives the MMC, it resets its endpoint burst sequence numbers to zero and sets it's transmit window based on the value of *bvDINAck* bit vector field in the W_{DT}CTA block. If the function endpoint has the data requested in setup command ready when the data phase protocol time slot arrives then it will then respond with a data burst, as directed by the W_{DT}CTA block, or should it be unable to respond with data, it returns a Handshake packet encoded with a NAK or STALL handshake code.

To start a control write transfer, the host will transmit an MMC with the same W_XCTA blocks as required for any OUT transaction (see Section 5.5). The host sets the *Setup Flag* to a 1B in the W_{DR}CTA block as above for the control IN transfer and appends the eight control request bytes to the W_{DR}CTA block. During the protocol time slot, the host will transmit a burst of data packets. As with any OUT transaction, the function endpoint will transmit a handshake packet at the handshake phase protocol time slot. Section 5.5 describes the *Handshake Code*s and conditions for each handshake code for an OUT transaction.

In keeping with the USB 2.0 semantics, a STALL handshake has the same effect on Wireless USB Control endpoints. Notably, a function endpoint will return a STALL handshake code if it cannot decode the request setup data. The host will not halt the pipe in response to a STALL handshake.

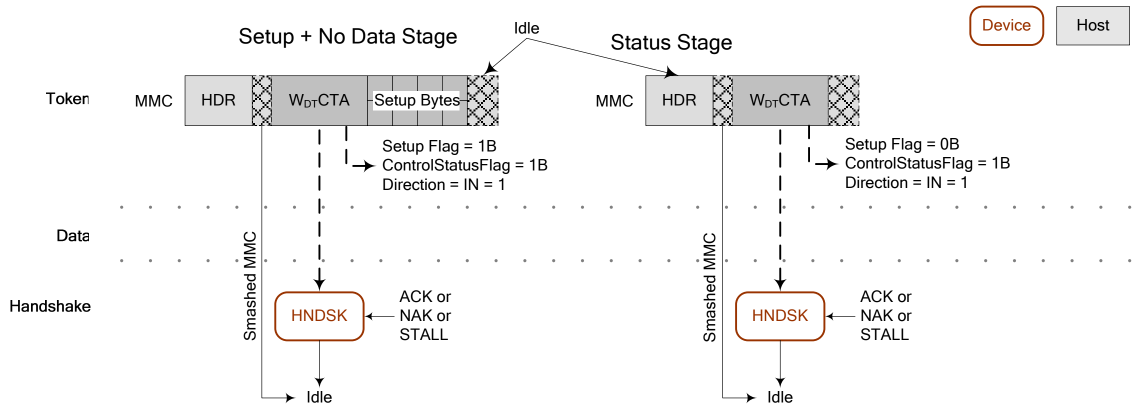

Figure 5-21. Setup w/No Data Stage & Bare Status Stage

To complete a control transfer, the host will transmit an MMC with a W_{DT}CTA block with the *Control Status Flag* and *Direction* fields set to a 1B. These settings tell the function control endpoint that the transaction is a status stage transaction for a control transfer. It must transmit a Handshake packet during the protocol time slot. The device must set the *rWUSBHeader.Endpoint Direction* field to a 1B for a handshake packet transmitted during a control transfer. The *Handshake code* set by the function control endpoint is set to the following:

NAK indicates that the function has not completed the action requested in the setup data bytes.

ACK indicates the control transfer is complete (and possibly the action requested in the setup data bytes is also complete).

STALL indicates that the function has an error that prevents it from completing the command.

Note, for a control read, the host may use the *bvDINAck* field of the status stage W_{DT}CTA to deliver the acknowledgement to last data transaction in the data stage. To start a control transfer that does not have a data stage, the host will transmit an MMC with a W_{DT}CTA block encoded for the status stage.

5.5.3 Device Notifications

Device Notifications are small messages transmitted by a device during a DNTS window, see Section 5.2.1.3 for details on requirements for transmitting the actual message packet. The individual device notification packets are not immediately acknowledged by the host. However it is necessary that device notifications be delivered reliably to the host, which generally means that hosts are required to acknowledge device notifications and devices need to track response and retransmit device notifications as necessary. Figure 5-22 illustrates the general model used in this specification for reliable delivery of device notifications. This general model is used as the base building block for all device notification communications. This discussion is intended to be a reference model that meets required behavior. It is not intended to require implementation of the documented states.

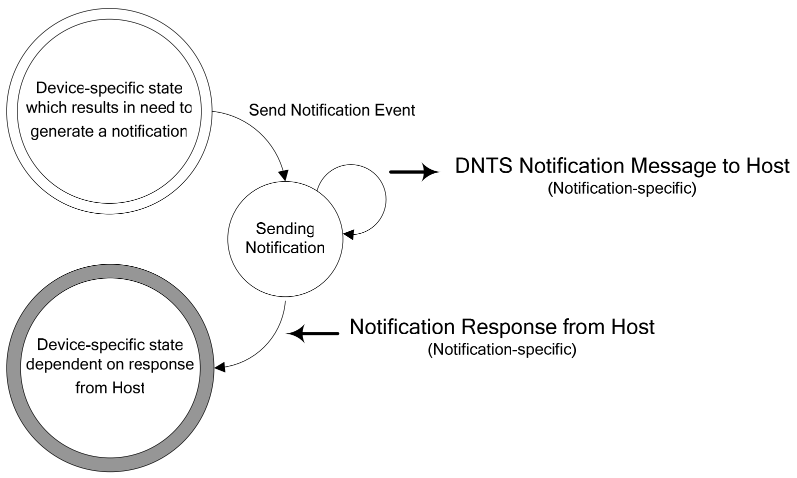
Device-specific state which results in need to generate a notification
Send Notification Event
Sending Notification
DNTS Notification Message to Host
(Notification-specific)
Notification Response from Host
(Notification-specific)
Device-specific state dependent on response from Host

Figure 5-22. General Device Perspective Model for Device Notification Transfers

The **Sending Notification** state is entered by the device (device context) when a system event (desire to connect to a host, endpoints ready, etc.) occurs that requires a device notification transmission. The device remains in this state, sending the specific Notification to the host until it receives the appropriate (for the Notification type) host response. The frequency of Notification retransmits depend on the type of the Notification and either the specification or implementation policies.

The particular response varies by notification message. For example, some notifications require acknowledgement via information elements in the MMC and some simply require transaction activity to the device. Refer to the detailed descriptions of the individual Device Notifications in Section 7.6 for details on required responses.

5.5.4 Flow Control

USB 2.0 has flow control built into the low-level protocol. The wired protocol requires the host to poll for a change in status once a flow control (NAK) response has been given by the device. USB host controller implementations poll aggressively (often) for a change in data stream readiness. This ‘busy-wait’ polling is extremely expensive in terms of occupying available bandwidth in the wireless environment; therefore, Wireless USB utilizes a less bandwidth consuming method for resuming a data stream after a flow-control event. Note that as with USB 2.0, the initial state of all endpoints after any configuration event is that they are assumed to be in the ‘ready’ state.

A device may respond with a flow control response to any token request. An IN Endpoint will return a handshake packet (NAK) instead of a data packet during the protocol time slot. An OUT Endpoint always returns a handshake packet to acknowledge the data packet(s) received during the data stage protocol time slot. There are two fields of particular interest in the OUT handshake packet, the *Handshake Code* and the *bvAckCode*. Table 5-10summarizes the Endpoint responses and the appropriate interpretation.

Table 5-10. Flow control Event Summary

Host	Device Response	Description
IN	DATA(X)	If not end of transfer, host will advance transfer state and begin another transaction when appropriate.
IN	bmStatus.Handshake Code = NAK	Flow control response
OUT DATA(X)	bmStatus.Handshake Code = ACK bvAckCode ≠ 0	More data OK
OUT DATA(X)	bmStatus.Handshake Code = ACK bvAckCode = 0	Flow control response. Device accepted all data transmitted during the data phase time slot, but does not have room for more.
OUT DATA(X)	bmStatus.Handshake Code = NAK	Flow control response. Device did not accept any data transmitted during the data phase. Note, bvAckCode must be a zero.

When a host receives a flow control response from a function endpoint, it will remove the endpoint data stream from the current active list of endpoints being serviced. In other words, the host will stop polling the function endpoint when it gives a flow control response.

When a Control or Bulk device endpoint is ready to resume the data stream (meaning it has data or space of one or more maximum packet sizes available), the device must send an *Endpoints Ready* notification message to the host during a DNTS, see Section 7.6.3.

Endpoints Ready notifications must match the encryption mechanism of the associated endpoint. For example, if a device flow controls a control transfer that is not using secure packet encapsulation, the associated *Endpoints Ready* notification for that endpoint must be transmitted without secure packet encapsulation. A device must not mix unencrypted endpoints with encrypted endpoints in the same *Endpoints Ready* notification.

A device must not use an *Endpoints Ready* notification for Isochronous or Interrupt Endpoints. The host will resume transaction traffic to endpoints of these transfer types at the next scheduled service interval. Note that when the host is to resume transactions with an OUT isochronous function endpoint at the next service interval (after a flow control event), it does not have any specific information about how much buffering is available on the function endpoint. For an isochronous function endpoint that has a burst size larger than one the host has no current information about how much buffering is available. Function endpoints must be prepared for a host to resume transactions at the next interval time via a W_{DT}CTA requesting a handshake packet, or a full OUT transaction where the host only transmits one data packet in the burst data phase. In response, the function endpoint provides a handshake packet, which contains the *bvAckCode* field, indicating to the host how much buffering is available for the next transaction.

5.6 Physical and Media-Access Layer Specific Characteristics

Table 5-11. Wireless USB Protocol Timing Parameters

Parameter	Symbol	MAC/PHY Equiv.	Value	Units
Standard Preamble	$t_{STDPREAMBLE}$	PLCP Std Preamble	9.375	µs
Streaming Preamble	$t_{STREAMPREAMBLE}$	PLCP Burst Preamble	5.625	µs
Maximum interval between MMC packets in a Wireless USB channel	$t_{MAXMMCINTERVAL}$	N/A	65	ms
Maximum Clock Drift	$t_{MAXDRIFT}$	N/A	1.3107 Note 2	µs
Duration of time slot for a maximum sized notification message.	$t_{NOTIFICATIONSLOT}$	N/A	24	µs
Streaming Inter packet Gap	$t_{STREAMIPG}$	MIFS	nominal	µs
Calculated guard time	$t_{GUARDTIME}$	N/A	(2 * MaxDrift) 1 µs (d ≤ 25ms) 2 µs (25ms d ≤ 50ms) 3 µs (d > 50ms) Note 1	µs
Minimum Inter-slot time (successive OUT slots)	$t_{INTERSLOTTIME}$	MIFS	MIFS	µs
Minimum transceiver turn time	$t_{BUSTURNTIME}$	SIFS	SIFS	µs
Minimum Inter-slot time (bus turn)	$t_{BUSTURNINTERSLOTTIME}$	SIFS	$t_{BUSTURNTIME} + t_{GUARDTIME}$	µs

Note 1 ‘d’ is the time between the start of the MMC and the start time for the current packet that must be received.

Note 2 calculated based on 20ppm and a maximum interval of 65535 µs.

Table 5-12. UWB PHY Related Parameters

Symbol	Description	Value	Units
PHY_TXRate	This parameter describes the bit transfer rates supported by the PHY. It is specified as a five-bit field with the following encodings: **<u>Value</u>** — **<u>Meaning (Mb/s)</u>** 00000B — 53.3 00001B — 80 00010B — 106.7 00011B — 160 00100B — 200 00101B — 320 00110B — 400 00111B — 480 01000B – 11111B — Reserved	n/a	n/a
Channel Number	The channel number encoding maps to a specific band group, TF Code. See reference [4] for details. For this revision of this specification, only channel numbers for band group one are required. **<u>Channel Number Range</u>** — **<u>(Band Group, TF Code)</u>** 0 – 8 — Reserved 9 – 15 — (1, 1 – 7) 17 – 23 — (2, 1 – 7) 25 – 31 — (3, 1 – 7) 33 – 39 — (4, 1 – 7) 45 – 46 — (5, 5 – 6)	n/a	n/a
SIFS	Short Interframe Spacing. Maximum TX to RX or RX to TX turnaround time allowed.	10	μs
MIFS	Minimum Interframe Spacing. Specifically, the minimum time between successively transmitted packets. For burst-mode transfers, this is the exact required time between successive packet transmissions.	1.875	μs
PHY Base Signaling Rate	The Base Rate or PHY Base Signaling rate is lowest common denominator transmit bit rate defined by the PHY or MAC Layer. The PHY [4] standard defines 53.3 as the base signaling rate.	53.3	Mb/s

Table 5-13. MAC Layer Header Field Settings for Wireless USB Protocol Time Slot Packets

Frame Control			DestAddr	SrcAddr	Sequence Control	Access Information
Bits	Name	Value	Host to Device		0000H [4]	C000H [5]
2:0	Version	000B	Device Address	Host DevAddr		
3	Secure [1]	(0 | 1)				
5:4	ACK Policy	00B	Device to Host			
8:6	Frame Type	011B [2]	Host DevAddr	Device Address		
12:9	Delivery ID [3]	1XXXB				
13	Retry	0B				
15:14	Reserved	0B				

[1] Value of the Secure field depends on the Device State, see Section 7.1 for details.

[2] The value of *Frame Type* for protocol time slots is **Data Frame**.

[3] For packet transmissions during protocol time slots, this field (*XXX*) contains the stream index value assigned by the host to the Wireless USB Channel.

[4] Wireless USB does not use this field, so devices and the host must set this field to the constant value of 0000H.

[5] The MAC Layer requires data packets transmitted during a Private reservation to have the *More Data* bit set to a one (1B).

Table 5-14. MAC Layer Header Field Settings for Wireless USB MMC Packets

Frame Control			DestAddr	SrcAddr	Sequence Control	Access Information
Bits	Name	Value	Host to Device		0000H	8000H
2:0	Version	000B	Broadcast Cluster DevAddr	Host DevAddr		
3	Secure [1]	1				
5:4	ACK Policy	00B				
8:6	Frame Type	001B [2]				
12:9	Frame Subtype [3]	1110B				
13	Retry	0B				
15:14	Reserved	0B				

[1] See Section 7.5 for rules for using secure packets on MMCs.

[2] The value of *Frame Type* for MMC packets is **Control Frame**.

[3] When the *Frame Type* is **Control Frame**, then this field indicates the *Frame Subtype*, which is **Application-specific Control Frame**.

Chapter 6 Wireless USB Security

6.1 Introduction

This chapter provides Wireless USB security-related information. It describes the security inherent in wired USB (USB 2.0). This inherent security establishes a baseline that a wireless version must meet to be successful. This chapter also describes the architecture, protocols, mechanisms, and USB framework extensions needed to meet this baseline.

When considering security solutions, one must keep in mind that no solution is currently or can be proven to be impervious. Security systems are designed not to explicitly stop the attacker, but rather to make the cost of a successful attack far higher than any gain the attacker might realize from the attack. For the sake of brevity, when we say that a particular solution prevents attacks, we mean that the solution meets the objective listed above. The solution is not impervious, but the cost of compromising the solution outweighs the gain to be realized.

6.1.1 Goal of USB Security

Wireless implementations of USB are wire-replacement technologies. The wire actually provides two services typically associated with security. It connects the nodes the owner/user specifically wants connected. It also protects all data in transit from casual observation or malicious modification by external agents. The goal of USB Security is to provide this same level of user-confidence for wirelessly connected USB devices.

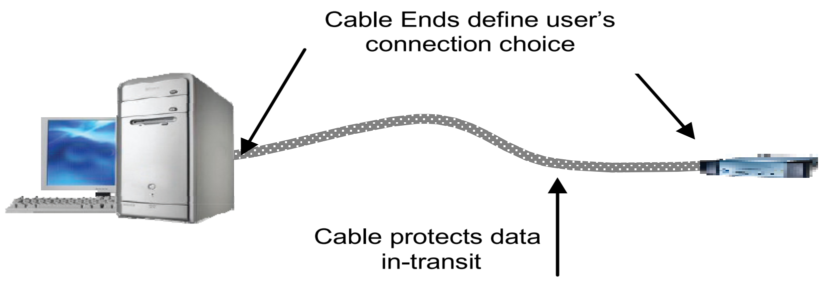
Cable Ends define user's connection choice
Cable protects data in-transit

Figure 6-1: Security provided by USB Cable

6.1.2 Security and USB

The USB core specification does not currently support a notion of security. Applications are free to build security on top of USB, but the USB core specification itself does not currently provide any level of security. A Wireless USB implementation built on the UWB radio requires some level of security. The owner/user's data must be kept private and protected. Likewise, the owner/user's equipment must be protected from unauthorized connections from potentially hostile agents.

Rather than fill this need specifically for a UWB connection, this chapter defines a base-level security architecture for USB in general. Defining the security architecture in general USB terms allows for common USB security managers on the host, regardless of the underlying media that the USB protocol is bound to. Security operations are handled via the USB control channel, allowing this specification to remain media independent.

6.2 Overview

6.2.1 Base of Trust

Any security architecture must begin with a base of trust. A "secure relationship" implies that A) there is a reason for the group to exist, i.e. some common goal, and B) there is some requirement that is used to restrict membership to the group. The members of the group "trust" each other because of 'B' in order to accomplish 'A'. An architecture that stops short of defining 'A' or 'B' is not useful because it only assumes an already functioning relationship. It does not address how the relationship is created in the first place. In common computing jargon, A and B are needed to 'bootstrap' the relationship.

If we examine the function of the USB cable in regards of its ability to establish secure relationships between host and device, we find that A) the purpose of the relationship is to transfer data, and B) membership to the group is restricted to only the nodes that the owner/user has chosen by connecting via the wire. If Wireless USB is a wire-replacement, then clearly Wireless USB Security must provide the same function. Therefore, USB Security defines "Ownership" as the base of trust for USB security. Trust always begins with the user/owner and is propagated to the devices by the owner/user. The USB Security architecture is designed to propagate trust from the user to the user's equipment. The individual nodes can then demonstrate this trust to each other in order to establish the relationships the owner wants.

Different security architectures have different bases of trust that may or may not be applicable to USB. Items that are specifically not applicable are:

- Device/Host characteristics such as manufacturer, model number, firmware revisions.
- External third parties such a Certificate Authorities or Clearinghouses
- Class/Type of device

If a client application requires any of these items for establishing trust, that application can define an application-specific security, implemented on top of the core USB stack.

6.2.2 Preserve the Nature of the USB Device Model

USB Security cannot be allowed to affect the fundamental nature of the USB device model. Implementation must not add significant cost or complexity to a device. It must provide a path for device implementation solely in hardware as USB does not currently require a device to have a processor. If cost or complexity must be introduced, they should be confined to the host.

USB Security must preserve the basic asymmetric communications model of USB. Communications are always, with the exception of signal-replacement messages (connect, disconnect, etc) initiated by the host. This holds true for the security architecture as well. Security operations are always initiated and driven by the host.

The connection policies of USB Security must provide for symmetric authentication between host and device. That is, the device must be given the opportunity to validate the host as the host is given to validate the device. It is just as important to insure that the owner's device connect to the right host as it is to insure that the owner's host connect to the right device. This is especially important as some usage models may have two owners involved, the owner of the host and a separate owner of the device.

6.2.3 Implementation of Security Extensions

USB already provides a "device control plane" via the USB framework. This framework was created for performing enumeration and control operations and as such is the ideal method of implementing security operations. Descriptors, Features, and Requests are all used to create the USB Security Framework.

USB Security requests always target the device as the request recipient. USB Security is device-wide and does not make a distinction between endpoints, just as the cable does not.

USB Descriptors are used to enumerate the security capabilities of a device to a host. The device always describes its capabilities to the host. The host then chooses the device capabilities to use. This simplifies device construction and is in keeping with the USB device model.

Descriptors are used to wrap common security objects such as encryption keys. This allows common key exchange mechanisms to be used regardless of key or encryption type.

USB features are defined to represent security-related control elements present in devices. The USB framework is used to manipulate these features in the standard USB manner.

USB Requests are defined for the security-related command elements. These requests are used for key management, challenge-response verification, and encryption control.

6.2.4 Encryption Methods

The standard method of encryption for the first generation of Wireless USB is AES-128 Counter with CBC-Mac (CCM). This is a symmetric encryption algorithm that uses the AES block cipher to create a robust stream cipher that can be used to provide integrity, encryption, or both. It is capable of real-time operation when implemented in hardware. This is the only method currently defined for general session encryption.

Wireless USB also supports public key encryption, but only for authentication. Devices may choose to start a first-time authentication with public key encryption. In this case, PK encryption is used to authenticate the device and to protect the distribution of the initial CCM key. When PK is used, it will be used in a manner that will allow for software implementations of the algorithms.

The CCM encryption suite provides 128 bits of security for run-time operation. The PK cryptography suite must provide the same level of strength or else the strength of the entire suite is compromised. For this reason, Wireless USB will use RSA with 3072 bit keys for encryption and SHA-256 for hashing.

The Security Architecture also recognizes a wired connection as an encryption method. This allows the SME to recognize a wired connection as a secure connection, without resorting to additional cryptography. This allows for wired/wireless devices, where the wired connection can be used for initial CCM Connection Key distribution.

6.2.5 Message Format

Encryption will generally cause the message length to grow. In addition to the original message, the encrypted message must now contain additional keying material, freshness values, and an integrity value. The exact nature of these additional message components is dependent on the type of encryption used. In general, any new additional material added to the message, other than the integrity value, will be added as a header. This header will immediately precede the encrypted message. The integrity value will immediately follow the message.

6.2.6 Encryption Keys

This section describes the various keys identified for USB Security. In general, the keys fall into two classes of keys, master keys and session keys.

6.2.6.1 Master Keys

Master keys refer to keys that are generally long-lived. Master keys are typically used as the shared secret for authentication. They are also used to derive or protect distribution of session keys

6.2.6.1.1 PK Association Key

This is really a Public Key encryption key-pair. It is optionally used by devices for establishing new connections. The PK pair can be used if the device has no other means for accepting distribution of the CC. The host and device exchange their public keys. The owner must validate that the correct public keys have been received to insure that the correct connection has been made. The host then uses the device public key to protect the distribution of the CC.

If implemented, this key-pair must reside in device memory at the time of the first connection. This key-pair must also be unique to the device.

If the device creates the key-pair dynamically, it must do so in a manner that does not greatly increase device initialization time. The current goal is that a device be capable of transitioning from power-up to fully-connected in 3 seconds or less. Exceeding this time seriously impacts user satisfaction.

6.2.6.1.2 Symmetric Association Key

The symmetric association key is another optional key used for establishing new connections. In this case, the key is a symmetric 128-bit CCM key that exists in device memory at the time of the new connection. This key is transferred to the host via the owner. This could be accomplished via installation software, data entry, or any suitable OOB channel means. This key is only used to protect the distribution of the CC.

6.2.6.1.3 Connection Key

The Connection key (CK) is the primary key used for establishing connections (see 6.2.10 below for details regarding the connection model). This key is created by the host and distributed to the device, along with a corresponding CHID and a CDID at the time of first connection or using an out-of-band method. The key is a 128-bit CCM key. The host should update this key periodically as this will only increase security robustness. The host is expressly forbidden from distributing one CK to multiple devices, unless the CK is the Diagnostic CK and the key is being distributed for diagnostic purposes. Each device must be given a unique CK. Wireless USB uses the CK for authentication and for derivation of the initial session key, the PTK.

Possession of a CK and the CHID/CDID pair implies that the possessor has the owner's trust, allowing the host and device to connect or reconnect without owner intervention.

6.2.6.2 Session Keys

Session keys are short-lived keys, typically used for operational encryption and decryption. These keys are created when a connection is established and discarded when the connection ends.

6.2.6.2.1 Pair-wise Temporal Key (PTK)

PTKs are the "working" keys for USB data encryption. These keys are derived during a 4 way handshake. The host maintains a separate PTK for every device connected. The host uses this key to encrypt all data packets sent to the corresponding device and to decrypt all packets received from the device.

The device uses this key to decrypt all data packets received from the host and to encrypt all packets sent to the host.

6.2.6.2.2 Group Temporal Key (GTK)

The GTK is a specialized temporary key that is shared by all members of the current USB cluster. It allows the host to send secured messages in a broadcast manner, such as an MMC. These messages are not encrypted, but they still require the addition of a MIC.

Devices may not use the GTK for encryption. Only the host can transmit messages secured with the GTK.

6.2.6.2.3 Names for Session Keys

The underlying MAC layer uses names for both PTKs and GTKs. This name is referred to as the Temporal Key ID (TKID) and is present in every secured packet sent between host and device. The TKID identifes the key used to encrypt the secured packet. The host is responsible for creating TKID values and supplying these to devices at the time of key derivation or key distribution.

6.2.6.2.4 Key Confirmation Key

The Key Confirmation Key (KCK) is a short-lived key used for message integrity during authentication. It is a 128-bit key, derived and used during authentication. It is discarded upon completion of authentication.

6.2.7 Correct key determination

Each encrypted message will have a TKID in the unencrypted header portion of the message. This TKID tells the device which key to use when decrypting the message.

6.2.8 Replay Prevention

The PTK and GTK both require a replay prevention mechanism to satisfy CCM requirements. This mechanism uses 3 components, a Secure Frame Counter (SFC), a Secure Frame Number (SFN) and a replay counter (RC). The host and Device each maintain a separate SFC for each session key they use to encrypt transmitted packets. They also maintain a separate RC for each session key they use to decrypt received packets. The counters are initialized to zero when a new key is installed. The SFN is a latched image of the SFC used to encrypt the packet. This image is included in the encrypted packet.

When a packet is encrypted for transmission, the transmitter first increments the SFC associated with the encryption key. The transmitter then copies the incremented SFC value to the SFN field of the message and encrypts the message. If retries are required, the message is re-encrypted before the retrying. Each successive retry will have a SFN value greater than the last attempt.

When a packet is received, the receiver compares packet SFN value with the value of the RC associated with the decryption key. If the SFN value is less than or equal to the value of RC, then the packet is declared to be a replay of a previous packet and is discarded. If the value of SFN is greater than the value of RC, the receiver will set RC to be equal to the SFN of the received packet. In an actual implementation, RC should only be updated after the packet has passed FCS validation and all security checks.

6.2.9 Secure Packet Reception

The processing of received secure packets is as follows:

- Validate FCS – this step is performed prior to validating security elements.
- Validate the received frame has the correct secure frame settings as per MAC Layer rules.
- Validate the MIC for the received frame.
- Check for replay attacks.
- Update RC only if the previous checks have all passed.

Received packets that fail security checks are discarded. The receiver should record the fact that an error occurred.

6.2.10 General Connection Model

The intent of the Wireless USB connection model is to mimic the wired connection model as closely as possible. Wireless devices will have to be "installed" on a computer just as wired USB devices do. This section presents an overview of the connection model without dwelling on the security-related portions of the process. For these examples, in order to provide clarity to the overall connection model, we assume that security is effectively disabled.

6.2.10.1 Connection Context

In order to make secure relationships consistent across multiple connections, some amount of context must be maintained by both device and host. For the case of Wireless USB, this context consists of three pieces of information, a unique host ID (CHID), a unique device ID (CDID), and a symmetric key(CK) that is shared by both parties. The symmetric key is referred to in this document as the *connection key*. This key is used to re-establish the connection at a later date. This key is always unique. The host never gives the same connection key to multiple devices.

Table 6-1: Elements of a Connection Context

Name	Size	Description
Connection Host ID (CHID)	128 bits	Unique Host ID. The device can use this ID to locate the host.
Connection Device ID (CDID)	128 bits	Unique Device ID. This ID uniquely identifies the device to the host specified by CHID. It is not guaranteed to be unique across multiple hosts.
Connection Key (CK)	128 bits	The key to be used to establish connections using this context. This key should be changed on a regular basis, preferably on every use.

A Connection Context (CC) must contain non-zero CHID and CDID values to be useable by the device. A host can use a CC with a value of zero in either field to revoke an existing context. When loading a context for connection purposes, if a device discovers a CC that contains CHID or CDID values of zero, it shall treat that CC as if it were entirely blank. The device shall make no use of the other fields.

Devices may find ways to add value by supporting multiple CCs. Each CC supported by the device must contain a unique CHID. In the case a device supports multiple CCs, only the CC used to connect to the host shall be made accessible to that host. If the device supports multiple CCs and establishes a first-time connection, the device shall make a blank CC accessible to the host for the purposes of allowing the host to establish an initial CC on that device.

Devices that only support a single CC shall always overwrite the current CC with a newly received CC.

Devices shall only accept CCs via secure channels. A device will only accept a CC delivered via a secure USB pipe or an out-of-band channel.

6.2.10.2 Connection Lifetime

The security of any given connection is based on the temporal keys being used to secure that connection. Wireless USB temporal keys have a lifetime of 2^{48} messages, based on the size of the replay counters. In addition, if the host loses communications with the device for *TrustTimeout* or more, the host must assume the device's temporal key has been compromised. At a minimum, a 4-way handshake is required to refresh the temporal keys before the regular traffic is resumed. When a device loses communications with the host for *TrustTimeout* or more, it must send a reconnect device notification to inform the host that the device wishes to resume operation, re-authenticate the host, and generate new temporal keys. A device that experiences a *TrustTimeout* shall only accept requests to its default endpoint until trust is reestablished. During this period, the device shall NAK requests to other endpoints.

6.2.10.3 New Connection

When a new connection is made, the device's connection context is assumed to be blank. The host may already have several contexts representing other devices already installed. It will create a new context for devices being installed for a first time.

This new context may be transferred to the device in one of two ways: either through an out-of-band (OOB) channel, or via secured USB commands. Devices that receive a CC via an OOB method do not make new connections. Since they are given CCs before attempting to connect, they follow the connection logic described below in 6.2.10.4.

A device with no valid CC will have a zero CDID value. Since a CDID is required for connection, the device must manufacture a temporary CDID value to use for this first-time connection request. The CDID value is needed to allow the host to differentiate between multiple requesting devices at the same unconnected device address. A device shall create this temporary CDID value from a cryptographic random number source. This requirement does not apply to devices that receive CCs via OOB methods.

Upon user initiation of a new connection, the device will seek a host advertising for new connections. When the device locates such a host, it will initiate the connection request with the host.

When the host receives the device request for a connection, it creates a new context for the device. This context consists of CHID, CDID, and CK. This context is transferred to the device using the Framework.

A device shall generate a new CDID value for every user invocation of a new connection. Once a device has created a temporary CDID, it shall retain the CDID until it receives a new CDID as part of a CC from its host or the current connection attempt fails.

Because the temporary CDID is a random number, there is a very small chance that two devices will attempt to make a new connection to the same host using the same CDID. This also means that there is a very small chance that the host will connect to the wrong device. The protocols used for establishing connections between host and device contain protections against this and will fail to connect. This error-checking combined with the requirement that a device create a new temporary CDID after a failure provides the means for recovering from this error.

6.2.10.4 Connection

A device may make a connection with any host that it has a connection context for. To make a connection, the device first has to locate the host advertisement for the known host. This is done by locating a host broadcasting a CHID that matches a Connection Context CHID known by the device. Once located, the device makes a connection request, supplying the corresponding Connection Context CDID.

The host, upon receiving the connection request uses CDID to locate the corresponding context for the device. If a context is found, the host completes the connection by issuing a Connect Acknowledge to the device. As part of the Connect Acknowledge, the host will also move the device to a unique un-authenticated device address.

At this point, both host and device perform mutual authentication and key derivation using a 4-way handshake and the CK. Upon successful completion of the 4-way handshake, the host will enable the device for operation.

6.2.10.5 Reconnection

When a device loses communications with the host for a period of time greater than *TrustTimeout*, it must inform the host that it no longer trusts the security of the connection. It does this by sending a connect request to the host, specifying its CDID and its previous device address. This request informs the host that the device wishes to re-authenticate the connection without losing its current device state and configuration. The host may choose to either allow the device to reconnect while retaining device state or it may process the request as a connect request, instructing the device to reset any remembered state.

6.2.10.6 Revocation

When we make provisions for creating secure relationships, we must also make provisions for deleting them. Revocation of a connection by either party can be accomplished by deleting the unique host ID from the device or by deleting the unique device ID from the host. The host can delete the context via framework commands. How the device deletes a context without host intervention is outside the scope of this document, however, the device should provide the user with an ability to return to “factory default settings”. This action should delete any contexts currently held by the device.

6.2.10.7 One-time Connections

In some use cases, hosts will not wish to establish permanent connections. Some use cases will want to establish a one-time connection where a CC is delivered to the device to allow a connection to occur, but the CC is not retained for future connections. The host can accomplish this by delivering a CC to the device, then revoking the CC immediately after the 4-way handshake is completed.

6.2.10.8 Diagnostic Support

Support for diagnostic equipment is important for the success of a new technology. Wireless USB recognizes this need and accommodates it by leaving the common trigger elements in clear-text in transmitted messages. These components are still authenticated by the integrity value. Only the data payload portion of Wireless USB transactions will be encrypted. Device Notifications are authenticated only and contain no encrypted data.

Specific security-defeat mechanisms will not be added to provide diagnostic access to encrypted payloads. . These types of mechanism tend to be more useful to the attacker than the developer. What is really required is a way to provide the diagnostic tool with the keys used by the host and device. This will typically be done on a channel other than Wireless USB, so this specification will not define the mechanism to be used. Instead, this chapter offers three suggestions of dealing with encryption when using diagnostic tools.

- Test in unauthenticated mode. This mode supports a limited subset of framework requests and test modes that can be used to test basic operation
- Make the "Connection Key" available to the diagnostic software. If a 4-way handshake sequence is captured, then the diagnostic tool has all the material needed to perform the same key derivation logic as the host and device did. To aid in this, this specification defines the CK value of 0 to be the Wireless USB Diagnostic Connection Key.
- Provide a means of moving the current temporal keys from the host to the diagnostic device.

6.2.10.9 Mutual Authentication

As noted in the introduction, authentication must be symmetrical between the host and device. USB Security accomplishes this by using a 4-way handshake that allows the host and device to prove to each other they hold matching CKs. The 4-way handshake also allows both parties to derive initial session keys from the CK without directly using the CK to encrypt transmitted messages.

6.2.10.9.1 4-Way Handshake

The 4-way handshake combines mutual authentication and temporal key distribution into a single 4-message protocol. Temporal keys are derived in a manner that does not expose the CK, thereby eliminating the need for a unique nonce for every use of the CK.

The host initiates the 4-way handshake by sending a TKID and a 128 random nonce *HNonce* to the device. The device also creates a random nonce *DNonce*. The device then provides it's *DevAddr*, the host's *DevAddr*, and the two nonces to a key generation function. This key generation function derives an initial session data key. The key generation function also derives the Key Confirmation Key (KCK), which is used during authentication and then discarded.

The host initiates Phase 2 of the 4-way handshake by asking the device for *DNonce*. The device returns the TKID and *DNonce*. It also adds a third item to the returned data. It uses KCK to compute a message integrity code (MIC) over the TKID and *DNonce*.

When the host receives the *DNonce* message, it can also derive the initial session data key. After deriving the key, it validates the MIC received from the device by performing the same MIC computation with KCK. If the MICs do not match, the host halts further processing and silently disconnects the device.

At the end of Phase 2, the host and device have both derived initial session keys and the host has proof that the device holds the correct CK. Phase 3 is used to provide proof to the device that the host also holds the correct CK and to instruct the device to install the derived key. The host initiates Phase 3 by installing the derived session key and constructing a message containing the TKID, *HNonce*, and a MIC computed over the previous fields with KCK. It sends this message to the device. The device validates the received MIC by also performing the MIC computation. If the MICs do not match, the device silently disconnects. If the MICs match, the device installs the derived session key.

Phase 4 of the 4-way handshake is used to tell the host that the device has successfully installed the session keys. Wireless USB devices perform phase 4 by successfully completing the status phase of the Handshake3 control request. The derived PTK is installed by the device and available for use immediately following the successful completion of the Handshake3 control request.

6.2.11 Key Management

This section describes the general key management philosophy of USB. In general, hosts are responsible for key management operations. Hosts track the life of session keys and are responsible for creating and distributing replacement keys, or causing replacement keys to be derived. Devices do not request new keys. If a device becomes unsynchronized with respect to the current session key, the device must send a Reconnection request.

Session keys are never transferred from device to host via the USB Framework. Distribution of a session key is a one-way function. Once in possession of a session key, the device never divulges that key.

Get Key operations are restricted to public keys. If needed, a host will ask a device to divulge its public key.

6.2.11.1 PTK Management

The PTK is derived during the 4-way handshake and typically does not change during the life of the connection unless a *TrustTimeout* occurs on either the host or device. Under extreme circumstances, the key can be consumed if it is used to encrypt 2^{48} packets. In this case, the host must perform an additional 4-way handshake with the device in order to derive a new PTK before the old PTK expires and the SFC associated with that key rolls over.

6.2.11.2 GTK Management

The GTK is shared among all devices. Because it is shared, it must be changed whenever a device leaves the current group. Problems with detecting device departure in a wireless environment may cause hosts to choose to update GTKs at a fixed periodic interval.

Distribution of the GTK presents special problems with key synchronization between the host and the devices. The key distribution mechanism provides that the device will be ready to use the new key at the completion of the distribution request. However, the host must re-key all the devices in the current cluster. A device can take several MMC periods to respond to a key distribution, so synchronizing a GTK change among devices is an almost impossible task.

To simplify this task, USB requires that devices be capable of holding two current GTKs. Session keys have a 7-bit index value. The host will use this value to distribute GTKs in numerical order. The device provides a table capable of holding two keys. The low bit of the session key index is used as the table index. This allows Key_2 to replace Key_0, Key_3 to replace Key_1, etc.

When the initial connection is established, the host distributes key K_0. When the host detects that a device has left the group, it distributes K_1 to all the devices in the group. When the last device responds, the host installs and begins using the new GTK. On the next departure, the host distributes K_2. Devices replace K_0 with K_2 as they receive it. The host continues to use K_1 until K_2 is installed on the last device. If another device departs before K_2 can be distributed to all devices, a host should abandon distribution of K_2, skip distribution of K_3, and begin distributing K_4.

A device should not discard an older GTK until the host begins to use the new GTK. Once the host uses the new GTK, it must not use the older GTK.

6.3 Association and Authentication

When a Wireless USB connection is made, the host and device must complete association and authentication phases before the connection is considered operational. As the device progresses through the phases, additional host components are made aware of the device. Upon completion of the phases, the device is presented to host USB stack and standard USB device enumeration begins. In short, this means that device and host must fully authenticate each other before the USB stack is made aware of the device.

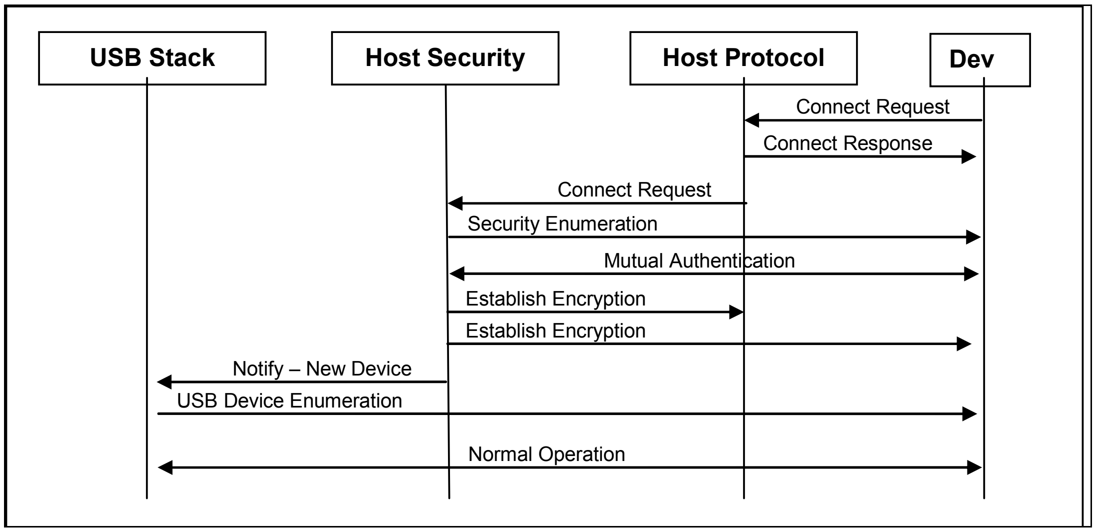

Figure 6-2: Association Phases

The Association phase of a Wireless USB connection deals with three separate problems that must be addressed in order to create a secure connection. The first problem is establishing the initial connection. In the wired-USB world, this happens because a physical connection is made by the user. The host and device are informed of the other's presence via circuit completion and electrical signaling. In the wireless world, we can't use circuit completion so we must compensate by adding additional protocol messages to allow a device to request a connection with a host.

The second problem that must be addressed is that of verification of the other party. In some manner or form, each party involved in any connection must demonstrate to the other party that they have the "owner's trust" and can therefore be connected.

The third problem is derivation of the initial PTK. Once the initial PTK has been derived and the GTK has been distributed, the device can be considered operational. It can then be presented to the main host USB Stack for the standard USB device enumeration sequence.

6.3.1 Connection and Reconnection Requests

A connection request is always initiated by a device and received by a host. The actual format of the Connection request is covered in Section 7.6.1. Of interest to security are the unique device identifier and one bit of information contained in the request. This bit is the NEW indicator. The state of these bits determines the type of connection request being made.

Table 6-2: Connection Request Types

NEW	Description
0	**Connection Request** – The device has previously been connected to this host. Verification will be performed with the CK.
1	**New Connection Request** – The device has not previously been connected to this host. The host will enumerate the security capabilities of the device to determine the appropriate authentication procedure. The host will distribute a CC to the device.

Not all connection types are allowed at all times. Typically, a host will always allow reconnection requests but only conditionally allow first-time connection requests. This allows automatic recovery from host reboot, loss and recovery of channel, roaming, etc.

Some user-initiated event must be required to allow a host to accept new connections. The user is the only entity in the user-host-device relationship who knows when a new device is present in the environment. Since user validation of new devices is required anyway, involving the user at this stage serves as a pre-validation step. This requirement also serves to prevent hosts from interfering with each other's ability to connect to a device.

A reconnection request is always initiated by a device and received by a host. The reconnection request allows a device to request that it be allowed to resume operation at its previous USB device address, in its previous device state. This allows the "trust" surrounding a connection to be regenerated without having to fully reset and re-enumerate the device.

6.3.2 Authentication

The purpose of the Authentication phase is to allow the device and host to prove to each other that they have the owner's trust. There are several means of doing this, but the essence of most of the methods is that both parties prove to each other that they know a common secret without revealing the actual secret.

New connections require some form of user-interaction as one of the parties to be connected will not yet have a Connection key. This means that Owner trust has not yet been conveyed to that entity. Just how the user is involved depends on the capabilities of the device. The different types of interactions are covered in detail in the sections below.

6.3.2.1 Authentication Related Device Capabilities

USB Security provides multiple means of authentication, based on the capabilities of a device. The host enumerates these capabilities from the device. It then uses the returned capabilities to determine which type of Authentication process to use. Different types are provided to allow the best method to be selected for a specific device type. Different methods make different requirements on devices, so one method may be easily accomplished on one existing device architecture, while adding significant cost/complexity to another. For instance, a printer may already contain a CPU sufficient for PK methods, whereas a specific ASIC state machine would require much additional complexity.

6.3.2.1.1 Device has Out-Of-Band Channel for Key Distribution

Devices can provide an out-of-band channel for distribution of a Connection key. Examples of such out-of-band channels might be wired USB connection, memory card, user-interface, etc. Devices that support this mode of key distribution never have to establish new connections. A connection context is transferred to the device via the OOB channel before the first wireless connection is ever made. Since the device was pre-loaded with a connection context, it can use connection requests to establish connections with the host.

6.3.2.1.2 Device has Symmetrical Association Key

Devices can have a fixed symmetrical association key. This is typically a hardwired key, set during manufacturing. This key must be unique to the device. The user is required to transfer this key from the device to the host in order to establish a new connection. This may be transferred by data entry or by OOB channel means.

6.3.2.1.3 Device has PK Association Keys

If devices wish, they can support Public Key encryption. In this case, the device must contain a unique key-pair. In this case, the user will not be required to transfer a key, but they will be required to participate in validating that host and device have received the correct public key from each other. This insures that the correct devices have been connected.

6.3.2.2 Ceremonies

A 'Ceremony' refers to the interactions between the entities involved in a secure relationship, specifically the interactions involved in establishing the relationship. Ceremony diagrams are very similar to network protocol diagrams. However, instead of requiring that protocol participants are network nodes, they encompass the user and possibly the environment. The following sections present the Association ceremonies for association methods listed. Note that many equivalent ceremonies can be constructed by simple reordering of ceremony steps. Actual implementations of the ceremonies presented here may use equivalent ceremonies if sufficient benefit exists for reordering.

6.3.2.2.1 Association Ceremony for Out of Band Channel Key Distribution

This ceremony is used with devices that have some means other than Wireless USB for receiving a connection context. Regardless of how this OOB channel is implemented, some type of user action will be required to either initiate or complete distribution of the connection context to the device. The user may either directly enter the context information or physically shuttle a context container. Either way, the user's involvement is required. To be useful for transferring secrets, the user must be able to control access to the OOB channel. This means that the OOB channel used is not susceptible to the same attacks as WUSB.

The Ceremony diagram shows that the ceremony first starts with a CC distribution between the user and the host. Presumably, a distribution from user to host only happens once, when the host is first initialized. If the CC is to be distributed electronically, the user will accept an electronic copy of the CC from the host (on appropriate media).

The next step is the user distribution of the CC to the device. The user transfers the CC to the device via the OOB channel. If the OOB channel is a wire or cable, then the user will not physically transfer the CC. The user will however, be required to instruct the host to distribute the CC to the device.

Once the device has received a valid CC, it uses connect protocol to connect to the host. This protocol is a 4-way handshake sequence using the CK. Successful completion results in the derivation of the PTK, distribution of the GTK, and the announcement of device arrival to the USB sub-system.

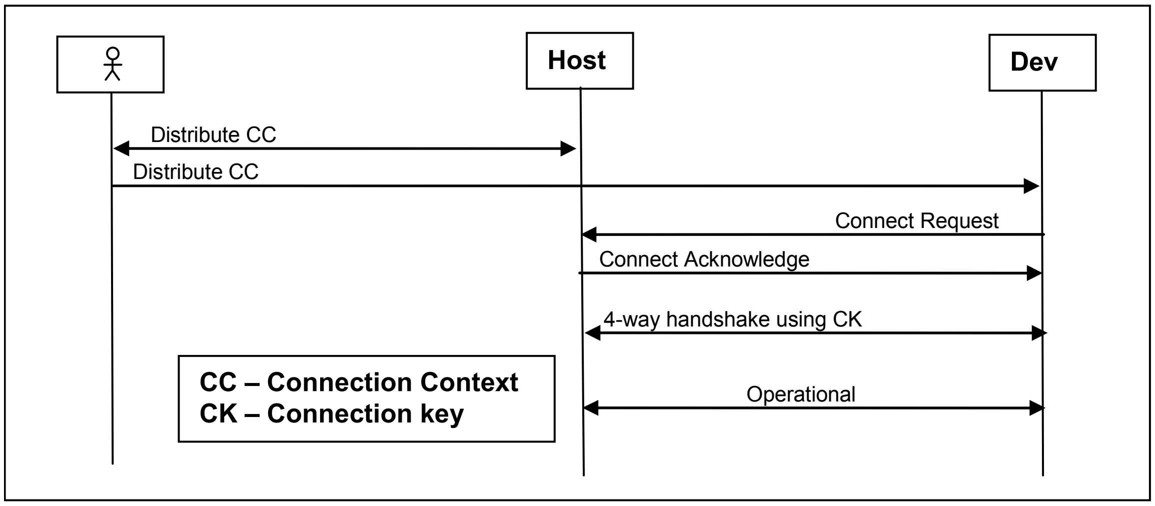

Figure 6-3 Ceremony for Out of Band Channel Context Distribution

6.3.2.2.2 Ceremony for Association using a Fixed Symmetric Key

This ceremony is used for devices that contain a "static" symmetric key for association purposes. This key must uniquely represent the device. This key can be created during manufacturing or in some cases, be created by the device when first powered up. The key must exist when the device attempts to associate.

The ceremony begins with the device distributing the key to the user. This distribution may be electronic or printed. It could be embedded in installation software or it could be printed on the bottom of the device. Regardless of how the distribution is done, the user is involved in transferring the key from the device to the host. This allows the host to demonstrate Owner Trust by its knowledge of the device key. The device likewise demonstrates Owner Trust by its knowledge of the key. It is important to note that we can treat this key as if it were the owner key because the owner/user was involved in transferring the key. By transferring the key, the owner has effectively approved use of the key as a temporary owner key. While transferring the key to the host, the owner should also instruct the host to advertise via its MMCs that new connections are allowed.

Note that in real implementations, the user may not transfer the key to the host until after the device has made its connect request. This allows the host to postpone seeing the key until it knows what the key purpose is.

The device will see the host allowing new connections and make a connect request with the NEW attribute set. The host completes the request and notifies the SME that a new device needs to be processed. The SME enumerates the device and discovers that a fixed symmetric key is required. It gets this key from user, either now or earlier, and enables encryption.

The host and device next perform a 4-way handshake that allows host and device to both prove to each other that they know the fixed key and to derive a PTK to protect the CC.

After successful completion of the 4-way handshake, the host distributes a CC to the device. It then announces the device to the USB sub-system.

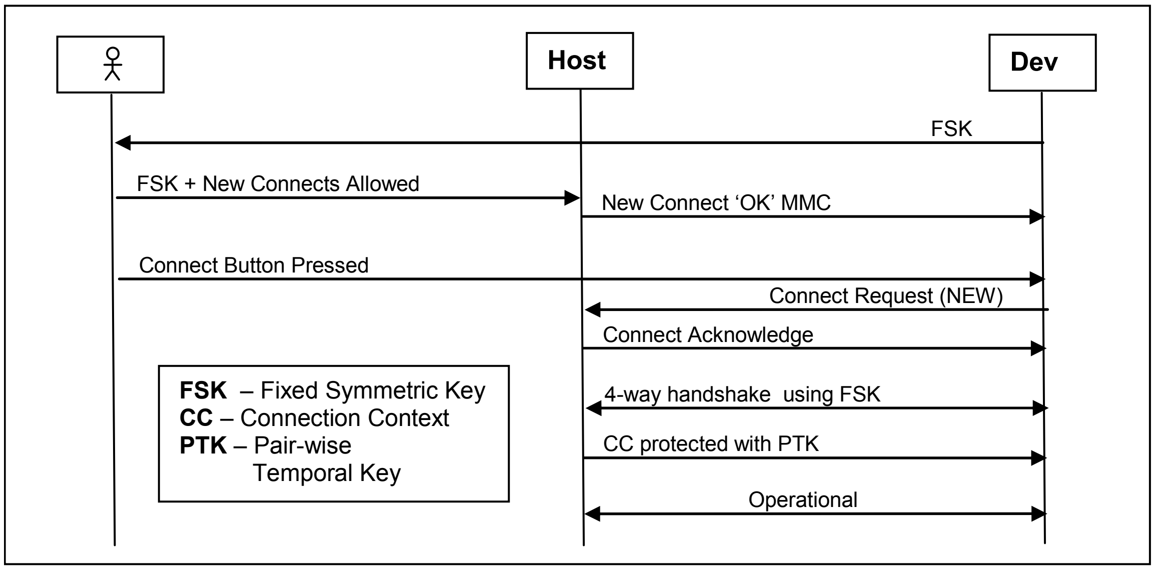

Figure 6-4 Ceremony for Fixed Symmetric Key Association

6.3.2.2.3 Ceremony for Association using Public Key Cryptography

This ceremony is used for devices that contain a PK encryption key-pair for association purposes. This key-pair must uniquely represent the device. They can be created during manufacturing or in some cases, be created by the device when first powered up. They must exist when the device attempts to associate.

The ceremony begins with the user instructing the host to advertise for and accept new connections. The device will see the host allowing new connections and make a connect request with the NEW attribute set. The host completes the request and notifies the Security Entity that a new device needs to be processed. The SME enumerates the device and discovers that PK encryption is required so it provides the device with the host public key and requests the device public key from the device.

At this point, the owner becomes involved. The owner must validate that both device and host have received the correct public IDs. This validation step is necessary to verify that the host and device are connected to each other and not to malicious agents. This step is also necessary because the user validation of the received keys serves as the transfer of Owner Trust from the owner to the device and host. The owner is instructing the host and device to trust the owners of the received keys.

After successful validation of the device public key, the host uses the device public key to protect and distribute a CC to the device. After distributing the CC, the host performs a 4-way handshake using the newly distributed CC. This allows the host and device to derive a PTK and begin secure operation.

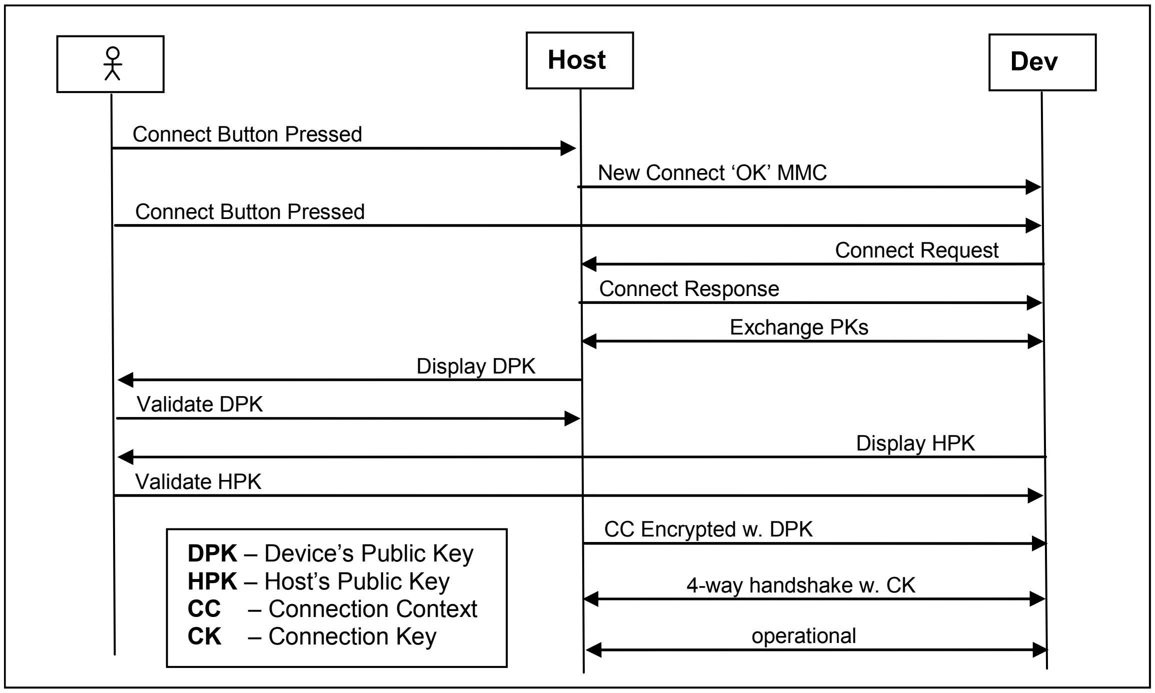

Figure 6-5 Ceremony Public Key Association

6.4 Interfacing to AES-128 CCM

This section provides the details for interfacing with AES-128 CCM. As specified, this protocol requires the message and some additional keying material be formatted into the CCM nonce, Counter-mode blocks, and encryption blocks. The CCM nonce provides uniqueness to each message. The Counter-mode blocks are used to calculate the MIC. The encryption blocks provide the keystream that it used to encrypt the message and the MIC.

6.4.1 CCM nonce Construction

The CCM nonce is constructed from the following MAC-Layer header components: SrcAddr, DestAddr, TKID, and SFN. The format of the nonce is given in Table 6-3. All values are stored in little-endian byte order.

Table 6-3: CCM nonce format

Offset	Field	Size	Description
0	*SFN*	6	The secure frame number value associated with this message.
6	*TKID*	3	The Temporal Key ID value 'names' the key used to encrypt/decrypt this message.
9	*DestAddr*	2	The address of the destination device.
11	*SrcAddr*	2	The address of the source device.

6.4.2 l(m) and l(a) Calculation

CCM encodes two values into the message. These values are l(m), the encrypted data length, and l(a), the additional authenticated data length. These values are calculated from appropriate Encryption Offset (EO) for the message and the length of the message. Note: “length of message” in this context is assumed to be the sum of the length of MAC header, Wireless USB header and data payload. The values are calculated as:

l(a) = EO + 14

l(m) = unencrypted message length – EO – 10

6.4.3 Counter-mode B_x Blocks

For calculation of the MIC, the message is broken into 2 or more counter-mode blocks. The CCM specification refers to these blocks as blocks B_0 – B_n. Table 6-4 gives the format of block B_0. Table 6-5 gives the format of Block B_1.

Table 6-4: Block B_0 format

Offset	Field	Size	Value	Description
0	*Flags*	1	59H	As per CCM specification
1	*CCM Nonce*	13	variable	The CCM Nonce as described above.
14	*MSB_l(m)*	1	variable	The most significant byte of the l(m) value.
15	*LSB_l(m)*	1	variable	The least significant byte of the l(m) value.

Table 6-5: Block B_1 format

Offset	Field	Size	Value	Description
0	*MSB_l(a)*	1	variable	The most significant byte of the l(a) value.
1	*LSB_l(a)*	1	variable	The least significant byte of the l(a) value.
2	*MAC Header*	10	variable	The entire MAC header in transmission order
12	*EO*	2	variable	The Encryption Offset component of the MAC-Layer security extensions to the MAC header.
14	*Security Reserved*	1	Variable	The Security Reserved component of MAC-Layer security extensions to the MAC header.
15	*Padding*	1	0	This byte is only used to pad the block. It is not part of the message and never transmitted.

When EO is non-zero, additional authentication blocks are built from payload bytes (in transmission order) until EO bytes have been consumed. The remainder of the block is padded with zeros as needed. The padding is not transmitted. This forces the start of the encrypted data to be aligned with authentication blocks, allowing for optimization of the encryption and decryption logic.

6.4.4 Encryption A_x Blocks

CCM uses the A_x blocks to generate the keystream that is used to encrypt the MIC and the portion of the message to be encrypted. Counter i is initialized to zero to form block A_0 and incremented for generating successive blocks. Block A_0 is always used to encrypt or decrypt the MIC. Additional A_x blocks are generated as needed for encryption or decryption of the payload.

Table 6-6: Block A_x format

Offset	Field	Size	Value	Description
0	*Flags*	1	01H	As per CCM specification
1	*CCM Nonce*	13	variable	The CCM Nonce as described above.
14	*MSB Counter i*	1	variable	The most significant byte of *Counter i*.
15	*LSB_Counter i*	1	variable	The least significant byte of *Counter i*.

6.5 Pseudo-Random Function Definition

This chapter makes use of cryptographic random numbers in several locations. The Pseudo-Random Function (PRF) definition provides these numbers. The function is also used for key derivation during the 4-way handshake and for calculating MICs to protect the 4-way handshake messages. As used in this chapter, 3 output sizes are needed: 64 bits, 128 bits, and 256 bits. We therefore define 3 versions of the PRF function.

- PRF-64, which outputs 64 bits,
- PRF-128, which outputs 128 bits, and
- PRF-256, which outputs 256 bits.

In the following, *K* denotes a 128-bit symmetric key, *N* denotes the CCM nonce, *A* denotes a unique 14-byte ASCII text label for each different use of PRF, *B* denotes the seed data stream to be processed, and *Blen* specifies the length of this data stream.

The following pseudo-function demonstrates how PRF interfaces to the CCM logic. This function takes a data string and returns the encrypted CBC Counter-mode MIC.

```
CCM-MAC-FUNCTION(K, N, A, B, Blen)

        Format Block B0 from l(m) = 0, N, and flags = 0x59
        Format Block B1 from A and l(a)  = Blen + 14
        Format additional blocks from B as specified by Blen
         (note, last block is padded with values of zero as needed)
        Format block A0 from flags = 0x01, N and Counter = 0
        The Bx blocks are processed in CBC-Counter mode to generate the MIC value for the
        blocks. The MIC is then encrypted with the keystream generated with the A0 block.

        return encrypted MIC
```

The next pseudo-function provides the logic for PRF itself. The process concatenates MIC values to create a value of the requested length.

```
PRF(K, N, A, B, Blen, Len)
       result = empty
       for (i = 0;  i < (Len + 63)/64; i++,  N.SFN++)
              result = result contatenated with CCM-MAC-FUNCTION(K, N, A, B, Blen)
        return result
```

The following definitions provide convenient handles for the 3 sizes of PRF results used in this specification.

```
PRF-64(K, N, A, B, Blen)  = PRF(K, N, A, B, Blen, 64)
PRF-128(K, N, A, B, Blen) = PRF(K, N, A, B, Blen, 128)
PRF-256(K, N, A, B, Blen) = PRF(K, N, A, B, Blen, 256)
```

6.5.1 Key Derivation

Key derivation during the 4-way handshake depends on PRF-256 to generate the actual session keys from the supplied data. Key derivation using PRF-256 requires users to specify the following parameters:

Name	Size (bytes)	Description
HID	2	Current DevAddr of the Host
DID	2	Current DevAddr of device
HNonce	16	Random value selected by Host (from message 1)
DNonce	16	Random value selected by device (from message 2)
TKID	3	Host-supplied key name (from message 1)
MK	16	The master key the PTK should be derived from, either a CK or FSK.

The Key Derivation function creates the PRF-256 parameters from these parameters as follows:

- MK - The master key being used for derivation
- N – B11-12=Host ID, B9-10=device ID, B6-8=TKID, B0-5=zero
- A – “Pair-wise keys”
- B – HNonce || DNonce
- Blen – 32

The Key Derivation function then calls PRF-256 to compute 256- bits of key stream. This key stream is then split to form the initial management and data keys. The least significant 16 bytes of *Key Stream* becomes the KCK while the most significant 16 bytes become the PTK.

KeyStream ← PRF-256(K, N, A, B, Blen)

Key	Name	Source
KCK	<none>	KeyStream[0..15]
PTK	TKID	KeyStream[16..31]

6.5.2 Out-of-band MIC Generation

The 4-way handshake uses out-of-band MIC calculations for handshake phases 2 and 3. PRF-64 is used to provide these OOB MIC calculations. The OOB MIC function creates the PRF-64 parameters as follows:

- K - The KCK from the key derivation process
- N – B11-12=Host ID, B9-10=Device ID ID, B6-8=TKID, B0-5=zero
- A – "out-of-bandMIC"
- B – Message data to be authenticated
- Blen – length in bytes of message data

MIC ← PRF-64(K, N, A, B, Blen)

6.5.3 Example Random Number Generation

In order to implement the cryptographic protocols outlined in this specification, every platform needs to be able to generate cryptographic grade random numbers. RFC 1750 gives a detailed explanation of the notion of cryptographic grade random numbers and provides guidance for collecting suitable randomness. It recommends collecting random samples from multiple sources followed by conditioning with PRF. This method can provide a means for an implementation to create an unpredictable seed for a pseudo-random generation function. The example below shows how to distill such a seed using random samples and PRF-128. In the example below, '||' denotes concatenation.

```
LoopCounter = 0
Nonce = 0
Result = empty
    while (LoopCounter < 32)  {
        result = PRF-128(0, Nonce, "InitRandomSeed",
                 USB Vendor ID || USB Product ID || Time || result || LoopCounter, dataLen)
        Nonce = Nonce  + 1
        result = result || <randomness samples>
    }
    GlobalSeed = PRF-128(0, Nonce, "InitRandomSeed",
                         DevAddr || Time || result || LoopCounter, dataLen)
```

Once the seed has been distilled, it can be used as a key for further random number generation. The 4-way handshake requires each party to supply a 128-bit random number. This number can be generated using the seed and PRF-128

```
GenerateRandomNonce
    N = DevAddr || DevAddr ||6 bytes of  zero
    <<Collect randomness samples>>
    result = PRF-128(Global Seed, N, "Random Numbers",
                     <randomness samples>, length of samples)
    return result
```

Chapter 7 Wireless USB Framework

This chapter describes the common attributes and operations of Wireless USB Device Management. It depends on Chapter 9, “USB Device Framework”, of the USB 2.0 Specification as the baseline, and then describes differences and extensions to the base USB Framework. The chapter starts with a description of a device state machine. This is followed by a description of extensions to standard Framework commands to support the wireless device space, then a description of the Security-specific extensions. This chapter concludes with a description of the additional Descriptors and Information Elements needed to support wireless devices.

7.1 Wireless USB Device States

A device has several possible states. Some states are visible to the Wireless USB host, while others are internal to the device. This section describes the visible states.

The device states envelope the USB device states documented in the USB 2.0 specification as illustrated in Figure 7-1

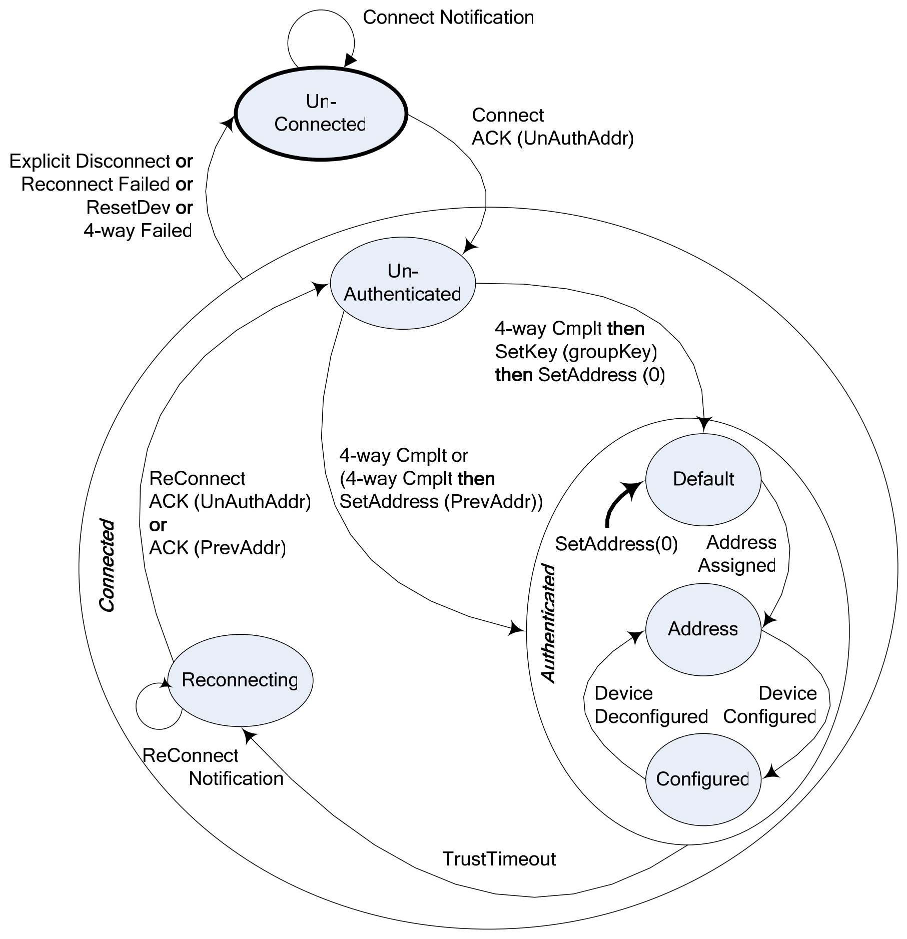

Figure 7-1. Wireless USB Device State Diagram

Because a physical connection does not exist, data communication between a device and host requires that a relationship be established to serve as a logical connection. As noted in previous chapters, a host and device have to make this logical connection secure before the host will use the functions advertised on the device. The model for establishing a connection and securing it is based on the device states illustrated in Figure 7-1. The sections below describe the specific device states and the general events or criteria required to occur for the device to make a state transition.

Devices don't receive power from the host platform which means they must use power from a local source. Therefore, the device state diagram does not include the notion of a "powered" device state.

7.1.1 UnConnected

A device that does not have any established communications with a Wireless USB host is in the **UnConnected** state. A device defaults to this state on power up and can return to this state if:

- The device or host executes an explicit disconnect, or
- A reconnection attempt fails (i.e. host does not acknowledge the encrypted *DN_Connect* notification from the device), or
- The device observes a ResetDevice_IE with a matching CDID element value, or
- A 4-way handshake does not complete successfully. Failures may occur due to a variety of factors, including taking longer than *TrustTimeout* seconds to complete, a STALL response, etc.

While in the **UnConnected** state, the only data communications a device can initiate with a host over its Wireless USB Channel is a connect device notification (see *DN_Connect* in Section 7.6.1) . A device in the **UnConnected** state must have its Wireless USB channel device address set to the *UnConnected_Device_Address*, see Section 4.3.8.5. A device must not use secure packet encapsulation (i.e. SEC bit = 0b) when transmitting *DN_Connect* notifications when in the **UnConnected** device state. A device stays in this state until it has attempted to connect (via a connect device notification) with a specific host on its Wireless USB channel and the host has acknowledged receipt of the connect notification by sending a *Connect Acknowledgement*. When the host responds to a connect notification, the acknowledgement will also assign the device a device address in the Unauthenticated_Device_Address_Range, see Section 4.3.8.5

At this point the device and host have exchanged information, so the two know that data communications are possible, and the device is logically "connected" to the host's Wireless USB channel. The device transitions to the **Connected** general state.

7.1.2 UnAuthenticated

The device entry sub-state within the connected device state is the **UnAuthenticated** device state, where data communications between the host and device are restricted to exchanging authentication messages and other related security information. This information can only be exchanged over the Default Control Pipe and because the device is unauthenticated most of the exchange must be conducted in plain text (i.e. no security encapsulation). Control requests are allowed in this state to authenticate the connection, allow the host to distribute the GTK, and to set the device to a specific USB device address in order to transition it to the **Authenticated** device state. The data communications that are allowed between a host and device from the **UnAuthenticated** state are described in Section 7.3.

When the device enters this state, it may have a device address in either the unauthenticated or USB device address range. If the host decides to completely re-enumerate the device, the following ordered set of control operations must successfully complete in order to transition the device to the **Default** sub-state of the **Authenticated** device state. Note, this is the required sequence the host must take when the device is coming from the **UnConnected** device state.

1. The host successfully completes the authentication process (4-way handshake). This set of control transfers establishes the PTK (used for data packet encryption).
2. The host successfully completes a SetKey(GTK) request. The host uses this request to load the current GTK onto the device so that the device can authenticate Wireless USB Channel broadcast packets (e.g.

MMCs). The host must encrypt the data stage of this request (using the PTK established during the 4-way handshake) in order to protect the delivery of the GTK.

3. And finally, the host completes a SetAddress(0) request. The device must authenticate the MMC which includes the new device address using the GTK.

After the 4-way handshake completes, the device and host are required to begin using the PTK to encrypt all data phase and handshake phase transaction packet transmissions. After the SetKey(GTK) is complete, the device must authenticate all MMCs before responding to requests.

The host may choose to simply re-authenticate the device and return it to its previous **Authenticated** device sub-state. To accomplish this the host must first re-authenticate the device (successfully complete a 4-way handshake) and optionally a SetAddress() to the device's previously authenticated USB device address. Note the SetAddress() is only required here if the host responded to the *DN_Connect* with a device address in the UnAuthenticated_Device_Address_Range.

If the ordered set of control operations fails to complete within *TrustTimeout* seconds (start to finish), the device returns to the **UnConnected** device state. Note that if the 4-way handshake fails from the host's perspective, the host will simply not continue with the authentication control requests. The host may give up retrying the SetKey() and SetAddress() requests after an implementation-specific number of tries. If the device responds to any of the control requests in this sequence with a STALL response, it will then return to the **UnConnected** device state.

There are no intended inter-dependencies between the different kinds of control requests that are valid in this state, besides those described above between the 4-way handshake, the SetKey(GTK) and the SetAddress(). In general a host should perform all ancillary control requests to read pertinent information from the device before beginning the ordered sequence of commands required to transition the device to the **Authenticated** device state.

7.1.3 Authenticated

The intent of this state is that it is the 'normal' operating state for functional data communications using secure packet encapsulation. If the device address on entry to this state is zero (0), then the required destination sub-state is the **Default** state. Whenever a SetAddress(0) completes, the device will transition directly to the **Default** device state. The side-effects of a SetAddress(0) are defined in Section 4.12. If the device address on entry to this state is not zero, the device returns to the appropriate sub-state it was in previously when it transitioned from the **Authenticated** to **Reconnecting** state.

The definition and use of the **Address** and **Configured** device sub-states are identical to those defined in the USB 2.0 specification (Chapter 9). Note that only the Default Control Pipe is available for data communications over the Wireless USB channel when the device is not in the Configured device state (see Figure 7-1). By definition, function endpoints do not exist until the device has been configured; therefore, a device must not respond to transactions addressed to non-configured endpoints.

Note that the host may initiate a 4-way handshake at any time with the device, including while it is in any sub-state of the **Authenticated** device state. The control transfers used to conduct the 4-way handshake do not use secure data encapsulation during the data and handshake phases of the control transfers. A device must continue to use its valid GTK to authenticate the MMCs transmitted by the host. This is the only exception to the rule that data communications that occur during the **Authenticated** device state use secure packet encapsulation.

A device will exit this state under the following:

- **Explicit disconnect event.** The device or host has initiated an explicit disconnect or the User of the device has initiated a New Connect operation. The device transitions to the **UnConnected** device state.
- **Authentication Refresh Fails.** In other words, a 4-way handshake fails to complete. The device transitions to the **UnConnected** device state.
- **Trust Timeout Event.** As described in Section 6.2.10.2 a device must not trust the data communications with its host whenever it loses communications for greater than a *TrustTimeout*. Precisely, when a device

does not observe Wireless USB channel broadcast packets (e.g. MMCs) from its host for a *TrustTimeout* period, it must cease responding to any data transactions and transition to the **Reconnecting** device state.

7.1.4 Reconnecting

The device enters this state whenever it has not received a Wireless USB channel broadcast packet (e.g. MMC) for greater than *TrustTimeout* seconds. When in this state, the device will attempt to reconnect to its host using the *DN_Connect* device notification as describe in Section 7.6.1.2. Devices must use secure packet encapsulation (i.e. *SEC* bit = 1b) when transmitting *DN_Connect* notifications while in this state.

The device will transition to the **UnAuthenticated** device state when the host acknowledges a reconnect notification. The host response to the reconnect notification is a Connect Acknowledge IE with *bDeviceAddress* field value equal to either the value of the *Previous Address* field in the *DN_Connect* notification or an address in the UnAuthenticated_Device_Address_Rrange.

The device will transition to the **UnConnected** device state if the host does not respond to the reconnect device notification attempts after 6 attempts.

Note that a device must retain its previous context from the **Authenticated** device state in the event the host does not assign the device address 0 during the process of returning it to the **Authenticated** device state (i.e. simply returns the device to its previous **Authenticated** device sub-state).

7.2 Generic Wireless USB Device Operations

All devices support the generic operations defined in USB 2.0. This section explicitly describes the new generic device operation specific to Wireless USB.

7.3 Standard Wireless USB Device Requests

All devices must support the required set of standard device requests defined in USB 2.0, chapter 9. All required USB 2.0 standard requests are available once the device is in the **Authenticated** device state. Since the USB device states are encapsulated as sub-states within the **Authenticated** device state, any and all USB 2.0 restrictions on request use or availability based on device state continue to be valid.

Wireless USB also places restrictions on which requests are allowed to be used outside of the Authenticated state. These restrictions provide a narrow window of functionality while the association and authentication processes are active. Table 7-1 summarizes these use restrictions for all requests defined in the USB 2.0 specification. The remainder of this section details modifications to the USB 2.0 standard requests and defines new standard requests for Wireless USB. Each request is annotated with information about what device states the request is available. Note the host must use the base signaling rate for all standard device requests (i.e. control transfers to the Default Control Pipe) unless specifically noted otherwise.

Table 7-1. Standard Request Availability in Wireless USB Device States

Request	Available in UnAuthenticated Device State	Note
CLEAR_FEATURE	Yes	
GET_CONFIGURATION	Yes	
GET_DESCRIPTOR	Yes	The type of descriptors that can be read from the device are limited to the Device and BOS Descriptors, any String Descriptors and all of the Security Descriptors.
GET_INTERFACE	No	This request is only valid in the Configured device state because it queries the current configuration.

Table 7-1. Standard Request Availability in Wireless USB Device States (cont.)

Request	Available in UnAuthenticated Device State	Note
GET_STATUS	Yes	
LOOPBACK_DATA_READ	Yes	
LOOPBACK_DATA_WRITE	Yes	
SET_ADDRESS	Yes	Note, see Section 7.3.1.3 for additional special requirements for handling this request in Wireless USB.
SET_CONFIGURATION	No	
SET_DESCRIPTOR	No	
SET_FEATURE	Yes	
SET_INTERFACE	No	This request is only valid in the Configured device state because it queries the current configuration.
SET_INTERFACE_DS	No	This request is optional. It is supported by devices with interfaces that support the dynamic switching mechanism.
SYNCH_FRAME	No	

7.3.1 Wireless USB Extensions to Standard Requests

This section describes extensions to the standard set of device requests defined in the USB 2.0 specification [1]. This includes extensions to the requests defined in the USB 2.0 specification (Section 9.4) and specific requests defined for Wireless USB. Table 7-2 summarizes the standard Wireless USB device requests and Table 7-20 summarizes the Wireless USB Security Requests.

Table 7-2. Wireless USB-specific Standard Device Requets

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	CLEAR_FEATURE	WUSB_DEVICE	WUSB Feature Selector and Feature Value	Zero	None
10000000B	GET_STATUS	Zero	Device Status Selector	Variable	Status Selector Data
00000000B	SET_ADDRESS	Device Address	Zero	Zero	None
00000000B	SET_FEATURE	WUSB_DEVICE	WUSB Feature Selector and Feature Value	Zero	None
00000001B	SET_INTERFACE_DS	Alternate Setting	Interface	2	Presentation Time
00000000B	SET_WUSB_DATA	WUSB Data Selector	Zero	WUSB Data Length	WUSB Selector Data
00000000B	LOOPBACK_DATA_WRITE	Zero	Zero	Data Length	Data
10000000B	LOOPBACK_DATA_READ	Zero	Zero	Data Length	Data

The USB 2.0 specification [1], Section 9.4 lists request codes for the standard device commands over the Default Pipe. Table 7-3 is an annotated list of additional standard request codes for devices. The annotations indicate whether the request code is specific to security.

Table 7-3: Wireless USB Standard Request Codes

bRequest	Value	Purpose
SET_ENCRYPTION	13	Security
GET_ENCRYPTION	14	Security
SET_HANDSHAKE	15	Security
GET_HANDSHAKE	16	Security
SET_CONNECTION	17	Security
SET_SECURITY_DATA	18	Security
GET_SECURITY_DATA	19	Security
SET_WUSB_DATA	20	General
LOOPBACK_DATA_WRITE	21	General
LOOPBACK_DATA_READ	22	General
SET_INTERFACE_DS	23	General

The USB 2.0 specification, Section 9.4 lists standard feature selector values for enabling or setting specific features. Table 7-4 is a list of additional standard features selectors for devices. Wireless USB uses one feature selector value (WUSB_DEVICE) then defines a family of Wireless USB specific features which are relative to WUSB_DEVICE feature selector. The commands ClearFeature, SetFeature and GetStatus detail how Wireless USB uses these features.

Table 7-4. Wireless USB Standard Feature Selectors

Feature Selector	Recipient	Value	Wireless USB Feature Selectors	
WUSB_DEVICE	Device	3	Feature Name	Code
			TX DRP IE	0
			DEV XMIT_PACKET	1
			COUNT PACKETS	2
			CAPTURE PACKETS	3

7.3.1.1 Clear Feature

This request is used to clear a specific feature. The USB 2.0 defined uses of this command remain in effect for Wireless USB, see Section 9.4.1 in the USB 2.0 specification. The following description details extensions to this request for Wireless USB.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	CLEAR_FEATURE	Feature Selector = WUSB_DEVICE	Wireless USB Feature Selector	Zero	None
			Zero or Feature Value		

When the feature selector field (*wValue*) is set to WUSB_DEVICE, the least significant byte of *wIndex* is used to further qualify which Wireless USB device feature is to be modified with this request. Wireless USB device features are summarized in Table 7-13. Table 7-5 summarizes which of the Wireless USB features can be modified with the ClearFeature() request.

UnAuthenticated State: If the the specified feature selector is WUSB_DEVICE, device behavior when this request is received while the device is in the **UnAuthenticated** state is not specified..

Default State: If the specified feature selector is WUSB_DEVICE, device behavior when this request is received while the device is in the **Default** state is not specified.

Address State: This request is valid in the **Addressed** state.

Configured State: This request is valid in the **Configured** state.

Table 7-5. Features Modifiable via ClearFeature()

Wireless USB Feature Selector	Explanation
TX DRP IE	On receipt of this request, the device will remove the associated DRP IE from its MAC Layer Beacon. This request must only be sent to Self Beaconing devices.
DEV XMIT_PACKET	On receipt of this request, the device will be disabled from directed packet transmission. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.1.
COUNT PACKETS	On receipt of this request, the device will be disabled for count packets operation. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.2.
CAPTURE PACKET	On receipt of this request, the device will be disabled for capture a packet operation. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.3.

7.3.1.2 Get Status

This request returns status information about different portions of a device. The following description details the extensions to the GetStatus() request for Wireless USB. For a description of the standard USB 2.0 request, refer to Section 9.4.5 in the USB 2.0 specification.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_STATUS	Zero	Device Status Selector	Variable	Status Selector Data

The format of this request is a standard GetStatus() request as defined in USB2.0, with extended definition of the *wIndex* field when the recipient is DEVICE. The USB 2.0 specification requires that *wIndex* have a value of zero when *bmRequestType* specifies the recipient is the device (with a *bRequest* of GET_STATUS). The extension for Wireless USB encodes *wIndex* with a value indicating the specific device information the host is interested in. Note that the availability of this request depends on the state of the device and the addressed recipient. For example, a host should not address this request to an endpoint when the device is in the **Default** state because it has undefined results.

UnAuthenticated State: This request is valid in the **UnAuthenticated** state.

Default State: This request is valid in the **Default** state (depending on recipient).

Address State: This request is valid in the **Addressed** state.

Configured State: This request is valid in the **Configured** state.

Table 7-6. Device-Level Status Selector Encodings for wIndex

wIndex	Status Type	Description
0000H	USB 2.0 Standard Status	This is the default encoding for this command defined in the USB 2.0 specification. It summarizes a small set of device level status indicators. This set of status indicators has been extended as described below for Wireless USB
0001H	Wireless USB Feature	This encoding returns the current values of Wireless USB specific features.
0002H	Channel Info	This encoding indicates the device must return information about its view of the Wireless USB channel. The format of the data returned is defined below.

Table 7-6. Device-Level Status Selector Encodings for wIndex (cont.)

wIndex	Status Type	Description
0003H	Received Data	This encoding instructs the device to return the data collected during the last count packets or capture a packet operation. This request must only be sent to Directed Beaconing devices.
0004H	MAS Availability	This encoding instructs the device to return its MAS Availability information, see below for details. This request must only be sent to Self Beaconing devices.
0005H	Current Transmit Power	This encoding instructs the device to return its current transmit power settings for Notifications and Beacons, see below for details.

USB 2.0 Standard Features

When the *wIndex* value is **USB 2.0 Standard Status**, the device returns the information illustrated in Figure 7-2.

Byte	D7	D6	D5	D4	D3	D2	D1	D0
0	Reserved, must be set to zero					Battery Powered	Remote Wake	Self Powered
1	Reserved, must be set to zero							

Figure 7-2. USB 2.0 Standard Status Information Returned by a GetStatus() Request to a Device

Refer to Section 9.4.5 of the USB 2.0 specification for descriptions of the *Remote Wake* and *Self Powered* fields. The *Self Powered* field should always return set to a one (1B) for a Wireless USB device. The default value for *Remote Wake* is defined in the USB 2.0 specification.

The *Battery Powered* field indicates whether the device is currently battery-powered. If D2 is reset to zero, the device is powered by a supply that is not a battery. The *Battery Powered* field value may not be changed by the SetFeature() or ClearFeature() requests.

Wireless USB Features

When the *wIndex* value is **Wireless USB Feature Status**, the device returns the information illustrated in Figure 7-3. The default values of these features after any device power-up or reset event is zero.

Byte	D7	D6	D5	D4	D3	D2	D1	D0
0	Reserved, must be set to zero				Capture Packet	Count Packets	Transmit Packet	TX DRP IE

Figure 7-3. Information Returned by GetStatus(WirelessUSBFeatures)

The *TX DRP IE* field indicates whether the device is enabled to add a DRP IE to its beacon. If *TX DRP IE* is a one, the device transmits the DRP IE information set by the cluster host. This field is changed by SetFeature() and ClearFeature(). This is a read-only field unless the device is a Self-beaconing device.

The *Transmit Packet* field indicates whether the *Directed Packet Transmission* feature is active. See Section 4.3.7.2.1 for a description of device behavior when doing Directed Packet Transmission.

The *Count Packets* field is set if the device is enabled for counting packets and reset otherwise. See Section 4.3.7.2.2 for a description of device behavior when enabled for counting packets.

The *Capture Packets* field is set if the device is enabled for capturing a packet and reset otherwise. See Section 4.3.7.2.3 for a description of device behavior when enabled for capturing a packet.

Channel Information

When *wIndex* value is equal to *Channel Info*, the device must return the information it has gathered with regards to the state of the underlying PHY channel. The format of the data returned is summarized in Table 7-7. A host can use this command to determine how well the device is receiving MMCs from the host.

Table 7-7. Wireless USB Channel Status Information Returned by Device

Offset	Field	Size	Value	Description
0	*LQI*	1	Number	LQI value of the last MMC packet received from the Host.

Received Data

When *wIndex* value is equal to *Received Data*, the device must return the information it has gathered from the latest Count Packets or Capture Packet operation. Section 4.3.8.3 describes operation of the Count Packets and Capture Packet operations. The format of the data returned is dependent on which operation was last completed.

Table 7-8 shows the format of data returned from a Count Packets operation. For each counted packet the device returns a 10 byte block that includes the Reception Time, the LQI, and the first six bytes of the MAC header.

Table 7-8. Count Packets Data format

Offset	Field	Size	Description
0	*Packet Count*	1	Number of packets that were counted and whose info is returned. Bits 0-6 hold the actual packet count. Bit 7 is set if there was a buffer overrun and zero otherwise.
1	*Reception Time*	3	USB Channel Time when this packet was received. Time marker is the beginning of the preamble.
4	*MAC Header Info*	6	First six bytes of the MAC header of the counted packet.
10	*LQI*	1	LQI value of the counted packet. A definition of LQI value is provide below.
11	*Next Packet info Block*	10	The block of information for the next counted packet.
…	…	…	…

Table 7-9 shows the format of data returned from a Capture Packet operation. For the captured packet, the device returns the MAC header and payload. The device also provides the time the packet was received and indicates if there was any problem receiving the packet.

Table 7-9. Capture Packet Data format

Offset	Field	Size	Description
0	*Receive Status*	1	Bit vector that provides reception information about the captured packet. Bit 0 is set if the captured packet did not have a valid FCS. Bit 1 is set of the captured packet was larger than the available buffer area. Bits [7:2] are reserved and must be set to zero.
1	*Reception Time*	3	USB Channel Time when this packet was received. Time marker is the beginning of the preamble
4	*MAC Header*	10	MAC Layer header from the captured packet.
14	*Payload*	N	Payload of the captured packet.
14+N	*LQI*	1	LQI value of the capture packet. A definition of LQI value is provided below.

LQI is an average SNR value at the FFT output for the received packet.

Table 7-10. Definition of LQI values

LQI Value	SNR
0	Too low to be accurately measured
1	- 6 dB
2	- 5 dB
…	…
31	24 dB
32-255	Reserved

MAS Availability

When the *wIndex* value is equal to *MAS Availability*, the device will then proceed to accumulate information from its neighbor's beacons about which MAS slots are available for this device to use (not reserved by any neighbor not a member of the Wireless USB cluster). Note that the device may have MAS Availability information that is current so that it does not have to accumulate the information. The data content of the data returned by the device is formatted as illustrated in Table 7-11. The device must ignore its host' s DRP IEs for the current Wireless USB channel when building its availability map.

The host must only issue this GetStatus request to a device that has identified itself as a Self-beaconing device. A Non- or Directed Beaconing device must respond to this request with a Request Error.

Table 7-11. MAS Availability Device Status Format

Offset	Field	Size	Value	Description
0	*bmMASAvailability*	32	Bitmap	This is a 256-bit map, where each bit location corresponds to a MAS slot in the MAC Layer super-frame. A 1B in a bit location means that the device is available for a reservation in the corresponding MAS slot. A 0B indicates the device is not available. Bit 0 corresponds to MAS slot 0.

Current Transmit Power

When the *wIndex* value is equal to *Current Transmit Power*, the device will respond with the current power level setting. The data content of the data returned by the device is formatted as illustrated in Table 7-12.

Table 7-12. Current Transmit Power Status Format

Offset	Field	Size	Value	Description
0	*bTxNotificationTransmitPower*	1	Number	Value indicating the number of steps below the highest power level that is currently being used for notification transmissions. The default value is zero.
1	*bTxBeaconTransmitPower*	1	Number	Value indicating the number of steps below the highest power level that is currently being used for beacon or directed transmissions. The default value is zero.

7.3.1.3 Set Address

This request is nominally identical to that specified in the USB 2.0 specification (see Section 9.4.6 in Reference [1]). The requirement of a device, for this request is to retain the current address until the Set Address request is complete. The device nominally considers the Set Address transfer complete when it transmits the Handshake packet for the Status stage. In order to tolerate the loss of the handshake and subsequent retries of the Status stage handshake, a device is required to retain the old device address until the host begins to use the new address which was sent to the device in the Set Address transfer. Note, the host must provide at least 2ms of relaxation time from the end of the SetAddress request and a request addressed to the new device address.

UnAuthenticated State:	This request is valid in the **Unauthenticated** state. See 7.1.2 for rules for side effects of this request when in the **UnAuthenticated** state.
Default State:	This request is valid in the **Default** state. If the address specified is zero, then the device must remain in the **Default** state.
Address State:	This request is valid in the **Addressed** state. If the address specified is zero, then the device must enter the **Default** state; otherwise, the device remains in the **Address** state but uses the newly-specified address.
Configured State:	This request is valid in the **Configured** state.

7.3.1.4 Set Feature

This request is used to set or enable a specific feature. The USB 2.0 defined uses of this command remain in effect for Wireless USB see Section 9.4.9 in the USB 2.0 specification. Note that the single exception to this is the TEST_MODE feature, which is explicitly not supported (as documented) by devices. The following description details extensions to this request for Wireless USB.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_FEATURE	Feature Selector = WUSB_DEVICE	Wireless USB Feature Selector	Zero	None
			Zero or Feature Value		

When the feature selector field (*wValue*) is set to WUSB_DEVICE the least significant byte of *wIndex* is used to further qualify which Wireless USB device feature is to be manipulated with this request. The Wireless USB device features are summarized in Table 7-13.

If wLength is non-zero, then the behavior of the device is not specified.

UnAuthenticated State:	If the specified feature selector is WUSB_DEVICE, device behavior when this request is received while the device is in the **UnAuthenticated** state is not specified.

Default State: If the specified feature selector is WUSB_DEVICE, device behavior when this request is received while the device is in the **Default** state is not specified.

Address State: This request is valid in the **Addressed** state.

Configured State: This request is valid in the **Configured** state.

Table 7-13. Features Modifiable via SetFeature()

Wireless USB Feature Selector	Description
TX DRP IE	TX_DRPIE. The side-effect of this setting is that the device must add a DRP IE to its beacon. The data for the DRP IE comes from information extracted from the Wireless USB Channel and from the host setting the WUSB Control Data for DRP IE Data. This request must be sent only to a Self Beaconing device.
DEV XMIT_PACKET	When this feature is set, the device is enabled for directed packet transmission. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.1 for a description of directed packet transmission.
COUNT PACKETS	When this feature is set, the device is enabled for counting packets. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.2 for a description of counting packets functionality.
CAPTURE PACKET	When this feature is set, the device is enabled for capturing a packet. This request must only be sent to Directed Beaconing devices. See Section 4.3.7.2.3 for a description of capture packet functionality

7.3.1.5 Set Interface DS

This request is to instruct the device to switch operation to a specified alternate setting for an interface at a specified time. The host must only send this request to devices that indicate that they support dynamic switching.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000001B	SET_INTERFACE_DS	Alternate Setting	Interface	2	Switch Time

Switch Time is a Wireless USB Channel time with 1/8 of a millisecond granularity. The time indicates when the interface should switch its operational characteristics to those specified by the alternate setting. This request is typically used to switch an interface to an alternate setting with different bandwidth requirements for one or more isochronous endpoints. Isochronous endpoints change their characteristics at the specified time, but do not flush associated data buffers. An Isochronous IN endpoint starts generating data in the new format at the specified Wireless USB Channel time (switch time). An Isochronous OUT endpoint assumes that all data received with presentation times after Switch Time are in the format associated with the specified alternate setting. The time at which data in the new format begins to be transmitted over the air is unknown when the Set Interface DS command occurs. The host must send a Set Interface request with the same alternate setting and interface values to the device when it expects over the air packets to start matching the new interface setting. See the dataflow chapter for a more detailed description of the use of Set Interface DS.

If the alternate setting specified does not exist the device responds with a request error.

UnAuthenticated State: Device behavior when this request is received while the device is in the **UnAuthenticated** is not specified.

Default State: Device behavior when this request is received in the **Default** state is not specified.

Address State: The device must respond with a request error.

Configured State: This request is valid in the **Configured** state.

7.3.1.6 Set WUSB Data

This request is required for Wireless USB devices. It is specifically used to update Wireless USB-specific descriptors and controls.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_WUSB_DATA	WUSB Data Selector	Zero	WUSB Data Length	WUSB Data

The *wValue* field specifies a selector value that corresponds to the data or control information. Table 7-14 summarizes the Wireless USB data selectors for items that can be modified on the device via this request.

If *wLength* is zero, then the behavior of the device is not specified.

If *wIndex* or *wValue* are not as specified above, the device responds with a Request Error.

UnAuthenticated State: Device behavior when this request is received while the device is in the **UnAuthenticated** state is not specified.

Default State: Device behavior when this request is received while the device is in the **Default** state is not specified..

Address State: This request is valid in the **Addressed** state.

Configured State: This request is valid in the **Configured** state.

Table 7-14. Wireless USB Data Selector Encodings for wValue

wValue	Selector Name	Description
0000H	Reserved	This encoding is reserved for future use.
0001H	DRPIE Info	This encoding indicates that the WUSB Data is information the device must use to construct and transmit a DRP IE in its beacon. This encoding must only be used to send DRP IE information to a Self-beaconing device. Table 7-15 illustrates the format of the WUSB Data for this encoding.
0002H	Transmit Data	This encoding indicates that the WUSB Data is the transmit packet data for a Directed Beaconing device. This encoding must only be sent to Directed Beaconing devices. Table 7-16 illustrates the format of the WUSB Data for this encoding.
0003H	Transmit Params	This encoding indicates that the WUSB Data contains parameters for directed packet transmission. This encoding must only be sent to Directed Beaconing devices. Table 7-17 illustrates the format of the WUSB Data for this encoding.
0004H	Receive Params	This encoding indicates that the WUSB Data contains parameters for counting packets or capturing a packet. This encoding must only be sent to Directed Beaconing devices. Table 7-18 illustrates the format of the WUSB Data for this encoding.
0005H	Transmit Power	This encoding indicates that the WUSB Data contains numeric values that correspond to the transmit power level that the device must use for all subsequent notification and beacon (or directed) transmissions. The device uses the highest power level for all transmissions by default. Table 7-19 illustrates the format of the WUSB Data for this encoding.
0006H - FFFFH		Reserved for future use.

DRPIE Information

A self-beaconing device uses this data in conjunction with data it derives from its host's Wireless USB channel to construct the DRP IE that it transmits in its beacon. A host may issue this command regardless of the current setting of the *TX DRP IE* feature. The device must include the new data within the next 2 Beacon transmissions. The Self Beaconing device must not modify the contents of the received DRP IE.

Table 7-15. DRP IE WUSB Data Format

Offset	Field	Size	Value	Description
0	*bmAttributes*	1	Bitmap	The values of these fields are used to construct the DRP Control field of a DRP IE. The field definitions are: **<u>Bits</u>** **<u>Description</u>** 2:0 Reservation Priority 7:3 Reserved. Must be zero
1	*DRPIEData*	4×N	DRP Allocation	This is the DRP Allocation blocks that must be included in the DRP IE transmitted by the device. N is the number of reservation blocks and the size of each block is 4 bytes. See Table 7-64 for full layout of a cluster member DRP IE layout.

Note, if the Self Beaconing device does not have an existing DRP IE for this Wireless USB channel, it simply adds the received DRP IE to its beacon. If the device has an existing DRP IE for this Wireless USB channel, then it must replace the existing DRP IE (for this Wireless USB channel) with the new DRP IE provided in this command payload.

Transmit Data

A directed beaconing device stores this data and transmits it in a packet when enabled for directed packet transmission. Directed packet transmission occurs at the power level specified by the current Beacon Transmit Power setting for the device. (See Section 4.3.7.2.1 in the Data Flow chapter). A host must not send this request to a device that has the *TRANSMIT PACKET* feature set.

Table 7-16. Transmit Data format

Offset	Field	Size	Value	Description
0	*MAC Header*	10	Record	This is the 10 byte MAC Layer header that the device must use when transmitting the packet.
11	*Payload*	N	Bitmap	This is the packet payload that the device must use when transmitting the packet.

Transmit Params

A directed beaconing device stores this data and uses it to determine when to transmit a packet when enabled for directed packet transmission. A host must not issue this command if the *TRANSMIT PACKET* feature is set. See Section 4.3.7.2.1 in the Data Flow chapter for information on how this data is used.

Table 7-17. Transmit Params format

Offset	Field	Size	Value	Description
0	*Transmit Time*	3	Number	USB Channel Time when the device must transmit a packet when enabled for directed packet transmission.
3	*Transmit Adjustment*	1	Number	Number of additional microseconds (above 64K) that a device should include in the repeat interval when doing directed packet transmission.

Receive Params

A directed beaconing device stores this data and uses it to determine where and when to count packets or capture a packet. See Section 4.3.7.2 in the Data Flow chapter for complete information on these operations. A host must not send this request to a device that has the *CAPTURE PACKET* or *COUNT PACKET* feature set.

Table 7-18. Receive Params format

Offset	Field	Size	Value	Description
0	*Receive Filter*	1	Bitmap	Bit vector that the device uses to determine if the received packet should be used in the count or capture operation.
1	*Receive Channel*	1	Number	PHY channel that device should use for count or capture operation. Encoding matches encoding of PHY specification.
2	*Receive Start Time*	3	Number	USB Channel Time when device should begin count or capture operation..
5	*Receive End Time*	3	Number	USB Channel Time when device should end count or capture operation.

Transmit Power

A host may send this command at any time the device state allows it to be sent. The format of the WUSB Data for Transmit Power selector is shown in Table 7-19. A device is required to use the new power setting for the next appropriate packet transmission after the device has transmitted the Status stage handshake to the Set WUSB Data request.

Table 7-19. Transmit Power WUSB Data Format

Offset	Field	Size	Value	Description
0	*bTxNotificationTransmitPower*	1	Number	Value indicating the number of steps below the highest power level that must be used for notification transmissions. The host must use a value that is supported by the device.
1	*bTxBeaconTransmitPower*	1	Number	Value indicating the number of steps below the highest power level that must be used for beacon or directed transmissions. The value must be zero for a Non-beaconing device.

7.3.1.7 Data Loopback Write

This request must be supported by all devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	LOOPBACK_DATA_WRITE	Zero	Zero	Data Length	Data

The data length must be less than or equal to the largest maximum packet size (devMaxPacketSize) of all the device's endpoints in any configuration. Device behavior is not specified if the value in *wLength* is larger than devMaxPacketSize. The device is required to store the data payload received in the data stage of the request.

For full requirements for the Data Loopback Write command and device behavior with stored loopback data refer to Section 4.7.4.

If *wValue* or *wIndex* are not as specified above, the device behavior is not specified.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default state: This is a valid request when the device is in the **Default** state.

Address state: This is a valid request when the device is in the **Address** state.

Configured state: This is a valid request when the device is in the **Configured** state for a device that contains one or more isochronous function endpoints in any of its configurations. For all other devices, behavior is undefined if this request occurs in the **Configured** state.

7.3.1.8 DATA Loopback Read

This request must be supported by all devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	LOOPBACK_DATA_READ	Zero	Zero	Data Length	Data

The data length must be less than or equal to the largest maximum packet size (devMaxPacketSize) of all the device's endpoints. Device behavior is not specified if the value in *wLength* is larger than devMaxPacketSize.

For full requirements for the Data Loopback Read command and device behavior with stored loopback data, refer to Section 4.7.4.

If *wValue* or *wIndex* are not as specified above, the device behavior is not specified.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default state: This is a valid request when the device is in the **Default** state.

Address state: This is a valid request when the device is in the **Address** state.

Configured state: This is a valid request when the device is in the **Configured** state for a device that contains one or more isochronous function endoints in any of its configurations. For all other devices, behavior is undefined if this request occurs in the **Configured** state.

7.3.2 Security-related Requests

This section describes the Requests defined for the USB Security Framework.

Table 7-20: Security Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
GetKey	10000000B	GET_DESCRIPTOR	Descriptor Type and Key Index	Zero	Descriptor Length	Key Descriptor
SetKey	00000000B	SET_DESCRIPTOR	Descriptor Type and Key Index	Zero	Descriptor Length	Key Descriptor
Handshake1	00000000B	SET_HANDSHAKE	One	Zero	Length of Handshake 1 Data	Handshake 1 Data
Handshake2	10000000B	GET_HANDSHAKE	Two	Zero	Length of Handshake 2 Data	Handshake 2 Data
Handshake3	00000000B	SET_HANDSHAKE	Three	Zero	Length of Hanshake3 Data	Handshake 3 Data
GetSecurityDescriptor	10000000B	GET_DESCRIPTOR	Descriptor Type	Zero	Descriptor Length	Descriptor Data
SetEncryption	00000000B	SET_ENCRYPTION	Encryption Value	Zero	Zero	None
GetEncryption	10000000B	GET_ENCRYPTION	Zero	Zero	One	Encryption Value
SetConnectionContext	00000000B	SET_CONNECTION	Zero	Zero	Fourty-eight	Connection Context
SetSecurityData	00000000B	SET_SECURITY_DATA	Data Number	Zero	Data Length	Security Data
GetSecurityData	10000000B	GET_SECURITY_DATA	Data Number	Zero	Data Length	Security Data

7.3.2.1 Get Security Descriptor

The host uses this command to retrieve the Security Descriptor and its associated sub-descriptors from the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Zero	Zero	Length of Descriptor	Security Descriptor

It is a Request Error if *wValue* or *wIndex* are other than as specified above. The request is valid for any device in the **Connected** device state.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Addressed** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.2 Set Encryption

The host uses this command to set the current device encryption type.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_ENCRYPTION	Encryption Value	Zero	Zero	None

The host uses this command to inform the device of the type of encryption that will be used. The host determines the types of encryption the device supports by enumerating the security descriptor and its encryption type descriptors. The host also uses this request to enable CCM encryption on the device before beginning the 4-way handshake.

Encryption Value comes from one of the Encryption Type descriptors contained in the Security Descriptor. A value of Zero always represents UNSECURE or no encryption. The request is valid for any device in the **Connected** device state.

It is a Request Error if Encryption Value does not represent a valid encryption type.

It is a Request Error to attempt to set WIRED as the current encryption type.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State:	This is a valid request when the device is in the **UnAuthenticated** state.
Default State:	This is a valid request when the device is in the **Default** state.
Address State:	This is a valid request when the device is in the **Address** state.
Configured State:	This is a valid request when the device is in the **Configured** state.

7.3.2.3 Get Encryption

The host uses this command to get the current device encryption type.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_ENCRYPTION	Zero	Zero	One	Encryption Value

Encryption Value comes from one of the Encryption Type descriptors contained in the Security Descriptor. A value of Zero always represents UNSECURE or no encryption.

A wired/wireless device always returns WIRED if it connected with a cable.

It is a Request Error if Encryption Value does not represent a valid encryption type.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State:	This is a valid request when the device is in the **UnAuthenticated** state.
Default State:	This is a valid request when the device is in the **Default** state.
Address State:	This is a valid request when the device is in the **Address** state.
Configured State:	This is a valid request when the device is in the **Configured** state.

7.3.2.4 Key Management

This section describes the requests that are related to key management.

The Security Framework uses Key Indices in both descriptors and requests. These values are used to specify individual keys. The Key Index has the following layout:

Table 7-21: Key Index definition

Bit	Description
0-3	Index: Allows selection of one of several of the same type of key.
4-5	Type: Specifies the type of key: 0 = Reserved 1 = Association Key 2 = GTK 3 = Reserved for future use
6	Originator: specifies original source of the key: 0 = Host 1 = Device
7	Reserved for future use

Devices may use the index field to select between multiple device-originated authentication keys.

Host-originated public keys use an index of zero (0).

7.3.2.4.1 Set Key

The host uses this command to distribute GTKs and host association keys. When Set Key is being used to give an initial GTK to a device, the device must initialize the replay counter associated with this GTK to the SFC value used in the secure encapsulation of the MMC where the Setup command bytes for this command are located. To support this, the host is required to deliver the currently active GTK to a device as its initial GTK.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_DESCRIPTOR	Descriptor Type and Key Index	Zero	Descriptor Length	Key Descriptor

When the device receives this command, it uses the key data in the accompanying descriptor to update its copy of the key specified by Key Index. The request is valid for any device in the Connected device state. This request may not be used to distribute new PTKs. PTKs are only distributed using the 4-way Handshake.

It is a Request Error if *wIndex* or *wValue* are other than as specified above.

It is a Request Error if key originator=device.

It is Request Error to distribute any key other than a host public key in UNSECURE encryption mode.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Addressed** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.4.2 Get Key

The host uses this request to get key descriptors from the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_DESCRIPTOR	Descriptor Type and Key Index	Zero	Descriptor Length	Key Descriptor

When the device receives this command, it uses Key Index to reference the appropriate key. The request is valid for any device in the Connected device state.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

It is a Request Error if Key Index refers to a non-existent key.

It is a Request Error if Key Index specifies any key type other than a public key

UnAuthenticated State:	This is a valid request when the device is in the **UnAuthenticated** state.
Default State:	This is a valid request when the device is in the **Default** state.
Address State:	This is a valid request when the device is in the **Address** state.
Configured State:	This is a valid request when the device is in the **Configured** state.

7.3.2.5 4-Way Handshake

This series of requests are used to establish a 4-way handshake between host and device. This 4-way handshake provides a means for the host and device to perform mutual authentication using the Connection Key while simultaneously deriving the initial PTKs.

The host always assumes an Initiator role in the 4-way handshake while the device always assumes the role of responder. During the 4-way handshake, the host and device exchange 128-bit cryptographic grade random numbers. These numbers are assembled and processed as described in the Security chapter.

The device must expect the individual handshake requests in order: Handshake1, Handshake2 and Handshake3. If a device receives a Handshake1 while it is processing or waiting for a Handshake2 or Handshake3 request, it must abort the handshake in progress and start again with the newly received Handshake1 request. If the device receives an unexpected handshake request, it must report a request error to the host. The format of the Handshake data for all Handshake requests is given in Table 7-22.

Table 7-22. Format of Handshake Data for the Handshake Commands

Offset	Field	Size	Value	Description
0	*bMessageNumber*	1	Number	Defines which stage this data payload is associated with. Valid values are: 1 Handshake1 2 Handshake2 3 Handshake3
1	*bStatus*	1	Number	0 Normal, the handshake sequence proceeds 1 Aborted per security policy 2 Aborted, handshake in progress with same master key 3 Aborted, TKID conflict
2	*tTKID*	3	Number	The base name for the PTK to be derived

Table 7-22. Format of Handshake Data for the Handshake Commands (cont.)

Offset	Field	Size	Value	Description
5	*bReserved*	1	Constant	Reserved for future use; Must be zero
6	*CDID*	16	Number	The device's CDID. Note, this corresponds to the MKID used by the MAC Layer, see reference [3]
22	*Nonce*	16	NONCE	The Nonce being exchanged for the handshake
38	*MIC*	8	Number	The MIC calculated over the previous fields of this packet payload. The value of this field for Handshake1 is zero.

The host establishes the TKID to be used with PTK by declaring this value in the Handshake1 data payload. This value is maintained throughout the handshake sequence. If the host or device receives a handshake payload with a different TKID, the payload should be discarded.

The last 4-way handshake message, Handshake3, is an instruction to install the freshly derived PTK. Upon completion of the status stage of a Handshake3 request, the device should install the derived PTK, enable CCM cryptographic operations and prepare for receipt of secured traffic.

The host may begin a 4-Way Handshake sequence anytime it decides the connection requires re-authentication. The individual 4-Way Handshake requests are valid in all sub-states of the **Connected** device state.

The data and status stages of the individual 4-Way Handshake requests are always sent without secure packet encapsulation, i.e. they are sent in plain-text.

The host must make sure the correct security suite is enabled on the device before beginning a 4-way handshake. It does this by using SetEncryption() to enable CCM encryption.

7.3.2.5.1 Handshake1

The host uses this request to begin the 4-way handshake procedure with a device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_HANDSHAKE	One	Zero	46	Handshake1 Data

This command is used to initiate the 4-way handshake sequence. The host starts the process by sending a key name and a 16-byte cryptographic grade random number, HNonce, to the device. The format of the Handshake1 data is given in Table 7-22.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Address** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.5.2 Handshake2

The host uses this request to retrieve the second 4-way handshake from the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_HANDSHAKE	Two	Zero	46	Handshake2 Data

The host uses this request to retrieve a 16-byte cryptographic grade random number from the device, DNonce, and to validate that the device has derived the correct keys. The format of Handshake 2 Data is given in Table 7-22.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Address** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.5.3 Handshake3

The host uses this request to instruct the device to install the derived key.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_HANDSHAKE	Three	Zero	46	Handshake3 Data

This request combines message 3 and message 4 of a 4-way handshake. The data stage of this request contains the host's message 3. The status stage of this request serves as the device response message 4. The format of the Handshake3 data is given in Table 7-22.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Address** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.5.4 Set Connection Context

The host uses this command to modify a device's Connection Context.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_CONNECTION	Zero	Zero	48	Connection Context

This command is used to modify the current device connection values for CHID, CDID, and CK. This request is only valid for devices in the Authenticated device state. A Connection Context must always be protected during delivery.

A host can also use this command to revoke a context. It does this by setting a context with zero values for CHID and CDID. Any device with a current CDID value of zero must be associated with a host before reconnections can be made.

Table 7-23. Format of Connection Context

Name	Size	Description
Connection Host ID (CHID)	16 bytes	Unique Host ID. The device uses this ID to locate the host's Wireless USB Channel.
Connection Device ID (CDID)	16 bytes	Unique Device ID. This ID uniquely identifies the device to the host specified by CHID. It is not guaranteed to be unique across multiple hosts.
Connection Key (CK)	16 bytes	The key used to establish reconnections using this context. This key should be changed periodically.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request in the **UnAuthenticated** state only after encryption is enabled. This may be as a result of a 4-way handshake/SetKey(GTK) or other authentication protocols tied to New Connection.

Default State: This is a valid request when the device is in the **Default** state.

Address State: This is a valid request when the device is in the **Address** state.

Configured State: This is a valid request when the device is in the **Configured** state.

7.3.2.6 Set Security Data

The host uses this command to pass in-band authentication messages and data to the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_SECURITY_DATA	Data Number	Zero	Data Length	Security Data

This command provides a wrapper for in-band authentication messages and data sent from the host to the device. Data Number represents a stage or message number, defined by the authentication protocol, this data piece is associated with. The request is valid for any device that has requested a New Connection.

It is a Request Error if Data Number does not represent a valid authentication-protocol designated value.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: Device behavior when this request is received while the device is in the **Default** state is not specified.

Address State: Device behavior when this request is received while the device is in the **Address** state is not specified.

Configured State: Device behavior when this request is received while the device is in the **Configured** state is not specified.

7.3.2.7 Get Security Data

The host uses this command to retrieve in-band authentication methods and data from the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_SECURITY_DATA	Data Number	Zero	Data Length	Security Data

This command provides a wrapper for in-band authentication messages and data sent from device to host. Data Number represents a stage or message number, defined by the authentication protocol, this data piece is associated with. The request is valid for any device that has requested a New Connection.

It is a Request Error if Data Number does not represent a valid authentication-protocol designated value.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

UnAuthenticated State: This is a valid request when the device is in the **UnAuthenticated** state.

Default State: Device behavior when this request is received while the device is in the **Default** state is not specified.

Address State: Device behavior when this request is received while the device is in the **Address** state is not specified.

Configured State: Device behavior when this request is received while the device is in the **Configured** state is not specified.

7.4 Standard Wireless USB Descriptors

All devices must support the required set of standard device descriptors defined in USB 2.0, chapter 9. All standard device descriptors are available once the device is in the **Connected** device state (see Section 7.1). Wireless USB defines several changes to some of the standard descriptors defined in USB 2.0, see the summary in Table 7-24. This specification includes only the USB 2.0 defined descriptors that are modified by the Wireless USB specification.

Table 7-24. Summary of Changes to USB 2.0 Defined Standard Descriptors

Descriptor Type	Wireless USB Delta	Explanation
DEVICE	No change	No change to the base descriptor, except for the version number in bcdUSB field, Wireless USB also requires an additional standard device-level capabilities descriptor, see section 7.4.1.
CONFIGURATION	Changes	Updates required for Wireless USB. See Section 7.4.2.
STRING	No change	
INTERFACE	No change	
ENDPOINT	Changes	Updates required for Wireless USB. Also require extensions to endpoint capabilities which utilize a companion descriptor, see Section 7.4.2.
DEVICE_QUALIFIER	n/a	Not allowed for a Wireless USB device
OTHER_SPEED_CONFIGURATION	n/a	Not allowed for a Wireless USB device
INTERFACE_POWER	n/a	Not allowed for a Wireless USB device
OTG	n/a	Not allowed for a Wireless USB device
DEBUG	No Change	See Debug device specification http://developer.intel.com/technology/usb/spec.htm
INTERFACE_ASSOCIATION	No change	

The USB 2.0 specification [1], Section 9.4 lists the standard descriptor types. Table 7-25 is the list of additional descriptor types and assigned descriptor type values for Wireless USB.

Table 7-25. Wireless USB Standard Extension Descriptor Types

Descriptor Types	Value	Section Reference
SECURITY	12	Section 7.4.5.1
KEY	13	Section 7.4.5.2
ENCRYPTION TYPE	14	Section 7.4.5.1.1
BOS	15	Section 7.4.1
DEVICE CAPABILITY	16	Section 7.4.1
WIRELESS_ENDPOINT_COMPANION	17	Section 7.4.2

7.4.1 Device Level Descriptors

The Device descriptor describes general information about a USB device. Please refer to Section 9.6.1 of the USB 2.0 specification for a full description of the Device descriptor. The *bMaxPacketSize0* field is the only field in the standard device descriptor with a different requirement than that documented in the USB 2.0. Wireless USB requires the *bMaxPacketSize0* field to be set to FFH (see Section 4.8.1). In addition, devices that conform to this revision of the Wireless USB specification must have the value of 0250H in the *bcdUSB* field. Additional device-level information for devices is contained in the BOS descriptor, see below.

This section defines a flexible and extensible framework for describing and adding device-level capabilities to the set of USB standard specifications. As mentioned above, there exists a device descriptor, but all device-level capability extensions are defined using the following framework.

The BOS descriptor (Binary device Object Store, see Table 7-26) defines a root descriptor that is similar to the configuration descriptor, and is the base descriptor for accessing a family of related descriptors. A host can read a BOS descriptor and learn from the *wTotalLength* field the entire size of the device-level descriptor set, or it can read in the entire BOS descriptor set of device capabilities. The host accesses this descriptor using the GetDescriptor() request. The descriptor type in the GetDescriptor() request is set to BOS (see Table 7-26). There is no way for a host to read individual device capability descriptors. The entire set can only be accessed via reading the BOS descriptor with a GetDescriptor() request and using the length reported in the *wTotalLength* field.

Table 7-26. BOS Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor.
1	*bDescriptorType*	1	Constant	Descriptor type: BOS.
2	*wTotalLength*	2	Number	Length of this descriptor and all of its sub descriptors.
4	*bNumDeviceCaps*	1	Number	The number of separate device capability descriptors in the BOS.

Individual, technology-specific or generic device-level capabilities are reported via Device Capability descriptors. The format of the Device Capability descriptor is defined in Table 7-27. The Device Capability descriptor has a generic header, with a sub-type field (*bDevCapabilityType*) which defines the layout of the remainder of the descriptor. The codes for *bDevCapabilityType* are defined in Table 7-28.

Table 7-27. Format of a Device Capability Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor.
1	*bDescriptorType*	1	Constant	Descriptor type: DEVICE CAPABILITY Type.
2	*bDevCapabilityType*	1	Number	Valid values are listed in Table 7-28.
3	*Capability-Dependent*	VAR	Variable	Capability-specific format.

Device Capability descriptors are always returned as part of the BOS information returned by a GetDescriptor(BOS) request. A Device Capability cannot be directly accessed with a GetDescriptor() or SetDescriptor() request.

Table 7-28. Device Capability Type Codes

Capability Code	Value	Description
Wireless_USB	01H	Defines the set of Wireless USB-specific device level capabilities.
Reserved	00H, 02-FFH	Reserved for future use.

The following section(s) define the specific device capabilities.

7.4.1.1 Wireless USB Device Capabilities – UWB

This section defines the required device-level capabilities descriptor which must be implemented by all Wireless USB devices. Wireless USB Device Capabilities – UWB descriptors cannot be directly accessed with a GetDescriptor() or SetDescriptor() request.

Table 7-29. Wireless USB Device Capabilities on UWB Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor.
1	*bDescriptorType*	1	Constant	Descriptor type: DEVICE CAPABILITY Type.
2	*bDevCapabilityType*	1	Constant	Capability type: WIRELESS USB.
3	*bmAttributes*	1	Bitmap	Bitmap encoding of supported device level features. A value of one in a bit location indicates a feature is supported; a value of zero indicates it is not supported. Encodings are: Bit / Encoding 0: **Reserved**. Must be set to zero. 1: **P2P-DRD.** A value of one in this bit location indicates that this device is Peer to Peer DRD capable. 3:2: **Beacon Behavior.** This field encodes the beaconing behavior of the device. The encoded values are: Value / Description 00B: Reserved. 01B: Self-Beacon 10B: Directed Beacon 11B: No-Beacon 7:4: **Reserved**. Must be set to zero.

Table 7-29. Wireless USB Device Capabilities on UWB Descriptor (cont.)

Offset	*Field*	Size	Value	Description
4	*wPHYRates*	2	Bitmap	Describes the PHY-level signaling rate capabilities of this device implementation represented as a bit-mask. Bit positions are assigned to speed capabilities possible in a PHY implementation. A ‘1’ in a bit position indicates the associated data rate is supported by the device. Encodings are: **Bit** — **Data Rate (Mbps)** 0 — 53.3 **required 1 — 80 2 — 106.7 **required 3 — 160 4 — 200 **required 5 — 320 6 — 400 7 — 480 15:8 — Reserved. Must be zero. ** Required encodings must be a one (1B).

Table 7-29. Wireless USB Device Capabilities on UWB Descriptor (cont.)

Offset	Field	Size	Value	Description
6	*bmTFITXPowerInfo*	1	Bitmap	This bit mask reports the PHY transmit power levels supported by this device when transmitting on a TFI channel. See below for details. **Bit** **Encoding** 3:0 **Power Level Steps**. Specifies the number of steps from the base TFI transmit power level supported by the device. 7:4 **Step Size.** The value specifies the number of dB between supported power levels as follows: **Value** **Step Size (dB)** 0 1.0 1 1.25 2 1.5 3 1.75 4 2.0 5 2.25 6 2.5 7 2.75 8 3.0 9 3.25 10 3.5 11 3.75 12 4.0 13 4.25 14 4.5 15 4.75
7	*bmFFITXPowerInfo*	1	Bitmap	This bit mask reports the PHY transmit power levels supported by this device when transmitting on an FFI channel. The encoding of this field is identical to the *bmTFITXPowerInfo* field. See below for details.
8	*bmBandGroup*	2	Bitmap	This bit map reports which UWB band groups are supported by this implementation. See below for details.
10	*bReserved*	1	Zero	Reserved for future use, must be zero

The fields *bmFFITXPowerInfo* and *bmTFITXPowerInfo* report the PHY transmit power levels supported by the device. The fields report the step size between power levels and the number of steps supported. This information can be used in the Set WUSB Data request to set the devices' maximum transmit power for various types of packets transmitted by the device to the Wireless USB channel. The power level for data packets is set by the *bmTXAttributes Transmit Power* field in a W_{DT}CTA, see Section 5.2.1.2. The host must use only those power levels reported by this descriptor. See Section 4.10.1 for detailed information and additional requirements on TPC.

The field *bmBandGroup* is a bit mask that reports which UWB band groups are supported by this implementation. A 1B in a bit position indicates all of the bands and channels in the associated PHY bandgroup are supported (i.e. a 1B in bit position 0 indicates that all bands in band group one are supported). For this revision of the specification, all devices are required to support bandgroup one (0001H). At this time, only 5 bandgroups are defined, therefore bits [15:5] are reserved. For full details on the PHY, refer to reference [4].

7.4.2 Configuration

The configuration descriptor describes information about a specific device configuration. Please refer to Section 9.6.3 of the USB 2.0 specification for a full description of the Configuration descriptor. The descriptor is included in its entirety below (for completeness). The Wireless USB specific additions/requirements are documented in the description column of Table 7-30.

Table 7-30. Standard Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Refer to Table 9-10 in Section 9.6.3 in the USB 2.0 specification for a full description.
1	*bDescriptorType*	1	Constant	
2	*wTotalLength*	2	Number	
4	*bNumInterfaces*	1	Number	
5	*bConfigurationValue*	1	Number	
6	*iConfiguration*	1	Index	
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7: Reserved (set to one) D6: Self-powered D5: Remote Wakeup D4 Battery-powered D3...0: Reserved (reset to zero) D7 is reserved and must be set to one for historical reasons. All attributes are available for a USB 2.0 device implementation. A Wireless USB device must always set Self-powered (D6) to a one (1B). If a device configuration supports remote wakeup, D5 is set to one.
8	*bMaxPower*	1	mA	A Wireless USB device must always set this field to zero.

7.4.3 Endpoint

The purpose and function of the endpoint descriptor is the same as defined in the USB 2.0 specification, see Section 9.6.6. This section describes the requirement modifications to the endpoint descriptor required for Wireless USB. The entire descriptor from the USB 2.0 specification is repeated here for convenience, with the explicit changes for Wireless USB identified. In addition, Wireless USB defines an Endpoint Companion descriptor for endpoint capabilities required for Wireless USB, that would not fit inside the existing endpoint descriptor footprint.

A Wireless USB endpoint descriptor must not be used in a USB 2.0 (wired) device configuration.

Table 7-31. Standard Endpoint Descriptor for Wireless USB Devices

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: **Bits** **Description** 3:0 The endpoint number 6:4 Reserved and must be set to zero 7 Direction (ignored for Control endpoints) 0B OUT endpoint 1B IN endpoint

Table 7-31. Standard Endpoint Descriptor for Wireless USB Devices (cont.)

Offset	*Field*	Size	Value	Description
3	*bmAttributes*	1	Bitmap	This field describes the endpoint's attributes when it is configured using the *bConfigurationValue*. **Bits** **Description** 1:0 Transfer type. Values have the following encoding: 00B Control 01B Isochronous 10B Bulk 11B Interrupt 6:2 This field is reserved and must be set to zero if the *Transfer Type* field is **Control** or **Bulk**. If the *Transfer Type* field is **Interrupt**, then bits [6:3] are reserved and must be set to zero and bit [2] has the following meaning: 0B Normal power interrupt endpoint 1B Low power interrupt endpoint If the *Transfer Type* field is **Isochronous**, then this field has the following encoding: **Bits** **Meaning** 3:2 Synchronization type 00B No Synchronization 01B Asynchronous 10B Adaptive 11B Synchronous 5:4 Usage type 00B Data endpoint 01B Feedback endpoint 10B Implicit feedback endpoint 11B Reserved 6 Reserved. Must be zero. 7 Data packet size adjustment flag. This bit must be set to zero for Control and Isochronous endpoints. 0B Data packet size adjustment not supported 1B Data packet size adjustment supported

Table 7-31. Standard Endpoint Descriptor for Wireless USB Devices (cont.)

Offset	Field	Size	Value	Description
4	*wMaxPacketSize*	2	Number	When transfer type is bulk, control or interrupt: Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. If the transfer type is isochronous the value of this field is the logical maximum packet size. Note: If there are not software compatibility issues using the wOverTheAirPacketSize, wMaxPacketSize must be set to the same value as wOverTheAirPacketSize. Note, for all endpoints, bits [15:0] specify the maximum packet size (in bytes). Note that each endpoint transfer type has additional constraints defined that limit the valid range of values for this field. See Chapter **4** For additional details.
6	*bInterval*	1	Number	When the transfer type is bulk or control, this field is reserved and must be set to zero. For interrupt endpoints, the *bInterval* value is used as the exponent for a $2^{\text{bInterval-1}}$ value; e.g., a *bInterval* of 6 means a period of 32 (2^{6-1}) units of 128 microseconds. For wireless interrupt endpoints this value must be from 6 to 16. Note: Not all service interval bounds in this range may be achievable. See Section 4.11.2.2 for details. For isochronous endpoints the value of this field is the logical service interval. This field allows isochronous endpoints to report a different value than *bOverTheAirInterval* to software for legacy compatibility reasons. The encoding is identical to the USB 2.0 specification encoding for FS/HS isochronous endpoints. The logical service interval value must be in the range 1 to 16. If there is no need to report different values the *bInterval* and *bOverTheAirInterval*.values must be identical. A continuously scalable dynamic switching capable isochronous endpoint specifies zero for the *bInterval* value and must support any interval.

The *bmAttributes* field has several changes from the definition in the USB 2.0 specification. These are enumerated below:

- Bits [6:2] are interpreted based on the value of the *Transfer Type* field. They are reserved and must be zero if Bulk or Control.
  - Bit [2] for an Interrupt type endpoint flags whether the endpoint is a normal or low-power type endpoint (see Section 4.6) but the remaining bits [6:3] are reserved and must be set to zero.
  - For Isochronous, the purpose of bits [5:2] remains unchanged from USB 2.0.[5] Bit [6] is reserved and must be set to zero.

[5] Warning: Acceptable response timeouts for feedback endpoints may be significantly longer than wired USB 2.0.

- Bit [7] applies to all transfer types and indicates whether the endpoint supports data burst packet maximum packet size adjustments for PER see Section 4.10.2.

7.4.4 Wireless USB Endpoint Companion

Each Wireless USB endpoint described in an interface must have a Wireless USB Endpoint Companion descriptor. This descriptor contains additional endpoint characteristics that are only defined for Wireless USB endpoints. A Wireless USB Endpoint Companion descriptor for each Wireless USB endpoint is always returned as part of the configuration information returned by a GetDescriptor(Configuration) request. A Wireless USB Endpoint Companion descriptor cannot be directly accessed with a GetDescriptor() or SetDescriptor() request. There is never a Wireless USB Endpoint Companion descriptor for endpoint zero. The Wireless USB Endpoint Companion descriptor must immediately follow the endpoint descriptor it is associated with in the configuration information.

Table 7-32. Wireless USB Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes
1	*bDescriptorType*	1	Constant	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	Number	The maximum number of packets the endpoint can send or receive as part of a burst. The value is a number from 1 to 16. Refer to Sections 4.5, 4.6, 4.7 and 4.8 for more information and maximum values for each endpoint type.
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32. The maximum sequence number value used on the endpoint is *bMaxSequence* -1. Section 5.4 Details how this value is used for data bursting and Sections 4.5 Through 4.8 Provides information on constraints per transfer type.
4	*wMaxStreamDelay*	2	Number	For isochronous endpoints this field is a value from 1 to 65535 indicating the maximum amount of delay in 128 microsecond units that can be supported by the stream. The endpoint must provide exactly the amount of buffering to support this delay. An IN endpoint must be able to store this amount of data before having to discard data. An OUT endpoint must be able to fill its buffering (except for storage for less than a maximum size burst) before it is allowed to NAK. Refer to the dataflow Section 4.11 for more information. For interrupt, bulk, and control endpoints this field is reserved and must be zero.

Table 7-32. Wireless USB Endpoint Companion Descriptor (cont.)

Offset	*Field*	Size	Value	Description
6	*wOverTheAirPacketSize*	2	Number	If the transfer type is isochronous: Maximum packet size this endpoint is capable of sending or receiving over the air when this configuration is selected. If the transfer type is bulk, interrupt, or control this field is reserved and must be set to zero. See Section 4.8.4.
8	*bOverTheAirInterval*	1	Number	This field is the interval for polling the isochronous endpoint. The *bOverTheAirInterval* value is used as the exponent for a $2^{bOverTheAirInterval-1}$ value; e.g., a *bInterval* of 6 means a period of 32 (2^{6-1}) units of 128 microseconds. See Section 4.7.2 A continuously scalable dynamic switching capable isochronous endpoint specifies zero for the *bOverTheAirInterval* value and must support any interval. If the transfer type is bulk, interrupt, or control, this field is reserved and must be set to zero.

Table 7-32. Wireless USB Endpoint Companion Descriptor (cont.)

Offset	*Field*	Size	Value	Description
9	*bmCompAttributes*	1	BitMap	Attributes of the endpoint companion descriptor: **Bits** **Description** 1:0 Dynamic Switching: If the *Transfer Type* field is Isochronous, then this field has the following encoding: 00B No dynamic switching 01B Dynamic switching 10B Dynamic switching and continuously scalable 11B Reserved This field indicates whether the interface containing the endpoint supports dynamic switching and whether it is continuously scalable. If the endpoint supports dynamic switching (but is not continuously scalable) the interface containing the endpoint can be dynamically switched between alternate settings using the process described in Section 4.10.6. If the endpoint is continuously scalable it can support any packet size up to the reported *wMaxPacketSize* and any interval. Device specific out of band mechanisms must be used to change the maximum packet size or interval. See Section 8.4.4.3 for an example of the use of continuously scalable endpoints. A continuously scalable dynamic switching endpoint does not receive a bandwidth reservation when its associated interface is selected. This field is reserved and must be set to zero when the *Transfer Type* field is Bulk, Control or Interrupt. 7:2 Reserved and must be set to zero

Note: refer to Section 4.10.6 for behavioral requirements for Dynamic Switching.

For isochronous endpoints, *bMaxBurst* and *wOverTheAirPacketSize* are used to reserve the average bus time in the schedule, required for the data payloads each *bOverTheAirInterval*. This value should indicate the average actual bandwidth required by the endpoint if no errors occur. Appropriate additional opportunities for retries are automatically scheduled by the host.

7.4.5 Security-Related Descriptors

This section describes the descriptors that are used by USB Security.

7.4.5.1 Security Descriptor

The Security Descriptor describes the Security capabilities of the device. The capabilities of the host are never advertised to the device. The host will select the appropriate device mode.

The Security Descriptor functions similarly to a Configuration Descriptor. It serves as a general container for the other descriptors that describe the device security properties in detail. It is done as a container so that new descriptors may be added as new encryption methods are employed or breaches are repaired.

The Security Descriptor and its contained payload is directly addressable using the Get Descriptor request, specifying the descriptor type as SECURITY. This is done so that the Security Descriptor information can be enumerated by a host in plain text without revealing any other descriptor information.

Key descriptors are not returned as part of the Security descriptor. Keys referenced in the Encryption Type descriptors can be read via the Get Key request.

Table 7-33: Security Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Number of bytes in this descriptor, including this byte
1	*bDescriptorType*	1	Constant	Descriptor Type: SECURITY Descriptor
2	*wTotalLength*	2	Number	Length of this descriptor and all sub-descriptors returned
4	*bNumEncryptionTypes*	1	Number	Number of supported encryption types

7.4.5.1.1 Encryption Type Descriptor

The Security Descriptor payload can contain multiple Encryption Type descriptors. It should contain one for each mode supported, except for UNSECURE.

The field *bEncryptionType* selects one of the encryption types defined in Table 7-35. The field *bEncryptionValue* specifies the value that should be used with Set Encryption in order to enable this type of encryption.

The device indicates that it can use this encryption type for New Connection authentication by supplying a valid Key Index in the *bAuthKeyIndex* field. This Key Index must reference a valid device key, i.e. Originator=Device, Type=Authentication.

The host enumerates the device's encryption type descriptors to determine what encryption suites the device supports. It does this by examining the *bEncryptionType* field of the descriptor to identify the type of security. When the desired descriptor is located, the host uses the *bEncryptionValue* field from the descriptor as the parameter to the Set Encryption request.

Table 7-34: Encryption Type Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Number of bytes in this descriptor, including this byte
1	*bDescriptorType*	1	Constant	Descriptor Type: ENCRYPTION TYPE Descriptor
2	*bEncryptionType*	1	Number	Type of encryption (See Table 7-35)
3	*bEncryptionValue*	1	Number	Value to use with Set Encryption
4	*bAuthKeyIndex*	1	Key Index	Non-zero if this encryption type can be used for New connection authentication. In this case the value specifies the Key Index to use for authentication.

Table 7-35: USB Encryption Types

Encryption Types	Value	Description
UNSECURE	0	No encryption enabled
WIRED	1	Virtual encryption provided by the wire
CCM_1	2	AES-128 in CCM mode
RSA_1	3	RSA-3072 encryption with SHA-256 hashing
Reserved	4-255	Reserved for future use

7.4.5.2 Key Descriptor

Key Descriptors are used to contain keys during key distribution.

Table 7-36: Key Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Number of bytes in this descriptor, including this byte
1	*bDescriptorType*	1	Constant	Descriptor Type: KEY Descriptor
2	*tTKID*	3	Number	The TKID value associated with this key, if any.
5	*bReserved*	1	Byte	Reserved, must be zero
6	*Key data*	Var	Bytes	The actual key data

7.5 Wireless USB Channel Information Elements

The information elements listed in this section are part of the information and control mechanisms for the Wireless USB Channel as controlled by the host. These information elements are transmitted by a host in MMC packets.

The general structure of the data payload portion of an MMC is illustrated in Figure 7-4 and detailed in Table 7-37. The information elements in the MMC are channel time allocations (for device notifications or Endpoint data streams) or host to device control information. Individual information elements may be targeted to a particular device or may be broadcast to the entire Wireless USB Cluster. MMC packets are transmitted using secure packet encapsulation with the Encryption Offset field in the security header set to the length of the MMC payload (packet (frame) length (from PHY Header) – Secure Packet overhead (20 bytes)). This results in the packet being transmitted in plain text and the secure packet encapsulation provides authentication of the packet. This approach allows the host to use a single MMC to conduct Wireless USB transactions to devices in both the **Authenticated** and **UnAuthenticated** states. See Section 7.1 for limitations on data communications with devices in the **UnAuthenticated** state.

(LSB) (MSB)

2	1	2	2	3	VAR	VAR	…	VAR
WUSB App Code (0100H)	MMC Code (01H)	Next MMC Time	Reserved	WUSB Channel Time Stamp	IE[0]	IE[1]	…	IE[n]

Figure 7-4. General Structure of an MMC Control Packet

The MAC Layer Header fields in the MMC packet are set to indicate an Application-defined control packet (frame).

Table 7-37. Detail Field Definition of MMC Packet

Offset	Field	Size	Value	Description
0	*Application Identifier*	2	Constant	Wireless USB [WUSB (0100H)]
2	*Type*	1	Constant	MMC Command Type: (01H)
3	*NextMMC Time*	2	Number	Units are in micro-seconds. This is the number of microseconds from the beginning of this MMC to the beginning of the next MMC packet.
5	*Reserved*	2	Number	This field is reserved and should be set to zeros.
7	*WUSB Channel Time Stamp*	3	Bitmap	This is a timestamp provided by the host based on a free-running timer in the host. The value in this field indicates the value of the host free running clock when MMC transmission starts. The accuracy requirement of the time stamp (to the host clock) is +/- 40 nanoseconds. The time stamp is formatted into two fields as follows: Bits — Description 6:0 — **Microsecond Count.** The microsecond count rolls over after 125 microseconds. Each time it rolls over the 1/8th Millisecond Count is incremented. 23:7 — **1/8th Millisecond Count.** This counter increments every time the Micro-second counter wraps.
10	*IE[0-n]*	Var	MMC IE	Array of information elements. There must be at least one IE.

The information elements (IEs) in an MMC are called Wireless USB Channel information elements and include protocol time slot allocations (for Data and Handshake Phases of Wireless USB transactions), DNTS declarations and host information and control information. A summary of the Wireless USB Channel IEs are provided in Table 7-38. The format of each Wireless USB Channel IE is defined in sections below, except for the IEs used to maintain the Wireless USB transaction protocol. These IE definitions are in the Protocol Chapter, see Section 5.2.1. A number of the IEs documented below utilize an array structure for addressing 1 to N devices in the same IE. A host must not include more than 4 elements in any array-based IE. The exception to this rule is the WCTA_IE, where the host is required to limit the number of W_XCTA blocks to 32.

Table 7-38. Wireless USB Channel IE Identifiers

IE Identifier	IE Name	Description
00H-7FH		Reserved for future use.
80H	WCTA_IE	Wireless USB Channel Time Allocation Information Element. May contain one or more channel time allocations for device endpoints to listen or transmit. See Section 5.2.1.
81H	WCONNECTACK_IE	Wireless USB Connect Acknowledge. See Section 7.5.1.
82H	WHOSTINFO_IE	Specific information about the host that controls the Wireless USB Channel. See Section 7.5.2.
83H	WCHCHANGEANNOUNCE_IE	This information element is used to notify cluster members that the host is moving the Wireless USB channel to a different PHY channel. See Section 7.5.3.
84H	WDEV_DISCONNECT_IE	Used by host to inform a specific device that it is being disconnected. See Section 7.5.4.
85H	WHOST_DISCONNECT_IE	Used by host to inform the cluster that all connected devices are being disconnected. See Section 7.5.5.
86H	WRELEASE_CHANNEL_IE	Used by the host to instruct its cluster members to transmit a control packet to release the remainder of an MAC Layer channel reservation. See Section 7.5.6.
87H	WWORK_IE	Used by the host in response to a Sleep notification. See Section 7.5.7.
88H	WCHANNEL_STOP_IE	Used by the host to notify cluster members that the Wireless USB channel is being stopped. See Section7.5.8.
89H	WDEV_KEEPALIVE_IE	Used by the host to force a device to 'check-in' in order to keep trust fresh or to detect implicit disconnects. See Section 7.5.9.
8AH	WISOCH_DISCARD_IE	Used by a host to indicate that it has discarded data intended for an isochronous OUT function endpoint. See Section 7.5.10.
8BH	WRESETDEVICE_IE	Used by the host to cause a device to perform a hard reset operation. See Section 7.5.11.
8CH	WXMIT_PACKET_ADJUST_IE	Used by the host to inform all devices with Transmit Packet feature enabled of a new transmit adjustment value. See Section 7.5.12.
8DH-FFH		Reserved for future use.

The first field of a channel information element is *bLength* and the second field is *IE_Identifier*. The value of *bLength* is the length of the information element, including the *bLength* and *IE_Identifier* fields.

There are only a few ordering rules for MMC_IEs. The WISOCH_DISCARD_IE (if present) must be first, followed by the WCTA_IE. The order of most other MMC IEs is arbitrary. The WHOSTINFO_IE when present must be the last IE in an MMC.

Note, device implementations must ignore (and skip) information elements that are not recognized. Future revisions of this specification may define additional information elements. The general format of information elements includes a *bLength* field followed by an *IE Identifier* field provides sufficient information to a device to be able to skip information elements it does not recognize.

7.5.1 Wireless USB Connect Acknowledge IE

The Connect Acknowledge IE is used to send one or more specific acknowledgements to devices that have requested association via the *DN_Connect* device notification (see Section 7.6.1). The format of this IE is illustrated in Figure 7-5. It is basically an array of connection acknowledgment blocks (ConnectAck), one for each connect request received.

(LSB) (MSB)

1	1	18	18		18
bLength	IE Identifier = WCONNECTACK_IE	ConnectAck[0]	ConnectAck[1]	…	ConnectAck[n-1]

Figure 7-5. Format of a Wireless USB Connect Acknowledge IE

The purpose of this information element is to let the requesting device know that its connect request has been received and to also give it a new device address, that will be used for the Authentication and Authorization Stage. Table 7-39 illustrates the format of a ConnectAck block.

Table 7-39. ConnectAck Block Format

Offset	Field	Size	Value	Description
0	*CDID*	16	Number	Connection Device ID. This field contains the CDID value from the device's *DN_Connect* notification.
16	*bDeviceAddress*	1	Number	This field contains a new device address for the associating device.
17	*bReserved*	1	Constant	Reserved. Must be set to zero.

Once a device has issued a *DN_Connect* notification it then waits for an MMC from the host that contains Connect Acknowledgement IEs. It looks for a ConnectAck block with a *CDID* field that contains the same CDID value sent in its *DN_Connect* notification. When a match is found, the device will set its device address to the value of the *bDeviceAddress*. The host must provide at least 2ms of address set-up relaxation from approximately the start of transmit of the ConnectAck to the start of Wireless USB Transactions addressed to the device's Default Endpoint at the assigned *bDeviceAddress*.

- **Stop Retransmission Condition:** a host will remove a ConnectAck block for a specific device from the Connect Acknowledge IE after either of the two events occur:
    - The host has observed the associated device responding to control transfers to the Default Control Pipe addressed to the assigned device address (i.e. value of *bDeviceAddress*).
    - The host has ceased attempts to talk to the device's Default Control Pipe at the assigned device address.

7.5.2 Wireless USB Host Information IE

The Wireless USB Host Information Element is used by a host to annotate a Wireless USB channel with its unique name. The unique name is the Connection Host ID (CHID, see Section 6.2.10.1). The format of this information element is detailed in Table 7-40. This Information Element should be located in an MMC after all WCTA_IEs.

Table 7-40. Host Information Element

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this information element (in bytes), including this field.
1	*IE Identifier*	1	Constant	Information IE Type: WHOSTINFO_IE for Host information element.
2	*bmAttributes*	2	Bitmap	Host-specific current capabilities information. __Bit__ __Meaning__ 1:0 **Connection Availability.** This field indicates to Un-connected devices what types of associations the host is available for at this time. The encodings are as follows: 00B **Reconnect only.** Host is available only for reconnect notifications. 01B **Limited.** Host is only available for connections and reconnections. 10B **Reserved.** 11B **ALL.** Host is open for connect, reconnect and new connect notifications. 2 **P2P-DRD Capable.** A one in this bit location indicates the host is Peer to Peer DRD capable. A zero in this bit location indicates the host is not Peer to Peer DRD capable (see Section 4.17). 5:3 **MAC Layer Stream Index.** Devices must ensure that all packet transmissions have the value of this field in the Stream Index field of the MAC Layer Header (see Section 5.6). 15:6 **Reserved.** Must be set to zero.
4	*CHID*	16	Number	Connection Host ID, which serves as a Unique Host ID. The device checks this value when locating a particular host.

Note, connect includes any *DN_Connect* device notification where the *New Connection* bit is a zero. The CHID is defined and rules for generating it are detailed in Section 6.2.8.

This information element is used by provisioned devices to locate the Wireless USB Channel of a specific host. A host is not required to include this information element in all MMCs. It must include the IE in at least three MMCs per superframe when the total number of MMCs in a superframe is greater than 3. Otherwise, it must include this IE in all MMCs.

- **Stop Retransmission Condition:** the rules for including this IE in the Wireless USB Channel are stated above.

7.5.3 Wireless USB Channel Change Announcement IE

The Channel Change Announcement IE is used by a host to announce the Wireless USB channel time at which devices must begin listening to a different PHY channel for continuation of the current Wireless USB Channel transmissions. The format of this information element is detailed in Table 7-41.

Table 7-41. Channel Change Announcement Information Element

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this information element (in bytes), including this field.
1	*IE Identifier*	1	Constant	Information IE Type: WCHCHANGEANNOUNCE_IE for Channel Change Announcement information element.
2	*bNewPHYChannelNumber*	1	Number	The PHY channel number where the host is moving the Wireless USB Cluster.
3	*SwitchTime*	3	Timestamp	The time at which the Wireless USB channel will switch to the alternate PHY channel specified in *bNewPHYChannelNumber*.

The allowable values of *bNewPHYChannelNumber* depend on the PHY channels supported by the host and the members of its Wireless USB Cluster. Refer to Section 4.10.4 for rules for *SwitchTime* values and other operational requirements for devices. Refer to 5.6 for a summary about valid values for the *bNewPHYChannelNumber* field.

- **Stop Retransmission Condition:** the host must cease transmitting this IE after the channel switch has completed.

7.5.4 Wireless USB Device Disconnect IE

The Device Disconnect IE is used to send Disconnect notifications to one or more specific devices. The format of this IE is illustrated in Table 7-42.

Table 7-42. Format of a Wireless USB Device Disconnect IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 2+N (+ optional 1) bytes, where N is the total number of *bDeviceAddresses* in this IE
1	*IE_Identifier*	1	Constant	WDEV_DISCONNECT_IE
2	*bDeviceAddress*	N	Array	Array of device addresses (each one byte)
N+2	*bReserved*	1	Constant	This is a pad byte that must be added by the host to make the total size of the IE instance be an even multiple of 2 bytes. The value the host puts into this field, when it exists, must NOT match any currently active assigned device address

The purpose of this information element is to allow the host to notify one or more devices that it is being disconnected.

- **Stop Retransmission Condition:** the host will cease transmitting this IE after at least 100 ms have elapsed and it has transmitted at least 3 MMCs that include this IE for the device(s).

7.5.5 Wireless USB Host Disconnect IE

The Host Disconnect IE is used to send Disconnect notifications to all devices in the cluster. The format of this IE is illustrated in Table 7-43.

Table 7-43. Format of a Wireless USB Host Disconnect IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 2 bytes
1	*IE_Identifier*	1	Constant	WHOST_DISCONNECT_IE

The purpose of this information element is to allow the host to notify all devices that they are being disconnected.

- **Stop Retransmission Condition:** the host will cease transmitting this IE after at least 100 ms have elapsed and it has transmitted at least 3 MMCs that include this IE.

7.5.6 Wireless USB Release Channel IE

The Release Channel IE is used to inform non-Wireless USB devices that the host is making the remainder of the current MAC Layer channel reservation block available to them for data communications. This means the host will not be using the channel time for Wireless USB channel data communications. The format of this IE is illustrated in Figure 7-6.

(LSB) (MSB)

1	1	6	6	…	6
bLength	IE Identifier = WRELEASE_CHANNEL_IE	UDRB[0]	UDRB[1]	…	UDRB[n]

Figure 7-6. Format for Release Channel IE

The *bLength* field includes the total length of the Release Channel Time information element, including the *bLength* field. This information element is comprised of one or more Unused DRP Release Blocks (UDRB). Each UDRB allocates Wireless USB channel time for a device to transmit a MAC Layer-defined UDR control packet. Note that a UDR control packet has only a MAC Header and no payload. See reference [3] for details. The format of a UDR Block is illustrated in Table 7-44.

Table 7-44. Unused DRP Release Block for WRELEASE_CHANNEL_IE

Offset	Field	Size	Value	Description
0	*wStart*	2	Number	Wireless USB channel time at which the addressed device must transmit a UDR control packet.
2	*wDurationValue*	2	Number	This is a timestamp provided by the host. The device places significant bits of this field into the Duration field in the MAC Header.
4	*bDeviceAddress*	1	Number	Device address of a device that should transmit the UDR packet at *wStart* time.
5	*bReserved*	1	Constant	Reserved. Must be set to zero.

The device whose Wireless USB cluster address matches the value of *bDeviceAddress* must begin transmitting a UDR control frame when its internal clock matches *wStart*. The interpretation of *wStart* is the same as that documented in Section 5.3 for W_XCTAs. The device must format the UDR as specified in Table 7-45 and it must use the *wDurationValue* provided in the associated UDRB in the *Duration* field of the UDR packet.

The host must ensure that the *wDurationValue* field values are consistent with the requirements of the MAC Layer specification. The host uses the inter-slot times defined in Section 5.3.2 for providing guard times between UDR transmission slots.

Table 7-45. Field Contents for a UDR Packet (MAC Layer Packet Only)

Offset	Name	Size	Value	Description
0	*Frame Control*	2	Bitmap	This field is encoded with the following values: **Bits** / **Name** / **Value** 2:0 Protocol Version 000B 3 Secure 0B 5:4 ACK Policy 00B 8:6 Frame Type 001B (Control) 12:9 Frame Subtype 0101B (UDR) 13 Retry 0B 15:14 Reserved 00B
2	*DestAddr*	2	Number	Must be set to the host's *DevAddr* (i.e. value is taken from the *SrcAddr* field of the MMC's MAC Header).
4	*SrcAddr*	2	Number	The device's *DeviceAddress*.
6	*Sequence Control*	2	Bitmap	This field must be set to the value of 0000H:
8	*Access Information*	2	Bitmap	This field is encoded with the following values: **Bits** / **Name** / **Value** 13:0 Duration *wDurationValue* 14 More Data 0B 15 Access Method 1B

All device implementations must implement this feature. The UDR packet must be transmitted at maximum available transmit power.

Note, that the host must also transmit a properly annotated UDR packet. The host may transmit it's UDR either before or after the channel time described and allocated by this IE.

- **Stop Retransmission Condition:** the host will transmit this IE only once.

7.5.7 Wireless USB Work IE

The host includes a Work IE in MMCs in response to a Sleep notification from one or more devices. Section 4.16 in the Data Flow chapter describes the full operation.

Table 7-46. Format of a Wireless USB Work IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 2+N (+ optional 1) bytes, where N is the total number of *bDeviceAddresss* in this IE
1	*IE_Identifier*	1	Constant	WWORK_IE
2	*bDeviceAddress*	N	Array	Array of Device Addresses (each one byte) This field contains the Device Address identifying which device this Work IE is a response to. **Bits** / **Description** 6:0 Device Address. The device address of the device the host is responding to. 7 Work Pending. This bit is a 1 if there is work pending for the device. The bit is 0 otherwise.

Table 7-46. Format of a Wireless USB Work IE (cont.)

Offset	Field	Size	Value	Description
N+2	*bReserved*	1	Constant	This is a pad byte that must be added by the host to make the total size of the IE instance be an even multiple of 2 bytes. The value the host puts into this field, when it exists, must NOT match any currently active assigned device address

- **Stop Retransmission Condition:** the host will transmit this IE for at least 3 MMCs.

7.5.8 Wireless USB Channel Stop IE

The host includes a Channel Stop IE in MMCs before stopping a USB channel. Section 4.16 in the Data Flow chapter describes the full operation.

Table 7-47. Format of a Wireless USB Channel Stop IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 6 bytes
1	*IE_Identifier*	1	Constant	WCHANNEL_STOP_IE
2	*bmAttributes*	1	Bitmap	This field contains attributes for the Channel Stop IE: __Bit__ __Description__ 0 Remote Wakeup: Value of 1 indicates that the host will be 'polling' for Remote Wakeup. 0 otherwise. 7:1 Reserved
3	*StopTime*	3	Timestamp	The time at which the Wireless USB channel will stop.

- **Stop Retransmission Condition:** the host will remove this IE from the Wireless USB channel when it is no longer in need of polling for remote wake notifications. If the host is not transitioning to a channel stop or is not open for remote wake notifications, then it must not include this IE in the Wireless USB channel.

7.5.9 Wireless USB Device Keepalive IE

The Device Keepalive IE is used to direct one or more Wireless USB cluster members to begin transmitting *DN_Alive* notifications. The purpose of this IE is described in Section 4.14. The frequency and placement of this IE in MMCs is host-dependent.

Table 7-48. Format of a Wireless USB Keepalive IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 2+N (+ optional 1) bytes, where N is the total number of *bDeviceAddresses* in this IE
1	*IE_Identifier*	1	Constant	WDEV_KEEPALIVE_IE
2	*bDeviceAddress*	N	Array	Array of device addresses (each one byte)
N+2	*bReserved*	1	Constant	This is a pad byte that must be added by the host to make the total size of the IE instance be an even multiple of 2 bytes. The value the host puts into this field, when it exists, must NOT match any currently active assigned device address

- **Stop Retransmission Condition:** the host will remove a device address from this IE once it has successfully received a *DN_Alive* notification from that device.

7.5.10 Wireless USB Isochronous Packet Discard IE

The Isochronous Packet Discard IE is used by the host to indicate to an isochronous OUT endpoint that the host has discarded one or more data packets. The format of this IE is illustrated in Table 7-49.

Table 7-49 Format of Isochronous Packet Discard IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE: 14 bytes
1	*IE_Identifier*	1	Constant	WISOCH_DISCARD_IE
2	*bDiscardID*	1	Number	An ID number for the discard IE
3	*bDeviceAddress*	1	Number	Device Address
4	*bmAttributes*	1	Bitmap	This bitmap has the following encoding **Bit** **Value** **Description** 3:0 Variable USB Endpoint Number 7:4 Zero Reserved
5	*bFirst ReceiveWindow Position*	1	Number	The sequence number of the first position in the *bmDeviceReceiveWindow* field.
6	*wNumber Discarded Packets*	2	Number	The number of discarded packets
8	*wNumber Discarded Segments*	2	Number	The number of discarded isochronous segments.
10	*bmDevice Receive Window*	4	Bitmap	Bitmap indicating the sequence numbers that the host expects to be active in the device receive window. The dataflow chapter explains how the host calculates the expected receive window when discards occur.

The Isochronous Packet Discard IE specifies the number of isochronous data packets and segments that the host has discarded and the device receive window now expected by the host. The Isochronous Packet Discard IE must be included before any W_XCTAs for the same endpoint in the MMC. The endpoint uses the information in the Isochronous Packet Discard IE to update its receive window. The value of *bDiscardID* must be incremented by the host for each new Isochronous Packet Discard IE that it transmits. The use of the Isochronous Packet Discard IE and additional requirements are described in more detail in Section 4.11.7.3.

- **Stop Retransmission Condition:** the host will remove this IE from the Wireless USB channel when it observes that the endpoint has received the IE.

7.5.11 Wireless USB Reset Device IE

The Reset Device IE is used by a host to cause a device to perform a full ‘reset’ operation. Reset in this context is intended to be equivalent to a power-on-reset.

Table 7-50 Format of Reset Device IE

Offset	*Field*	Size	Value	Description
0	*bLength*	1	Constant	The size of this IE.
1	*IE_Identifier*	1	Constant	WRESETDEVICE_IE
2	*CDID*	16xN	Number	Array of Connection Device IDs (each one 16 bytes). Each array element contains the CDID value from the device’s *DN_Connect* notification.

Stop Retransmission Condition: the host will include this IE for a device in the Wireless USB channel for 6 MMCs.

7.5.12 Wireless USB Transmit Packet Adjustment IE

The Transmit Packet Adjustment IE is used by a host to direct all devices appropriately enabled for transmit packet to change their transmit adjustment value.

Table 7-51. Format of a Wireless USB Transmit Adjustment IE

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this information element (in bytes), including this field.
1	*IE Identifier*	1	Constant	Information IE Type: WXMIT_PACKET_ADJUST_IE
2	*bTransmitAdjustment*	1	Number	Number of additional microseconds (above 64K) that a DBD should include in the repeat interval when doing directed packet transmission.
3	*bReserved*	1	Constant	Reserved, must be zero.

On receipt of this IE, a device is required to apply the new adjustment value after the next Transmit Time occurs.

- **Stop Retransmission Condition:** the host must transmit this IE for at least 3 MMCs.

7.6 Device Notifications

Wireless USB provides analogs to USB 2.0 device signaling events such as connect, disconnect, etc. This class of information exchange is characterized as short (small), point-to-point, device to host (upstream), infrequent and asynchronous.

Wireless USB defines a method of device initiated data communications called Device Notifications. Device notifications are by definition: infrequent, asynchronous, small bits of data that a device issues to its host. The Device Notification mechanism is not intended to be used for large information exchanges, so by specification, the data payloads of device notification messages are limited to 32 bytes. Note this data payload maximum size applies to the size of the *Notification specific* field documented below in Table 7-53 (i.e. the 32 byte payload does not include the *rWUSBHeader* or *bType* fields). Device Notification messages only occur during time slots allocated by a host. These time slots are called Device Notification Time Slots (DNTS). A host is not allowed to transmit during a DNTS. All device notification packets must be transmitted using the transmit power indicated by the value of the *bTxNotificationTransmitPower* field in the WUSB Data (see Section 7.3.1.6).

All device notification packets transmitted by devices are directed to the host. All device notification packets must have the standard Wireless USB Header field as the first portion of the packet frame payload. The contents of this header are illustrated in Table 7-52. In summary, all fields in the Wireless USB Header (for notification packets) are set to fixed values.

Table 7-52. Common Wireless USB Header Contents Rules for Device Notification Packets

Offset	Field	Size	Value
0	*bmAttributes*	1	This field has the following field settings: **Bits** **Value** 3:0 0000B (Endpoint Number) 6:4 DN (Packet PID) 7 Zero
1	*bmStatus*	1	00H

Device notification packets have the format illustrated in Table 7-53. They include the standard Wireless USB header followed by a type field (*bType*). The value of the *bType* field determines the actual length and format of the bytes following the *bType* field.

Table 7-53. Format of a Device Notification Packet

Offset	Field	Size	Value	Description
0	*rWUSBHeader*	2	Record	See Table 7-52
2	*bType*	1	Number	See Table 7-54
3	*Notification specific*	Var	Number	The message portion is dependent on the value of the *bType* field.

Table 7-54 lists the Device Notification Message types. The table includes the value that must be used in the *bType* field of a device notification packet and also includes the valid device state where the notification may be transmitted by the device. Several device notifications are valid in the same device state. For cases where a device state has more than one allowable device notification message, Table 7-55 lists the notification message priority relative to the device state. A lower value designates a higher priority. When a device has multiple device notifications ready at the same time, it must send the highest priority notification message pending. For example, if a device has simultaneous pending *DN_EPRdy* and *DN_MASAvailableChange* notifications, it must send the *DN_EPRdy* message first because it has a higher priority.

Table 7-54. Device Notification Message Types

Name	Value	Valid Device State	Description
N/A	00H	N/A	Reserved
DN_Connect	01H	UnConnected Reconnecting Device Asleep	Connect and Reconnect notification (Section 7.6.1)
DN_Disconnect	02H	Authenticated Device Asleep	Disconnect device (explicit disconnect) (Section 7.6.2)
DN_EPRdy	03H	Authenticated UnAuthenticated	Device Endpoints Ready (Section 7.6.3)
DN_MASAvailChanged	04H	Authenticated	Device's MAS availability information has changed (Section 7.6.4)
DN_RemoteWakeup	05H	Device Asleep	Notification sent to wake up a sleeping host (Section 7.6.6)
DN_Sleep	06H	Authenticated Device Asleep	Notification that device is going into a lower power state (Section 7.6.5)
DN_Alive	07H	Authenticated Device Asleep	This notification is the fall-back response to a Keepalive IE (Section 7.6.7)

Table 7-55. Device Notification Message Priority List

Device State	Priority	Device Notification
UnConnected	1	DN_Connect
UnAuthenticated	1	DN_EPRdy
Authenticated	1	DN_Disconnect
	2	DN_MASAvailabilityChanged
	3	DN_EPRdy
	4	DN_Sleep
	5	DN_Alive
Reconnecting	1	DN_Connect
Device Asleep	1	DN_Disconnect
	2	DN_Sleep
	3	DN_RemoteWakeup
	4	DN_Alive

The access method for devices to transmit during a DNTS is based on Slotted Aloha. Refer to Section 5.2.1.3 for details. Note, in Section 5.1 is the requirement that all Device Notifications use secure packet encapsulation, unless specifically noted otherwise.

7.6.1 Device Connect (DN_Connect)

Before a device can communicate with a host, it must first establish a connection to that host. At other times, communications failures may require an existing connection to be reestablished. The method for initiating a connect, or reconnect event to a host is via the *DN_Connect* Device Notification. In order to send this device notification, the device must set its Device Address to the *Unconnected_Device_Address* (FFH). Once it has found the host of interest (see Section 4.13) the device will watch MMCs, waiting for a DNTS time slot which has the appropriate permissions set to allow a connect notification of the type this device needs to make (e.g. reconnect or new connect). Once a DNTS with the correct permissions on the intended host is identified, the device will transmit a *DN_Connect* device notification during the selected DNTS. Table 7-56 illustrates the format of the *DN_Connect* device notification packet.

Table 7-56. DN_Connect Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_Connect* for a Wireless USB Associate notification.
3	*bmAttributes*	2	Bitmap	This field contains attributes for the specific device which the host requires to complete the association process. The format of this field is: **Bit** — **Description** 7:0 — **Previous Device Address.** 8 — **New Connection.** This must be set to a one when the device is attempting a 'New' connection. It must be set to zero for all other connection notifications. This bit is ignored if Previous Device Address has a value other than zero. 10:9 — **Beacon Behavior**. This field encodes the beaconing behavior of the device. The encoded values are: **Value** — **Description** 00B — **Reserved.** 01B — Self-Beacon 10B — Directed Beacon 11B — No-Beacon 15:11 — **Reserved.** Must be set to zero.
5	*CDID*	16	Number	Connection Device ID

When the *bmAttributes.New Connection* field is set to a one, then the *bmAttributes.Previous Device Address* field must be set to 00H. Otherwise, *bmAttributes.Previous Device Address* is set to the last address explicitly assigned to the device by this host.

The intent of the *CDID* field is to allow a host to be able to uniquely discriminate this device connect/reconnect request from all other devices sending *DN_Connect* notifications to the same host at the same (relative) time (e.g. they all use the same device address (*UnConnected_Device_Address*)). Refer to Section 6.2.10.1 for rules on how a device generates the CDID value.

- **Maximum Retransmit Rate:** a device should retransmit this notification no more frequently than three per 100 milliseconds.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it observes a WCONNECTACK_IE that has a ConnectAck block with a CDID field that matches the device's CDID. Device will also cease transmission attempts of this notification after a *TrustTimeout* numbers of seconds have elapsed from the first transmission attempt.

7.6.1.1 Connect Request

When a device wants to initiate a connect event, it sends a *DN_Connect* request to the host, specifying a *bmAttributes.Previous Device Address* value of zero (0). The device may optionally set *bmAttributes.New Connection* field. When the *bmAttributes.New Connection* field is set to a one, then the *bmAttributes.Previous Device Address* field must be set to 00H. Otherwise, *bmAttributes.Previous Device Address* is set to the last address explicitly assigned to the device by this the host. Refer to Section 6.2.10.3 for rules about how to generate a CDID when the *New Connection* flag bit has a value of one. A connect request transmitted from a device in the **UnConnected** device state must not use secure packet encapsulation.

7.6.1.2 Reconnect Request

When a device wants to initiate a reconnect event, it sends a *DN_Connect* request to the host specifying its current USB device address in the *bmAttributes.Previous Device Address* field. A reconnect request must always use secure packet encapsulation.

7.6.2 Device Disconnect (DN_Disconnect)

The USB 2.0 wired model for disconnecting a device is via the user explicitly disconnecting the device from a wired USB port. Wireless USB provides emulated support for this required model via a *DN_Disconnect* device notification.

Table 7-57. Wireless USB Device Disconnect Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_Disconnect* for a Wireless USB Disconnect notification.

A device disconnect can be initiated by either the device (via a *DN_Disconnect* notification) or the host (via Disconnect IEs included in MMCs). A device initiates a disconnection by sending a Device *DN_Disconnect* notification to the host during a DNTS. In response to a *DN_Disconnect* notification or the host stack initiating disconnection, the host will include a WDEV_DISCONNECT_IE, targeting the requesting device, in MMCs transmitted to the cluster. See Section 4.14 for operational details.

- **Maximum Retransmit Rate:** a device should retransmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it observes a WDEV_Disconnect_IE that includes its device address, or it completes three tries of the notification.

7.6.3 Device Endpoints Ready (DN_EPRdy)

Whenever a device does not have data or space available, it will respond to a transaction request with a flow control response (see Section 5.5.4). The host response to a flow control event is to remove the endpoint from the active list of endpoints being serviced during the Wireless USB reservation time. A device must use a *DN_EPRdy* notification to signal the host that one or more endpoints are ready to resume data streaming. Note, these notifications are only used for asynchronous type Endpoints (i.e. Control and Bulk).

Table 7-58. Wireless USB Device Endpoints Ready Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_EPRdy* for a Wireless USB Endpoints Ready device notification.
3	*bLength*	1	Number	This field contains the number of endpoint ready elements that are present in this notification.

Table 7-58. Wireless USB Device Endpoints Ready Notification Format (cont.)

Offset	Field	Size	Type	Description
4	*bEPsReadyArray*	Var	Array	This is an array of endpoint ready elements where each element identifies the endpoint address (number and direction) that is ready for data transfer. The format of an element is: **Field** / **Size** / **Description** Endpoint Address / 1 / This is a standard Endpoint address, where bit [7] is the Direction (0 = OUT; 1 = IN) and bits [3:0] are the endpoint number. Bits [6:4] must be zero. Buffers Available / 4 / This is a bit vector with the same information as the bvAckCode and bvDINAck fields.

This message allows the device to bundle multiple endpoint ready indications into a single device notification. The device must include only those endpoints that previously issued flow control events and have recently transitioned to the 'ready' state.

When the host sees the notification it must resume sending transactions to the endpoints within 50 milliseconds, when the host has data or buffer space available. Devices transmit this notification only when they have Endpoints that have previously issued flow-control events. If devices transmit a *DN_EPRDY* notification for Endpoints which have not been through a flow-control event, the resulting behavior is undefined.

Note that control endpoints encapsulate two endpoints (an IN and OUT at the same endpoint number). If a device gives a flow control response to the host during a control transfer, then the subsequent *EndpointReady* notification must correctly indicate the direction of data flow that must be resumed. For example, a device may respond to the Status Stage of a SetAddress request (an IN transaction) with a NAK handshake. The device must indicate the IN Default Control Endpoint Address (80H) in an endpoint ready element.

Also note that the maximum number of endpoints that can be contained in a single *DN_EPRDY* notification is six (6). If a device has more than 6 flow-controlled endpoints that are now ready for data streaming, the device must use more than one *DN_EPRDY* notification.

This notification may be used in the **UnAuthenticated** device state, but ONLY for the Default Control Pipe endpoints.

- **Maximum Retransmit Rate:** a device should retransmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it observes the host has resumed transactions to any of the listed function endpoint(s) or the host has not responded to the *DN_EPRdy* for an endpoint for *TrustTimeout* seconds.

7.6.4 Device MAS Availability Changed (DN_MASAvailChanged)

When a Self-beaconing device detects a change in its MAS availability, it must transmit a *DN_MASAvailChange* notification so that the host can retrieve the updated MAS availability information by a GetStatus request (see Section 7.3.1.2). This Notification is only required for implementation by Self-Beaconing devices.

Table 7-59. Wireless USB Device MAS Availability Changed Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_MASAvailChanged* for a device MAS Availability Changed notification.

- **Maximum Retransmit Rate:** a device should transmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it receives a GetStatus(MASAvailability) or 100 ms elapses, whichever comes first.

7.6.5 Device Sleep (DN_Sleep)

Whenever a device wants to conserve power by sleeping for a period of time, the device sends a *DN_Sleep* notification to the host. See Section 4.16 in the Data Flow chapter that describes device power management. Note: devices must not use (transmit) this notification if they are currently processing a control transfer (i.e. have not yet responded with an ACK to the Status stage of the control transfer).

Table 7-60. Device Sleep Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_Sleep*.
3	*bmAttributes*	1	Bitmap	This field contains attributes for the Sleep notification: **Bit** — **Description** 0 — GTS. Value of 0 indicates that the device is going to sleep even if host response indicates there is pending work. WTS. Value of 1 indicates device will be awake if host response indicates that there is work pending. 7:1 — Reserved

- **Maximum Retransmit Rate:** a device should retransmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it observes a Work_IE with its device address included or after three attempts, whichever comes first.

7.6.6 Remote Wakeup (DN_RemoteWakeup)

Whenever a device wants to wake up a sleeping host, the device sends a *DN_RemoteWakeup* notification to the host. See Section 4.16.2.2 in the Data Flow chapter that describes remote wakeup operation.

Table 7-61. Remote Wakeup Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_RemoteWakeup*.

- **Maximum Retransmit Rate:** a device should retransmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition:** a device will cease retransmission attempts of this notification when it observes that the host has removed the Channel Stop IE from MMCs.

7.6.7 Device Alive (DN_Alive)

When a device observes a Keepalive IE and its device address matches one of the device addresses in the Target Device Address array, it will begin transmitting *DN_Alive* notifications.

Table 7-62. Wireless USB Device Alive Notification Format

Offset	Field	Size	Type	Description
0	*rWUSBHeader*	2	Record	See Table 7-52 for the default values in this field.
2	*bType*	1	Constant	The value of this field must be *DN_Alive.*

The device may choose to substitute some other type of device notification for the *DN_Alive*, depending on the current operational needs of the device.

- **Maximum Retransmit Rate:** a device should retransmit this notification at every DNTS opportunity provided by the host.
- **Stop Retransmission Condition**: If in response to a Keepalive_IE, a device will cease transmitting *DN_Alive* notifications when it no longer observes a Keepalive IE that includes its device address. If the device is transitioning from the **Asleep** power state (within a *TrustTimeou* period), the device will cease transmitting DN_Alive notifications after 3 transmissions, or if the host has started transactions to function endpoints on the device, whichever comes first.

7.7 MAC Layer-Specific Information/Framework

A host and Self-Beaconing devices must include a Distributed Reservation Protocol Information Element (DRP IE) in their Beacon to protect the Wireless USB channel. As described in the Dataflow Chapter, the host allocates a Cluster Broadcast ID and Stream Index for every Wireless USB channel it creates. The host may also establish reservations for other applications. To avoid confusion between different reservations a host must ensure that the pair {*DevAddr*, *Stream Index*} is globally unique across all of the applications reserving MAC Layer channel time from the host's platform. Table 7-63 and Table 7-64 summarize the DRP content settings for the values in a DRP IE for the Wireless USB application. The entire contents of the DRP IE are provided for completeness. Note that this specification defers to reference [3] on all discrepancies.

Table 7-63. Host Wireless USB MAC Layer DRP IE Settings

Offset	Field	Size	Value	Description
0	*Element ID*	1	Constant	Distributed Reservation Protocol IE (0x09)
1	*Length*	1	Number	This field contains the length of this descriptor, not including the Element ID and Length (4+4×N), where N is the number of DRP allocation blocks.
2	*DRP Control*	2	Bitmap	<u>**Bits**</u> / <u>**Name**</u> / <u>**Value**</u> 2:0 / Reservation Type / 011B (Private) 5:3 / Reservation Priority / Variable, Note 1 8:6 / Stream Index / Assigned 11:9 / Reason Code / 000B 12 / Status / 1B 13 / Owner / 1B 15:14 / Reserved / 000B
4	*DevAddr*	2	Number	Wireless USB Broadcast Cluster ID
6	*DRP Allocation*	Var	Record	Array of DRP Allocations. Refer to reference [3] for details.

Note 1. The value for the *Reservation Priority* field is derived according to host-specific policy.

Table 7-64. Wireless USB Cluster Member MAC Layer DRP IE Settings

Offset	Field	Size	Value	Description
0	*Element ID*	1	Constant	Distributed Reservation Protocol IE (0x09)
1	*Length*	1	Number	This field contains the length of this descriptor, not including the Element ID and Length (4+4×N), where N is the number of DRP allocation blocks.
2	*DRP Control*	2	Bitmap	**Bits** / **Name** / **Value** 2:0 / Reservation Type / 011B (Private) 5:3 / Reservation Priority / Same as host 8:6 / Stream Index / Same as host 11:9 / Reason Code / 000B 12 / Status / 1B 13 / Owner / 0B 15:14 / Reserved / 000B
4	*DevAddr*	2	Number	Host DevAddr
6	*DRP Allocation*	Var	Record	Set up by host, see Section 4.3.8.4.

This page intentionally left blank

Chapter 8 Wire Adapter

The Wire Adapter Class definition provides a generic abstraction that delivers the capabilities of a Wired or Wireless USB Host Controller through a USB device interface. This definition includes devices that are used to connect wired USB devices to a Wireless USB host (Device Wire Adapter) and also to those devices that add Wireless USB capabilities to a USB 2.0 host (Host Wire Adapter).

This chapter contains all necessary information for a designer to build a compliant Wire Adapter device. It specifies the standard and class-specific descriptors that must be present in each Wire Adapter. It further explains the use of class-specific requests that allow for full control of the Wire Adapter. This chapter is intended to be useful for:

- A hardware device vendor or firmware engineer intending to build and program a Wired or Wireless USB connected Wire Adapter which adheres to this specification, and
- A software driver developer.

This chapter first describes the common operation model and control mechanisms for Wire Adapters. This is followed by a Device Wire Adapter-specific section then a Host Wire Adapter specific section. The last section details requirements for a radio management interface, which is a required interface for a Host Wire Adapter.

8.1 Operational Model

This chapter defines two forms of Wire Adapters as illustrated in Figure 8-1. A Host Wire Adapter (HWA) is a USB device whose upstream connection is a USB 2.0 wired interface. The HWA operates as a host to a cluster of downstream Wireless USB devices. The Device Wire Adapter (DWA) is a USB device whose upstream connection is a Wireless USB interface. The DWA operates as a USB 2.0 (wired) host to devices connected below its downstream port(s). Figure 8-1 illustrates the USB device connection topology enabled when HWAs and DWAs are 'connected' in series. The HWA is the host to the DWA and WUSB Dev devices and the DWA is the host for the Mass Storage and other USB2 device.

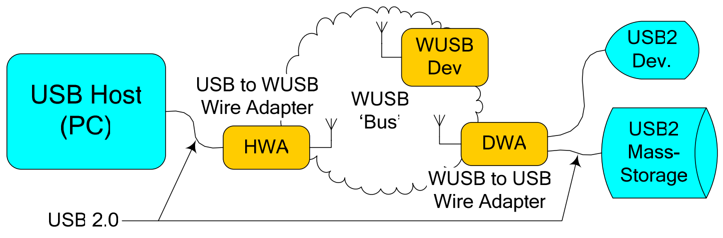

Figure 8-1. Wire Adapter Enabled USB Device Topology

8.1.1 Functional Characteristics

The basic functional block models for a HWA and DWA are illustrated in Figure 8-2. The common functional components of a Wire Adapter (WA) include the device control and WA functions. Device control is accessed via the Default Control Pipe using the USB 2.0 or Wireless USB standard device requests. These requests are defined in Chapter 9 of the USB 2.0 specification and Section 7.3 of this specification. Host Wire Adapters provide an interface for controlling the UWB radio (see Section 8.6). Device Wire Adapters provide an Isochronous Streaming interface to support isochronous data streams (see Section 8.4.1).

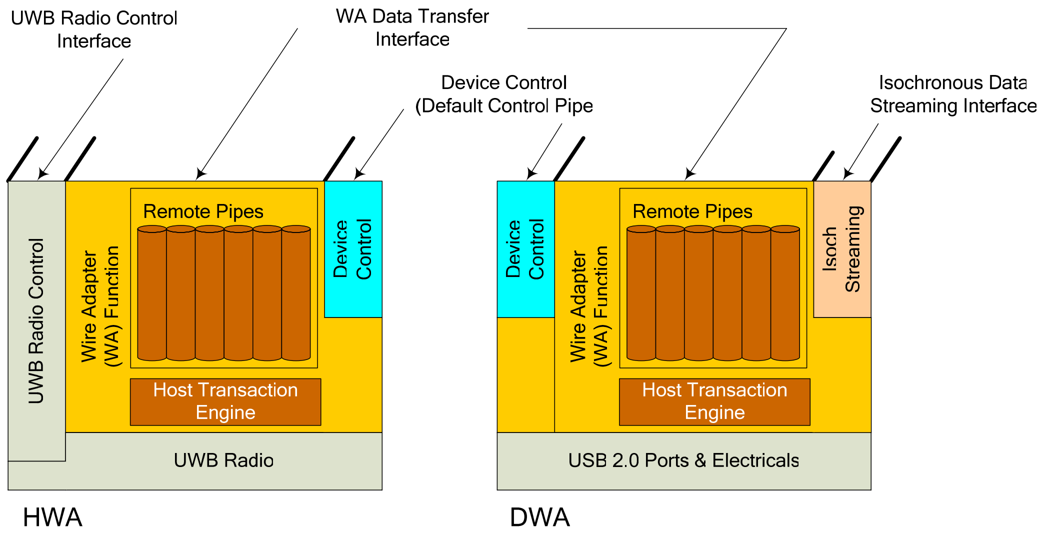
UWB Radio Control Interface
WA Data Transfer Interface
Device Control (Default Control Pipe
Isochronous Data Streaming Interface
UWB Radio Control
Wire Adapter (WA) Function
Remote Pipes
Device Control
Host Transaction Engine
UWB Radio
HWA
Device Control
Wire Adapter (WA) Function
Remote Pipes
Isoch Streaming
Host Transaction Engine
USB 2.0 Ports & Electricals
DWA

Figure 8-2. Wire Adapter Functional Model Block Diagrams

The WA function is operationally common to all WA implementations. The WA function is managed via the WA Data Transfer Interface (see Section 8.1.2). HWA and DWA-specific portions of the Data Transfer interface are described in Sections 8.5 and 8.4, respectively.

8.1.2 Data Transfer Interface

This interface has a minimum of three function endpoints. These endpoints, plus the default Control endpoint are used to accomplish all of the data and control communications between the USB host system and the Wire Adapter. The endpoints and their purposes are enumerated below:

Control Endpoint	This is the standard Default Control Pipe. It handles all non-transfer requests including all of the required standard requests which are defined in chapter 9 of the USB 2.0 specification and the WA class specific requests defined in Section8.3.
Notification Endpoint	This Interrupt IN endpoint provides transfer status notifications to the host. Transfer status is not returned on this endpoint. Instead, it provides a notification that transfer status or data is available on the Data Transfer Read Endpoint. Other asynchronous notifications are also returned on this endpoint. A Wire Adapter can send multiple notifications (up to the max packet size of the interrupt endpoint) when it is polled. See Section 8.3.2 for the types of notifications that may be sent to a host by a Wire Adapter.
Data Transfer Endpoint Pair	This set of paired (IN/OUT) endpoints is used to move data and data transfer requests through the Wire Adapter to/from the client function endpoint. [Note 1] The bulk OUT endpoint is used to send transfer requests and transfer data from the host to the Wire Adapter. The bulk IN endpoint is used to return transfer status and transfer data from the Wire Adapter to the host. See below for details.

[Note 1] A DWA does not use the Data Transfer endpoint pair in the Data Transfer Interface to perform Isochronous transfers. All DWAs that support Isochronous transfers must have an Isochronous Streaming interface (See Section 8.4.1).

8.1.3 Remote Pipe

A Remote Pipe is a logical abstraction that provides a data flow through the Wire Adapter to a specific endpoint on a specific device. Pipes are described in detail in Chapters 5 and 10 of the USB Specification 2.0.

A Wire Adapter provides a fixed number of Remote Pipes. The number of Remote Pipes supported is up to the implementer. A simple Wire Adapter must provide at least 1 Remote Pipe to provide support for a single attached device that only requires a control endpoint. The maximum number of Remote Pipes that a Wire Adapter would need to support is 3937 (127 devices X 31 endpoints/device); however Wire Adapter implementations are never expected to support 3937 physical Remote Pipes. Host software will multiplex Remote Pipes between Asynchronous endpoints. Periodic endpoints that have active transfers will require dedicated Remote Pipes. The minimum number of Remote Pipes is twice the number of devices (HWA) or ports (DWA) that the Wire Adapter supports at the same time.

The Default Control Pipe is used to initialize and manage individual Remote Pipes and the Data Transfer Pair is used to move data through them. The general operational flow of a Remote Pipe is:

1. Host initializes a Remote Pipe resource on the Wire Adapter via requests on the Default Control Pipe (see Section 8.3.1).
2. To start a control, bulk, or interrupt transfer to a device connected downstream of a Wire Adapter, the host sends a Transfer Request (Section 8.3.3) over the Data Transfer OUT endpoint to the Wire Adapter function. The Transfer Request is addressed to a specific Remote Pipe resource on the Wire Adapter. If the associated client function endpoint is an OUT, the OUT data will immediately follow the Transfer Request. In general the Data Transfer OUT endpoint is used for the following purposes:
   - Send Transfer Requests
   - Send data destined for a device connected to one of the ports of the Wire Adapter
   - Stop a Transfer Request by sending Abort Transfer Request

   The Wire Adapter must check the length and transfer request type of the Transfer Request packet received and ensure that they match. It also must check that the target Remote Pipe is configured to the same transfer type.

   If the Transfer Request received by the Wire Adapter is an OUT transaction, then the data destined for the downstream device will be sent immediately after the request in the next packet. The amount of data that follows the Transfer Request is described in the Transfer Request.

   The Wire Adapter does not STALL the endpoint when the Transfer Request is incorrect. Rather, it continues to accept the Transfer Request and any data that may follow the request. The Wire Adapter must then send a Transfer Completion notification on the notification endpoint. The host will respond by polling the associated Data Transfer IN endpoint to get the Transfer Result, which must state that the Wire Adapter detected an error in the Transfer Request. The error values are defined in Table 8-15 (see Section 8.3.3.4).
3. When the transfer completes, the Wire Adapter sends a Transfer Complete Notification (Section 8.3.3.3) to the host on the notification endpoint.
4. The data and transfer results generated from a bulk, interrupt or control transfer request are transferred to the host from the Wire Adapter through the Data Transfer IN endpoint. The data stream on this endpoint is organized as a Transfer Result (Section 8.3.3.4) followed by an optional stream of transfer data from the associated endpoint. The amount of data to be returned to the host is described in the Transfer Result.

   If a transfer does not complete successfully, the Wire Adapter will only return a Transfer Result and will not return any data back to the host.

If the host sends more Transfer Requests than a Remote Pipe in the Wire Adapter can concurrently handle (as reported in its RPipe descriptor) the Wire Adapter will NAK the transaction until it has completed a pending transfer on that Remote Pipe.

Section 8.5.1 describes how isochronous transfers are handled on a HWA and Section 8.4.1 describes how they are handled on a DWA.

8.1.4 Wire Adapter Functional Blocks

The Wire Adapter consists of five functional blocks as illustrated in Figure 8-3.

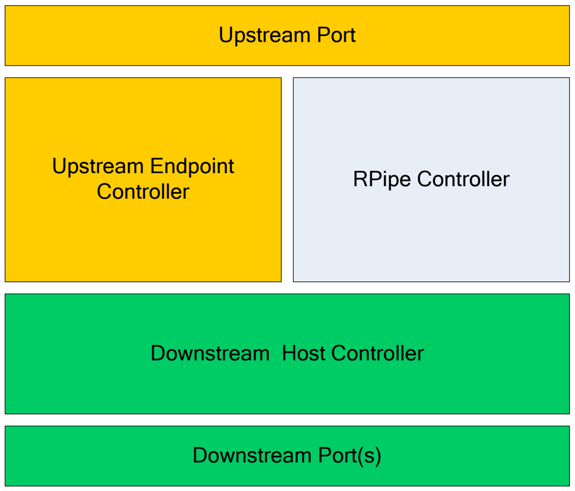

Figure 8-3. General WA Function Blocks

8.1.5 Downstream Port(s)

A Device Wire Adapter has one or more downstream ports. The ports behave like those in the USB 2.0 hubs. The Device Wire Adapter monitors the status of all of the ports and reports them to the host if there is any change.

The number of downstream ports that is implemented on a Device Wire Adapter is indicated in the bNumPorts field of the Wire Adapter Class Descriptor. The maximum number of downstream ports that can be implemented on a Wire Adapter is 127. The functions and behavior of the downstream port on a DWA are the same as the ones of a USB 2.0 Hub which are described in the Section 11.5 of the USB 2.0 Specification.

A Host Wire Adapter does not have downstream ports since all the devices it can communicate with are Wireless. The HWA will forward all asynchronous notifications (connect/reconnect/disconnect/sleep etc) received from a downstream device to Host software. Host software is responsible for handling the various downstream port functions for an HWA. However, an HWA needs to store some information for each device connected downstream of it, See Section 8.5.3.6. An HWA specifies the total number of devices that can be connected to it in the bNumPorts field in its Wire Adapter descriptor.

8.1.6 Upstream Port

A host communicates with a Wire Adapter via its upstream port. The port is used for:

- Control of the Wire Adapter function
- Notification of changes to the host
- Communicating with the devices connected downstream of the Wire Adapter (via Remote Pipes)

A HWA may be a Bus Powered USB 2.0 device. The upstream port of the HWA must operate at Full-speed and High-speed. If the HWA is operating at full speed then Isochronous transfers are not supported. The upstream port of an HWA is the device side interface of a USB 2.0 device which is described in Section 11.6 of the USB 2.0 Specification.

The upstream port of a DWA is the device side interface of a Wireless USB device which is described in Chapter 7 of this specification.

8.1.7 Downstream Host Controller

A Wire Adapter has a host controller on which it creates, schedules, and manages the transaction protocol to devices connected downstream. The downstream host controller receives the transfer information from Transfer Requests and the associated RPipe Descriptor to schedule the newly added transfer to the system. At the same time, it manages the schedule following the protocol of the downstream bus. The downstream host controller also controls the downstream ports on a DWA for data transfer.

A Host Wire Adapter must be a Wireless USB Host Controller to devices 'connected' downstream.

A Device Wire Adapter must be a USB 2.0 Host Controller which is described in Chapter 10 of the USB 2.0 Specification.

8.1.8 Upstream Endpoint Controller

All Wire Adapters must have at least the following four endpoints:

- Default Control Endpoint
- Notification Endpoint
- Data Transfer Pair (Write (Bulk OUT) Endpoint and a matching Read (Bulk IN) Endpoint).

Wire Adapters may have additional endpoints depending on what additional features they support. For example a DWA that supports Isochronous streaming will have one or more isochronous endpoints. In the case of an HWA, it has a Radio Control Interrupt endpoint as part of the Radio Control Interface.

8.1.9 Remote Pipe Controller

This section describes how the Remote Pipes are used.

8.1.9.1 RPipe Descriptor

The RPipe descriptor holds all the information necessary to perform data transfers between a Wire Adapter and an endpoint on a device connected downstream of it. It has to be configured before performing any transaction with a downstream endpoint. Host software uses the SetRPipeDescriptor request to configure a Remote Pipe. The descriptor may be overwritten to retarget the Remote Pipe at a different endpoint using another SetRPipeDescriptor request. Host software is responsible to save the current state of the Remote Pipe before retargeting a Remote Pipe to a different endpoint. Host software can get the current state of a Remote Pipe by sending a GetRPipeDescriptor request to the Wire Adapter.

Host software can send a SetRPipeDescriptor request to a Remote Pipe only when that Remote Pipe is in the Idle or UnConfigured state. Host software is required to correctly multiplex the available Remote Pipes over the downstream endpoints that need to be serviced.

8.1.9.2 Bulk OUT Overview

For a Host to Device (OUT) data stream, the basic model is that the Host sends data to the Wire Adapter in the context of a Remote Pipe and the Wire Adapter moves the data to the Wired or Wireless USB Endpoint, utilizing the information present in the previously configured Remote Pipe.

The Host can determine from the RPipe Descriptor exactly how much buffering the Wire Adapter has allocated or can allocate to this Remote Pipe. Figure 8-4 illustrates the generic data flow model for an OUT-bound Bulk data stream. It shows the transfer request/data stream and the feedback/transfer status stream.

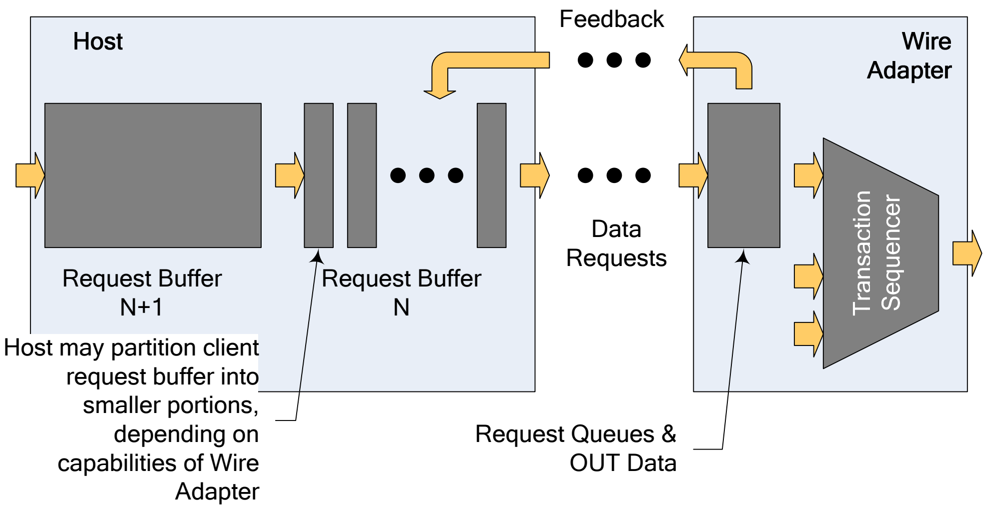
Host
Feedback
Wire Adapter
Request Buffer N+1
Request Buffer N
Data Requests
Transaction Sequencer
Host may partition client request buffer into smaller portions, depending on capabilities of Wire Adapter
Request Queues & OUT Data

Figure 8-4. Wire Adapter Bulk OUT Operational Data Flow Model

In order to move the client buffer into the Wire Adapter, host software may have to divide the client buffer into smaller chunks and forward them to the Wire Adapter. Host software is required to send data to the Wire Adapter only in multiples of the Remote Pipe's Maximum Packet Size field. The lone exception to this rule is when the buffer remaining is not an even multiple of the Remote Pipe's Maximum Packet Size field. In this case the last data payload from the host to the Wire Adapter is the residual of the client data buffer. Host software must not concatenate client data buffers in order to fill a data payload of the Remote Pipe's Maximum Packet Size field.

The size of each chunk depends on how much buffering has been allocated to the Remote Pipe. Each OUT pipe transfer request is followed by the data for the Remote Pipe. The Wire Adapter is required to move the data portions sent by the host software to the Wired or Wireless USB Endpoint in the same order as the host sent them.

It is important that the visibility of the original client buffer boundaries be preserved into the Wire Adapter. Per-transfer attributes are used to inform the Wire Adapter how to manage the buffer portions. For example, the attributes include information about whether this is a first, middle or end buffer portion. In addition, the host software may be allowed to queue buffer portions of more than one buffer to the Wire Adapter (at the same time). Therefore all transfer requests associated with the same client buffer must have a unique identifier (i.e. tag). Host software is responsible for generating unique transfer request identifiers. The Wire Adapter will send these identifiers back to the host software in a transfer result when it completes a transfer request.

8.1.9.3 Bulk IN Overview

For a Device to Host (IN) data stream, once host software has client buffer space available, it sends a transfer request to the Wire adapter to begin requesting data from a downstream connected Wired or Wireless USB endpoint. It ensures that it does not ask for more data from the Wired or Wireless USB endpoint than the Wire Adapter has buffering. Figure 8-5 illustrates the general data flow model of a Bulk IN data stream.

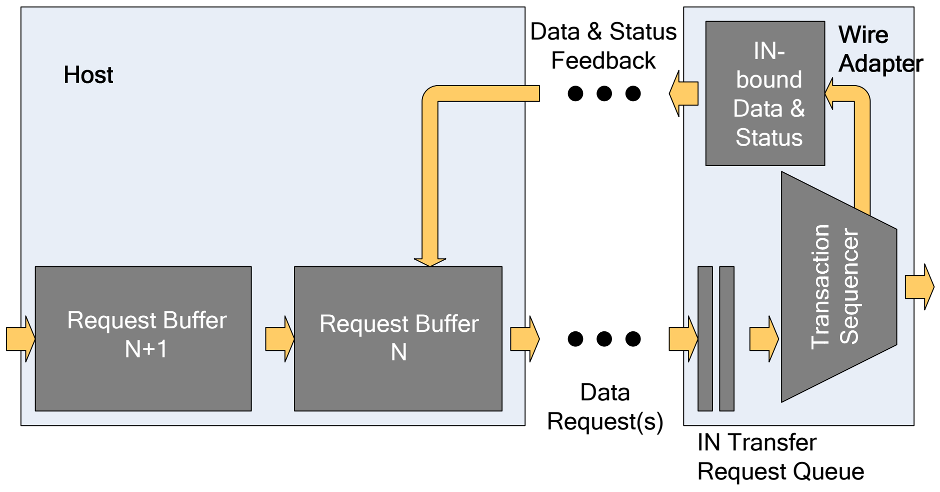

Figure 8-5. Wire Adapter IN Operational Data Flow Model

Host software may queue multiple transfer requests to the Wire Adapter. Each transfer request maps to a single client input buffer. The size of the transfer request is allowed to be up to $2^{\wedge 32}$ - 1. If the client buffer is larger than the Wire Adapter has buffering for, host software will split the buffer into multiple segments that the Wire Adapter can accommodate and then manage the appropriate short packet semantics when short packets occur in the data stream. Host software will tag each IN transfer request with a unique identifier. The maximum number of IN requests per Remote Pipe the Wire Adapter may accommodate is determined by an attribute provided in the RPipe descriptor.

The feedback data stream is multiplexed data and transfer status information. The Wire Adapter may implement a shared IN data buffer across all IN Remote Pipes. It may optionally implement individual buffering for each IN Remote Pipe. The granularity and frequency of data/transfer status communications to the USB Host on the feedback stream is implementation dependent. However, feedback communications must occur frequently enough to deliver data to the host without causing frequent data streaming stalls.

Whenever the Wire Adapter observes that the associated Wired or Wireless USB IN Endpoint provides a short packet, the Wire Adapter will send the residual queued data to the host with a transfer status indicating the transfer request is completed. It will then begin servicing the next transfer request queued for the Remote Pipe at the next appropriate opportunity.

As feedback communications arrive at the Host, host software must parse the multiplexed data/status stream, copying data into the client buffer and noting or responding to status feedback as appropriate.

8.1.9.4 Control Transfer Overview

A USB control transfer has 2 (Setup and Status only) or 3 (Setup, Data and Status) stages depending on the request. If the size of the data stage of the USB control transfer is less than or equal to the buffer available on the Remote Pipe then the USB control transfer can be completely described in one transfer request and the Wire Adapter is responsible for completing all stages of the USB control transfer.

If the USB control transfer has a data stage larger than the buffer available on the Remote Pipe, host software will split the transfer into multiple segments. The first transfer request segment will have a valid set of bytes in the Setup data and describe the amount of data that needs to be sent or received from the device. The subsequent transfer request segments will not have any valid bytes in the Setup data field. A Wire Adapter must only decode and send the Setup data included in the first transfer request segment of a multi-segment transfer request. All segments of the transfer request must describe a buffer that is an exact multiple of the Remote Pipe's Maximum Packet Size field except for the final segment of the transfer request. This is required so that the Wire Adapter can perform a status stage transaction either when a short packet occurs in one of the transfer request segments or when the last transfer request segment has completed. Each transfer request segment is tagged with a unique identifier, in order to allow host software to match returned status and possibly IN data with the client request.

8.1.9.5 Interrupt Transfer Overview

Interrupt IN-bound and OUT-bound Remote Pipes have interface and data transfer semantics essentially identical to the Bulk IN/OUT model described in Sections 8.1.9.2 and 8.1.9.3. The only difference is that each Remote Pipe is typed as an Interrupt and includes an additional attribute that indicates the period at which the endpoint should be provided service. The Wire Adapter has full freedom to determine the actual servicing of the endpoint, as long as it is at least as frequent as the period requested by the bInterval field in the Remote Pipe descriptor.

8.1.9.6 Isochronous Transfer Overview

An overview of Isochronous streaming support is provided in Section 8.5.1 for an HWA and Section 8.4.2 for a DWA.

8.1.10 Suspend and Resume

Wire Adapters are bridges between Wired and Wireless USB buses. Wire Adapters must support suspend and resume both as a device and in terms of propagating suspend/resume events between the busses it bridges.

8.1.10.1 DWA Suspend and Resume

A Device Wire Adapter uses the mechanisms defined in Section 4.16 to manage its power consumption. Depending on the state of the Host (Wireless Host State) it is connected to and the state the DWA is in (DWA Upstream State) the DWA might decide to go to sleep. Table 8-1 provides the requirements of a DWA when it wants to go to sleep and what it must do to stay in the Sleep state for each combination of these states.

A DWA is a bridge device in a USB hierarchy (i.e. it has a Wireless USB bus upstream and a Wired USB 2.0 bus downstream). The suspend resume management model for a DWA is derived directly from the USB 2.0 model defined in Section 7.1.7.7 of reference [1]. A summary of this model is: a DWA must always attempt to propagate resume signaling, regardless of whether it has been enabled for remote wake itself. It will turn other events into remote wake signaling if and only if it has been enabled for remote wake. A DWA must always serve as the Controlling Hub in response to resume signaling from a downstream device. Table 8-1 summarizes the DWA operational requirements to meet this model. The Event column lists the wake events and the Effect column indicates the action a DWA must take when it detects the wake event for each combination of Wireless Host State and DWA Upstream State. The Requirement column describes DWA behavior while no downstream wake events occur.

Table 8-1. DWA Suspend/Resume Requirements

Wireless Host State	DWA Upstream State	Requirement	Event	Effect
Awake	Awake	Normal Operation		
	Sleep/ Sleep+[Note 1]	Wake up at least once every *TrustTimeout* period Send sleep notification (Want to Sleep) to host	Any event	If the event originated from a suspended downstream port then resume downstream port. Send Reconnect or DN_Alive as per Section 4.16.1.2 Send event notification to host
Host Sleep [Note 2]	Sleep	Wake up at least once every *TrustTimeout* period Check for host awake	Any event	Ignore
Host Sleep+ [Note 3]	Sleep	Wake up at least once every *TrustTimeout* period Check for host awake	Resume Signaling	Resume downstream port Send Remote Wake Notification Send Reconnect request Send wake event notification to host
	Sleep+	Wake up at least once every *TrustTimeout* period Check for host awake	Any wake event	

[Note 1] Device is sleeping with remote wake enabled

[Note 2] Host Sleep is where host has stopped the Wireless USB channel

[Note 3] Host Sleep+ has stopped the Wireless USB channel, but will resume the Wireless USB channel on a periodic basis in order to provide opportunities for remote-wake enabled devices to signal remote wake.

8.1.10.2 HWA Suspend and Resume

A Host Wire Adapter's upstream power state, like any USB 2.0 device, is managed by the Host. An HWA may be active or suspended (HWA Upstream State) depending on whether it observes SOFs on its upstream port. Further an HWA may be directed to start or stop the Wireless USB Channel (HWA Downstream State) and either enabled or disabled for remote wakeup by the driver for the HWA. Table 8-2 provides the HWA requirements as a Wireless USB Channel Host for legal combinations of HWA Upstream and Downstream State.

An HWA is a bridge device in a USB hierarchy (i.e. it has a Wired USB 2.0 bus upstream and a Wireless USB 'bus' downstream). The suspend resume management model for an HWA is derived directly from the USB 2.0 model defined in Section 7.1.7.7 of reference [1]. A summary of this model is: an HWA must always attempt to propagate resume signaling, regardless of whether it has been enabled for remote wake itself. It will turn other events into remote wake signaling if and only if it has been enabled for remote wake. An HWA must always serve as the Controlling Hub in response to resume signaling from a downstream device. Table 8-2 summarizes the HWA operational requirements to meet this model. The Event column lists the wake events and the Effect column indicates the action an HWA must take when it detects the wake event for legal combinations of HWA Upstream and Downstream State. The Requirement column describes HWA behavior while no downstream wake events occur.

Table 8-2. HWA Suspend Resume Requirements

HWA Upstream State	HWA Downstream State	Requirement	Event	Effect
Active	Awake	Normal Operation		
	Host Sleep	Before sleeping, send at least 3 MMCs with Channel Stop IE with remote wakeup bit set to 0B	Any event	Ignore
	Host Sleep+	Wake up at least once every *TrustTimeout* period Send at least 3 MMCs with Channel Stop IE with remote wakeup bit set to 1B	Remote Wake Notification or DN_Connect or DN_Disconnect	Start Channel Send DN Receive Notification
Suspended	Host Sleep	Before sleeping, send at least 3 MMCs with Channel Stop IE with remote wakeup bit set to 0B	Any event	Ignore
	Host Sleep+	Wake up at least once every *TrustTimeout* period Send at least 3 MMCs with Channel Stop IE with remote wakeup bit set to 1B	Remote Wake Notification or DN_Connect or DN_Disconnect	Resume Signaling on upstream port Start Channel Send DN Receive Notification

8.1.11 Reset Behavior

A Host Wire Adapter can be reset through the upstream USB bus with standard signaling. After reset, HWA clears all the status, state machines and registers and sets default values in the descriptors.

A Device Wire Adapter can be reset either by sending it a SetAddress (0) command or by sending it a Wireless USB Reset Device IE. After reset, DWA clears all the status, state machines and registers and sets default values in the descriptors. Though this is implementation dependent, it may wait for user interaction before trying to find an available host.

To reset just the host controller in a Wire Adapter, the host issues the class specific Set Feature(WIRE_ADAPTER_RESET) request.

8.1.12 Device Control

Host software can enable, disable and/or reset the host controller in a Wire Adapter device using the Set/Clear Wire Adapter Feature requests. See Section 8.3.1.3 and Section 8.3.1.9 for details on these commands.

In addition, host software can query the Wire Adapter host controller status using the Get Wire Adapter Status command described in Section 8.3.1.6.

8.1.13 Buffer Configuration

A Wire Adapter must have buffers to store the data received on its upstream port for OUT transfers and for the data received from downstream devices for IN transfers. This buffer consists of one or more buffer blocks. The size of each buffer block is described in the Wire Adapter Class Descriptor and the number of the blocks for each RPipe is described in the RPipe's Descriptor.

The size of each block is implementation dependent. The number of buffer blocks per RPipe may be fixed by the Wire Adapter implementation; in this case the number is a read only field and cannot be changed in the RPipe Descriptor. This is suitable for a Wire Adapter implementation for permanently attached devices or designed exclusively to be used with a particular class of devices. Such Wire Adapters must not have any ports that are user accessible.

If the number of blocks per RPipe is dynamically manageable by host software then a value of "zero" must be reported in the *wBlocks* field of an RPipe Descriptor after reset. In this case, Host software is responsible to correctly assign the amount of buffer per RPipe. The total number of available buffer blocks is determined by the *wRPipeMaxBlock* field in the Wire Adapter Descriptor. This implementation choice is suitable for an all-purpose Wire Adapter.

8.2 Descriptors

Wire Adapter descriptors are derived from the general USB device framework. Wire Adapter descriptors define a Wire Adapter device. The host accesses these descriptors through the Wire Adapter's control endpoint. The Wire Adapter class pre-defines certain fields in standard USB descriptors. Other fields are either implementation-dependent or not applicable to this class.

The Wire Adapter class defines additional device class descriptors. Vendor-specific descriptors may be defined. Host Wire Adapter descriptors are described in Section 8.5.2. Device Wire Adapter descriptors are described in Section 8.4.3.

8.3 Requests

All Wire Adapter devices must implement all required standard commands in the core device framework. All Wire Adapters must support the Class specific requests in Section 8.3.1.

Host Wire Adapter specific requests are specified in Section 8.5.3.

Device Wire Adapters must support all required Wireless USB extensions to the Chapter 9 framework as specified in the Wireless USB framework chapter. A DWA must support the class specific requests defined in Section 8.4.4.

The valid values for the *bmRequestType.Recipient* field are extended in this class specification to allow addressing of Ports and RPipes as illustrated in Table 8-3.

Table 8-3. Recipient Encoding Extension

Value	Recipient
0	Device
1	Interface
2	Endpoint
3	Other
4	Port
5	RPipe
6-31	Reserved

8.3.1 Wire Adapter Class-Specific Requests

Table 8-4. Wire Adapter Class-Specific Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
Abort RPipe	00100101B	ABORT_RPIPE	Zero	RPipe Index	Zero	None
Clear RPipe Feature	00100101B	CLEAR_FEATURE	Feature Selector	RPipe Index	Zero	None
Clear Wire Adapter Feature	00100001B	CLEAR_FEATURE	Feature Selector	Interface Number	Zero	None
Get RPipe Descriptor	10100101B	GET_DESCRIPTOR	Descriptor Type	RPipe Index	Descriptor Length	RPipe Descriptor
Get RPipe Status	10100101B	GET_STATUS	Zero	RPipe Index	1	RPipe Status
Get Wire Adapter Status	10100001B	GET_STATUS	Zero	Interface Number	4	Wire Adapter Status
Set RPipe Descriptor	00100101B	SET_DESCRIPTOR	Descriptor Type	RPipe Index	Descriptor Length	RPipe Descriptor
Set RPipe Feature	00100101B	SET_FEATURE	Feature Selector	RPipe Index	Zero	None
Set Wire Adapter Feature	00100001B	SET_FEATURE	Feature Selector	Interface Number	Zero	None
Reset RPipe	00100101B	RESET_RPIPE	Zero	RPipe Index	Zero	None

Table 8-5. Wire Adapter Class Request Codes

bRequest	Value
GET_STATUS	0
CLEAR_FEATURE	1
Reserved	2
SET_FEATURE	3
Reserved	4-5
GET_DESCRIPTOR	6
SET_DESCRIPTOR	7
Reserved	8-13
ABORT_RPIPE	14
RESET_RPIPE	15

Table 8-6. Wire Adapter Class Feature Selector

Feature Selector	Recipient	Value
WIRE_ADAPTER _ENABLE	Wire Adapter Device	1
WIRE_ADAPTER _RESET	Wire Adapter Device	2
RPIPE_PAUSE	RPipe	1

8.3.1.1 Abort RPipe

This request aborts all transfers pending on the given RPipe.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100101B	ABORT_RPIPE	Zero	RPipe Index	Zero	None

Upon receipt of this request, the Wire Adapter will terminate all pending transfers for the given RPipe and place the RPipe in the Idle state. The Wire Adapter must return a transfer completion notification, transfer result and any data that was received and acknowledged from the targeted endpoint for all terminated transfers.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies an RPipe that does not exist.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.2 Clear RPipe Feature

This request resets a value in the reported RPipe status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100101B	CLEAR_FEATURE	Feature Selector	RPipe Index	Zero	None

The *wIndex* field contains an RPipe Index. The RPipe index must be a valid RPipe index for that Wire Adapter.

Clearing a feature disables that feature; see Table 8-6 for the feature selector definitions that apply to an RPipe as a recipient. Features that can be cleared with this request are:

- RPIPE_PAUSE

It is a Request Error if *wValue* is not a feature selector listed in Table 8-6, if *wIndex* specifies an RPipe that does not exist, or if *wLength* is not as specified above.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.3 Clear Wire Adapter Feature

This request is used to clear or disable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	CLEAR_FEATURE	Feature Selector	Interface Number	Zero	None

The lower byte of *wIndex* contains the target interface number. Clearing a feature disables that feature; see Table 8-6 for the feature selector definitions that apply to the controller as a recipient. Features that can be cleared with this request are:

- WIRE_ADAPTER_ENABLE

It is a Request Error if *wValue* is not a feature selector listed in Table 8-6 or *wLength* is not as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.4 Get RPipe Descriptor

This request returns the current Wire Adapter RPipe Descriptor.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100101B	GET_DESCRIPTOR	Descriptor Type	RPipe Index	Descriptor Length	RPipe Descriptor

The GetDescriptor() request for the RPipe descriptor follows the same usage model as that of the standard GetDescriptor() request. The lower byte of *wValue* must be set to zero and the RPipe Index is given in the lower twelve bits of the *wIndex* field.

If *wLength* is larger than the actual length of the descriptor, then only the actual length is returned. If *wLength* is less than the actual length of the descriptor, then only the first *wLength* bytes of the descriptor are returned; this is not considered an error even if *wLength* is zero.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.5 Get RPipe Status

This request returns the current status for the given RPipe.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100101B	GET_STATUS	Zero	RPipe Index	1	RPipe Status

The *wIndex* field contains an RPipe Index. The RPipe index must be a valid RPipe index for that Wire Adapter. The returned value describes the current status of the specified RPipe. The meanings of the individual bits are given in Table 8-7.

Table 8-7. RPipe State Report

Offset	Field	Size	Value	Description
0	*RPipeState*	1	Bitmap	State of this RPipe: __Bit__ __Description__ 0 1 = Idle, 0 = Active 1 1 = Paused, 0 = Not Paused 2 1 = Configured 0 = UnConfigured 7:3 Reserved

The RPipe state diagram is given below.

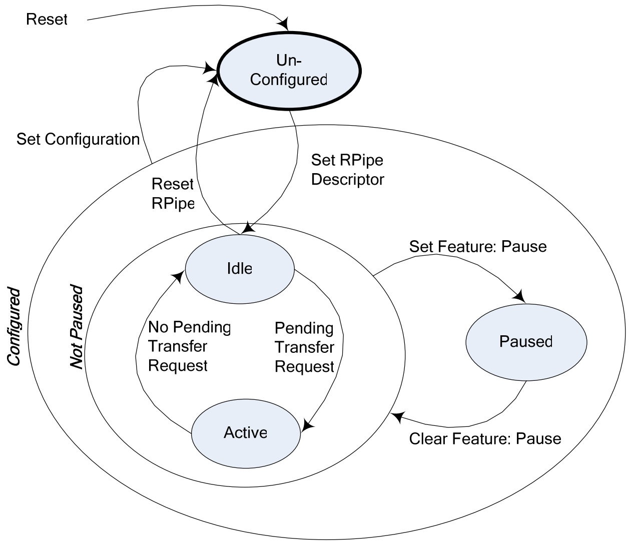

Figure 8-6. RPipe State Diagram

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies an RPipe that does not exist.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.6 Get Wire Adapter Status

This request returns the current status of the Wire Adapter.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_STATUS	Zero	Interface Number	4	Wire Adapter Status

The lower byte of *wIndex* contains the target interface number. The returned value gives the current Wire Adapter status. The meanings of the individual bits are given in Table 8-8.

Table 8-8. Wire Adapter Status Bits

Bit	Description
0	**Controller Enabled/Disabled:** This field indicates whether the controller is enabled or disabled. **Value** **Description** 0 Controller is disabled 1 Controller is enabled
1	**Reset:** This bit is set while a Reset is in progress. It is cleared by the Wire Adapter once Reset is completed
31:2	**Reserved:** These bits return 0 when read.

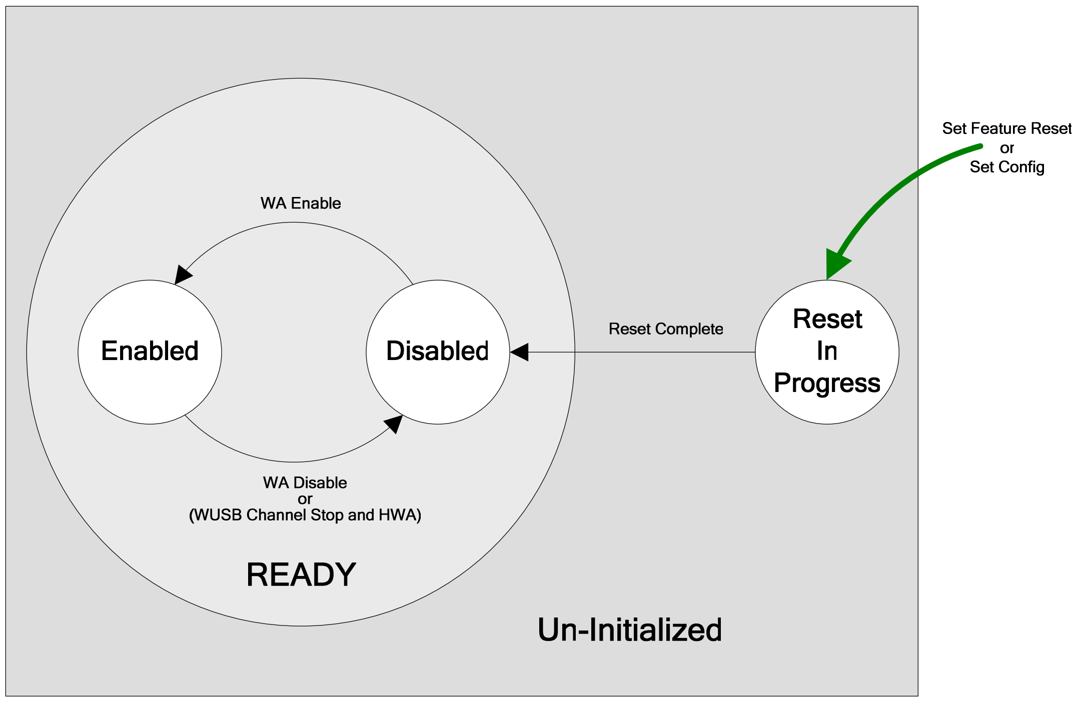

Figure 8-7. Wire Adapter Host Controller State Diagram

Table 8-9. Wire Adapter Enabled Behavior

WA Type	Behavior
DWA	Parses Schedule Sends SOFs
HWA	Parses Schedule Sends MMCs

It is a Request Error if *wValue* or *wLength* are other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.7 Set RPipe Descriptor

This request sets the related attributes of specified RPipe.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100101B	SET_DESCRIPTOR	Descriptor Type	RPipe Index	Descriptor Length	RPipe Descriptor

The host supplies the new RPipe settings in the RPipe descriptor it sends in the data phase. The lower byte of *wValue* must be set to zero and the RPipe Index is given in the lower twelve bits of the *wIndex* field.

It is a Request Error if the RPipe is not in the Idle or UnConfigured state when this command is received.

It is a Request Error if *wLength* is not equal to the RPipe Descriptor length.

It is a Request Error if *wValue* or *wIndex* are other than as specified above.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.8 Set RPipe Feature

This request sets the specified RPipe to the specified RPipe state.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100101B	SET_FEATURE	Feature Selector	RPipe Index	Zero	None

The *wIndex* field contains an RPipe Index. The RPipe index must be a valid RPipe index for that Wire Adapter.

Setting a feature enables that feature; see Table 8-6 for the feature selector definitions that apply to an RPipe as a recipient. Features that can be set with this request are:

- RPIPE_PAUSE

It is a Request Error if *wValue* is not a feature selector listed in Table 8-6, if *wIndex* specifies an RPipe that does not exist, or if *wLength* is not as specified above.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.9 Set Wire Adapter Feature

This request is used to set or enable a specific feature.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_FEATURE	Feature Selector	Interface Number	Zero	None

The lower byte of *wIndex* contains the target interface number. Setting a feature enables that feature or starts a process associated with that feature; see Table 8-6 for the feature selector definitions that apply to the Wire Adapter as a recipient. Features that can be set with this request are:

- WIRE_ADAPTER_ENABLE
- WIRE_ADAPTER_RESET

It is a Request Error if *wValue* is not a feature selector listed in Table 8-6 or *wLength* is not as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.1.10 Reset RPipe

This request resets the specified RPipe to a known state.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100101B	RESET_RPIPE	Zero	RPipe Index	Zero	None

This request resets an RPipe in the Idle state. After reset, the RPipe will transition to the UnConfigured state and transfer sequencing mechanism for the RPipe will be reset to its start state.

The host must either wait for pending transfers to drain or abort the pending transfers on this RPipe with the ABORT_RPIPE request (see Section 8.3.1.1) before sending this request.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies an RPipe that does not exist.

If the RPipe is not in an Idle state, the Wire Adapter's response to this request is undefined.

If the Wire Adapter is not configured, the Wire Adapter's response to this request is undefined.

8.3.2 Notification Information

Asynchronous notification messages are sent back on the notification endpoint. All Wire Adapters return Transfer Completion notifications on the notification endpoint. The format and use of a Transfer Completion notification is given in Section 8.3.3.3.

Host Wire Adapter specific notifications are detailed in Section 8.5.4.

Device Wire Adapter specific notifications are detailed in Section 8.4.5.

8.3.3 Transfer Requests

To initiate a transfer, the host must first configure an RPipe to the target endpoint on the target device. The index of the configured RPipe is then used in the transfer requests.

After configuring the RPipe, the host submits a transfer request and an arbitrary amount of data to the Wire Adapter via the Data Transfer Write endpoint. The amount of data accompanying the transfer request is controlled by the total amount of data in the transfer and the amount of buffering available to the RPipe. The amount of buffer available on the RPipe is indicated in the RPipe descriptor. RPipes must support at least two concurrent requests per Interrupt RPipe in order to support Interrupt transfers. In addition an HWA must support at least four concurrent requests per Isochronous RPipe to support Isochronous transfers.

8.3.3.1 Control Transfers

Control Transfers are performed using a Control Transfer Request as shown in Table 8-10. The format of this request includes the setup data for the control transfer to be performed to the downstream connected device. Table 8-11 describes the operational requirements of the Wire Adapter when it receives a Control Transfer Request segment.

It is the responsibility of the host to insure that the amount of data to be transferred for all Control Transfer Request segments except for the last Control Transfer Request segment is a multiple of the *wMaxPacketSize* field in the RPipe descriptor so as to maintain USB short packet semantics. The Wire Adapter must send back intermediate transfer completion notifications (See Section 8.3.3.3) and transfer results (See Section 8.3.3.4) when it completes each Control Transfer Request segment. It is the responsibility of the Wire Adapter to perform a status stage transaction when the last Control Transfer Request segment of the data stage is completed or if it receives a short packet from the device in any Control Transfer Request segment of the data stage.

The data for each segment of a non zero length Control Write Transfer is sent immediately after each Control Transfer Request segment in the next packet.

Table 8-10. Control Transfer Request

Offset	Field	Size	Value	Description
0	*bLength*	1	18H	Length of this request
1	*bRequestType*	1	80H	REQUEST_TYPE_CONTROL – indicates a control transfer
2	*wRPipe*	2	Number	RPipe this transfer is targeted to
4	*dwTransferID*	4	Number	Host-assigned ID for this transfer. All pending *dwTransferID* are unique.
8	*dwTransferLength*	4	Number	Amount of data following for an OUT transfer or the maximum amount of returned data for an IN transfer

Table 8-10. Control Transfer Request (cont.)

Offset	Field	Size	Value	Description
12	*bTransferSegment*	1	Bitmap	**Bit** / **Description** 6:0 Segment Number 7 Last Segment
13	*bmAttribute*	1	Bitmap	**Bit** / **Description** 0 Control Transfer direction **Value** / **Meaning** 0 Control transfer write 1 Control transfer read 7:1 Reserved, must be zero
14	*wReserved*	2	Zero	Reserved, must be zero.
16	*baSetupData*	8	Byte array	8-byte setup packet data

Table 8-11. WA Control Transfer Request Operational Requirements

bRequestType	Segment Number	Last Segment	Operational Requirement
REQUEST_TYPE_CONTROL	0	1	Send contents of *baSetupData* The transfer request describes the complete USB control transfer The WA must perform a status stage transaction after the data stage if any
REQUEST_TYPE_CONTROL	0	0	Send contents of *baSetupData* The transfer request describes a USB control transfer with the first segment of the data transfer stage The WA must perform a status stage transaction if it receives a short packet from the device during this transfer segment
REQUEST_TYPE_CONTROL	>0	0	The transfer request describes a subsequent segment of the data transfer stage of a USB control transfer The WA must perform a status stage transaction if it receives a short packet from the device during this transfer segment
REQUEST_TYPE_CONTROL	>0	1	The transfer request describes the last segment of the data transfer stage of a USB control transfer The WA must perform a status stage transaction if it receives a short packet from the device during this transfer segment or at the end of this transfer segment

8.3.3.2 Bulk and Interrupt Transfers

Bulk and Interrupt transfers use the Bulk or Interrupt Transfer Request as shown in Table 8-12. This request type allows large transfers to be segmented into multiple smaller transfers to avoid RPipe buffer overflow on the Wire Adapter. When transfers are segmented, the host must insure that the amount of data for all segments except for the last segment is a multiple of the *wMaxPacketSize* field in the RPipe descriptor. This is necessary to maintain USB short-packet semantics.

For OUT transfers, the request and the data are sent as consecutive transactions. This allows the Wire Adapter to receive and interpret the request first and prepare to receive the data.

Table 8-12. Bulk or Interrupt Transfer Request

Offset	Field	Size	Value	Description
0	*bLength*	1	10H	Length of this request
1	*bRequestType*	1	81H	REQUEST_TYPE_BULK_OR_INTERRUPT – indicates a bulk/interrupt transfer
2	*wRPipe*	2	Number	RPipe this transfer is targeted to
4	*dwTransferID*	4	Number	Host-assigned ID for this transfer
8	*dwTransferLength*	4	Number	Amount of data following for an OUT transfer or the maximum amount of returned data for an IN transfer
12	*bTransferSegment*	1	Bitmap	**Bit** **Description** 6:0 Segment Number 7 Last segment
13	*bReserved*	1	Zero	Reserved for future use, must be zero.
14	*wReserved*	2	Zero	Reserved for future use, must be zero.

8.3.3.3 Transfer Completion Notification

When the Wire Adapter completes a transfer or a portion of a segmented transfer, it will return a Transfer completion notification on the Notification endpoint. The Transfer Notification format is shown in Table 8-13. The data transfer endpoint on which the transfer result and data if any is available is given in the *bEndpoint* field. The type of the notification is indicated in the *bNotifyType* field.

Table 8-13. Transfer Notification

Offset	Field	Size	Value	Description
0	*bLength*	1	4	Length of this block of data
1	*bNotifyType*	1	93H	NOTIFY_TYPE_TRANSFER
2	*bEndpoint*	1	Number	Endpoint on which the transfer result is available
3	*bReserved*	1	Number	Reserved

8.3.3.4 Transfer Result

Host software can get the Transfer Result from the Data Transfer Read endpoint number indicated in the previous transfer completion notification. If the corresponding transfer was an IN transfer (Bulk/Interrupt IN or Control Transfer Read), the transfer result and the IN data will be returned as separate and consecutive transfers. This allows the host to receive the result packet, interpret the results and identify the correct host buffer in which to receive the IN data. The Transfer Result format is illustrated in Table 8-14.

Table 8-14. Transfer Result

Offset	Field	Size	Value	Description
0	*bLength*	1	10H	Length of this block of data (not counting transfer data)
1	*bResultType*	1	83H	RESULT_TYPE_TRANSFER – indicates result type
2	*dwTransferID*	4	Number	Host-assigned ID for this transfer
6	*dwTransferLength*	4	Number	Amount of data transferred for either OUT or IN

Table 8-14. Transfer Result (cont.)

Offset	Field	Size	Value	Description
10	*bTransferSegment*	1	Bitmap	<u>**Bit**</u> <u>**Description**</u> 6:0 Segment Number 7 Last segment
11	*bTransferStatus*	1	Number	The transfer status
12	*dwNumOfPackets*	4	Number	Number of Packet lengths and status following. This will be zero for non Isochronous transfers. This field is reserved and must be set to zero for DWAs.

The Transfer (or Packet) Status field in the Transfer Result (or Packet Status) returned to host software is used to decode whether a transfer completed successfully or the type of error that occurred while performing the transfer described by a previously received transfer request. The set of legal Transfer/Packet Status values is defined in Table 8-15.

Table 8-15. Transfer/Packet Status

Bit	Description
5:0	<u>**Status Value**</u> <u>**Description**</u> 0 TRANSFER_STATUS_SUCCESS The transfer completed successfully. Bit 6 and 7 are set to zero. 1 TRANSFER_STATUS_HALTED This means that the endpoint that this transfer was attempted on is currently halted. 2 TRANSFER_STATUS_DATA_BUFFER_ERROR There was a data buffer under/over run. 3 TRANSFER_STATUS_BABBLE A babble was detected on the transfer. This could be either Frame babble or Packet babble or both. 4 Reserved 5 TRANSFER_STATUS_NOT_FOUND Returned as a response to an Abort Transfer request that has an invalid or already completed TransferID. 6 TRANSFER_STATUS_INSUFFICIENT_RESOURCE Returned in the transfer result when the Wire Adapter could not get enough resources to complete a previously accepted transfer request. Note unless mentioned otherwise, bit 7 is set for all status values.

Table 8-15. Transfer/Packet Status (cont.)

Bit	Description	
5:0	7	TRANSFER_STATUS_TRANSACTION_ERROR Returned in the transfer result when the Wire Adapter encountered a transaction error while performing this transfer. **Bits** **Description** 7:6 Indicates whether this was an error or a warning. **Value** **Meaning** 00B Undefined 01B The transfer completed successfully but transaction errors occurred which were successfully retried. 10B The transaction failed after the number of retry attempts specified in *bmRetryOptions* field of the RPipe descriptor. 11B Undefined Timeout, Bad PID, CRC error are examples of DWA transaction errors. Timeout, Bad PID, FCS error, Bad sequence number are examples of HWA transaction errors.
	8	TRANSFER_STATUS_ABORTED The transfer was aborted by an Abort Transfer Request or by an AbortRPipe command.
	9	TRANSFER_STATUS_RPIPE_NOT_READY The transfer request was sent to an unconfigured RPipe.
	10	INVALID_REQUEST_FORMAT This status may be sent back for one of two reasons: • The transfer request length was not equal to the length field for the specified request type • The request type was unknown.
	11	UNEXPECTED_SEGMENT_NUMBER The transfer request segment numbers were not received in incrementing order starting with zero.
	12	TRANSFER_STATUS_RPIPE_TYPE_MISMATCH The transfer type in the transfer request did not match the transfer type that the RPipe was previously configured to.
	13-63	Reserved
6	**Warning** This bit is set when the status is warning.	
7	**Error** This bit is set when the status is error	

8.3.3.5 Abort Transfer

The Abort Transfer request shown in Table 8-16 allows the host to abort a specific transfer. When the Wire Adapter receives this request, it will abort the specified transfer, send a Transfer completion notification, Transfer Result and any data that it received and acknowledged from the targeted endpoint back to the host. The Transfer Result must indicate that the transfer was aborted and the number of bytes that were sent or received before the request was aborted. The Abort Transfer request itself is acknowledged by the Wire Adapter when it ACKs the request.

Table 8-16. Abort Transfer Request

Offset	Field	Size	Value	Description
0	*bLength*	1	08H	Length of this request
1	*bRequestType*	1	84H	REQUEST_TYPE_ABORT – abort the specified transfer
2	*wRPipe*	2	Number	RPipe on which the transfer must be aborted
4	*dwTransferID*	4	Number	Host-assigned ID for the transfer request to be aborted

8.4 DWA Interfaces, Descriptors and Control

This section provides details on the DWA specific interfaces and includes all the descriptors that a DWA should present to host software. It also defines the DWA class specific control transfers and describes how isochronous streaming is supported on a DWA.

8.4.1 DWA Isochronous Streaming Interface

This interface is reported only by Device Wire Adapters that support isochronous endpoints on downstream connected devices. The number of isochronous endpoints in this interface determines the number of downstream isochronous streams the DWA can support simultaneously. The DWA isochronous endpoints have special configuration procedures described in the next section. However, once they are configured they behave like standard Wireless USB isochronous endpoints as described in Section 4.11.

8.4.2 DWA Isochronous Streaming Overview

A Device Wire Adapter supports isochronous transfers only if it has an upstream Wireless USB isochronous endpoint. When Host software is requested to transfer Isochronous data from a device connected downstream of a DWA, the host software driver for the DWA will establish a corresponding Isochronous stream to an upstream Wireless USB isochronous endpoint on the DWA. The host will transfer the data as a stream to/from the DWA. For isochronous OUT data transfers, the DWA will start performing the transfer as per the timestamp information present in Wireless USB header that is sent with every Wireless USB isochronous packet. For isochronous IN data transfers, the DWA will start performing the transfer as soon as it receives the first IN transaction on the upstream Wireless USB bus. The association from the downstream endpoint to its corresponding upstream endpoint is specified in the RPipe descriptor. See Section 8.4.6 for details.

8.4.3 DWA Descriptors

Device Wire Adapters must support standard Wireless USB device commands as defined in the Framework chapter of this specification.

8.4.3.1 Device Descriptor

Table 8-17. Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	12H	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	1	DEVICE Descriptor Type.
2	*bcdUSB*	2	250H	Wireless USB Specification Release Number in Binary-Coded Decimal. This field identifies the release of the Wireless USB Specification with which this device and its descriptors are compliant.
4	*bDeviceClass*	1	Number	

Table 8-17. Device Descriptor (cont.)

Offset	Field	Size	Value	Descriptor
5	*bDeviceSubClass*	1	Number	
6	*bDeviceProtocol*	1	Number	
7	*bMaxPacketSize0*	1	FFH	Maximum packet size for endpoint zero.
8	*idVendor*	2	ID	Vendor ID (assigned by the USB-IF)
10	*idProduct*	2	ID	Product ID (assigned by manufacturer)
12	*bcdDevice*	2	BCD	Device release number in binary-coded-decimal
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer
15	*iProduct*	1	Index	Index of string descriptor describing product
16	*iSerialNumber*	1	Index	Index of string descriptor describing product serial number
17	*bNumConfigurations*	1	1	Number of possible configurations

If the Device Wire Adapter exports an Isochronous interface then it must use an Interface Association Descriptor to group the two interfaces (Data Transfer Interface and Isochronous Streaming Interface) together so that one driver is loaded for both. In such a case the Device Wire Adapter must set *bDeviceClass*, *bDeviceSubClass* and *bDeviceProtocol* fields to EFH, 02H and 02H respectively. This class code is defined as the Wire Adapter Multifunction Peripheral (WAMP) class code.

If the Device Wire Adapter does not support downstream Isochronous endpoints then *bInterfaceClass*, *bInterfaceSubClass* and *bInterfaceProtocol* identify the DWA interface on this device.

8.4.3.2 Binary Device Object (BOS) Descriptor

A Device Wire Adapter must define a BOS descriptor. Host software can read the BOS descriptor using the GetDescriptor() request with a descriptor type set to BOS.

A Device Wire Adapter must always have a Wireless USB Device Capabilities on UWB descriptor as part of its BOS Descriptor set. The values within the Wireless USB Device Capabilities on UWB descriptor are implementation specific, see Section 7.4.1.

8.4.3.3 Configuration Descriptor

Table 8-18. Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	2	CONFIGURATION Descriptor Type
2	*wTotalLength*	2	Number	Total length of all descriptors in this configuration
4	*bNumInterfaces*	1	Number	Number of interfaces included in this configuration
5	*bConfigurationValue*	1	Number	Value to use to reference this configuration
6	*iConfiguration*	1	Index	Index of String Descriptor describing this configuration

Table 8-18. Configuration Descriptor (cont.)

Offset	Field	Size	Value	Description
7	*bmAttributes*	1	Bitmap	Configuration characteristics **Bit** **Description** 3:0 Reserved (reset to zero) 4 Battery-powered 5 Remote Wakeup 6 Self-powered 7 Reserved (set to one) Self-powered (D6) must always be set to a one (1B). If the DWA supports remote wakeup, D5 must be set to one.
8	*bMaxPower*	1	0	Reserved and zero for Wireless USB Device.

8.4.3.4 Security Descriptors

The Device Wire Adapter has security capabilities of its own (on its logical upstream port), it reports those by responding to a Get Descriptor (SECURITY type) request, see Section 7.4.5.

8.4.3.5 Interface Association Descriptor

If the Device Wire Adapter has an Isochronous Streaming interface then it must use an Interface Association Descriptor to describe the two interfaces that it uses so that host software can enumerate the device correctly.

Table 8-19. Interface Association Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	8	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	0BH	INTERFACE_ASSOCIATION Descriptor Type
2	*bFirstInterface*	1	0	Interface number of the first interface
3	*bInterfaceCount*	1	2	Number of contiguous interfaces associated with this function. This count includes this first interface as well.
4	*bFunctionClass*	1	E0H	Wireless Controller
5	*bFunctionSubClass*	1	02H	Wireless USB Wire Adapter
6	*bFunctionProtocol*	1	02H	Device Wire Adapter Control/Data Streaming interface
7	*iFunction*	1	Index	Index of a string descriptor that describes this Wire Adapter

8.4.3.6 Data Transfer Interface Descriptor

The Device Wire Adapter interface descriptor and the other descriptors that are part of this interface describe the endpoints on the DWA that are necessary to communicate with devices connected downstream of the Device Wire Adapter.

Table 8-20. Data Transfer Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	4	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	0	Number of this interface.
3	*bAlternateSetting*	1	0	Value used to select this alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	3	Number of endpoints used by this interface.
5	*bInterfaceClass*	1	E0H	Wireless Controller
6	*bInterfaceSubclass*	1	02H	Wireless USB Wire Adapter
7	*bInterfaceProtocol*	1	02H	Device Wire Adapter Control/Data Streaming interface
8	*iInterface*	1	Index	Index of String Descriptor describing this interface

8.4.3.7 Wire Adapter Class Descriptor

This descriptor describes the characteristics of the DWA to host software. This includes but is not limited to the amount of buffering available on the DWA, the number of RPipes, the number of ports and if ports are user accessible.

Table 8-21. Wire Adapter Class Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	21H	Wire Adapter Descriptor Type
2	*bcdWAVersion*	2	0100H	WA Class Specification Release Number in Binary-Coded Decimal. This field identifies the release of the WA Class Specification with which this interface is compliant.
4	*bNumPorts*	1	Number	The number of ports supported by this Wire Adapter

Table 8-21. Wire Adapter Class Descriptor (cont.)

Offset	Field	Size	Value	Description
5	*bmAttributes*	1	Bitmap	**Bit** — **Description** 0 — Logical Power Switching Mode 0: Ganged power switching (all ports' power at once) 1: Individual port power switching 1 — Over-current Protection Mode 0: Global Over-current Protection. The Wire Adapter reports over-current as a summation of all ports' current draw, without a breakdown of individual port over-current status. 1: Individual Port Over-current Protection. The Wire Adapter reports over-current on a per-port basis. Each port has an over-current status. 2 — Port Indicators Supported 0: Port Indicators are not supported on its downstream facing ports and the PORT_INDICATOR request has no effect. 1: Port Indicators are supported on its downstream facing ports and the PORT_INDICATOR request controls the indicators. 6:3 — Reserved, reset to zero 7 — Reserved
6	*wNumRPipes*	2	Number	The number of RPipes supported by this Wire Adapter
8	*wRPipeMaxBlock*	2	Number	The maximum number of buffer blocks assignable to all RPipes.
10	*bRPipeBlockSize*	1	Number	The size of an RPipe buffer block, expressed In the form $2^{\text{bRpipeBlockSize-1}}$ bytes per block. For example, a value of 10 would be 512.
11	*bPwrOn2PwrGood*	1	Number	Time (in 2 ms intervals) from the time the power-on sequence begins on a port until power is good on that port. The USB System Software uses this value to determine how long to wait before accessing a powered-on port.
12	*bNumMMCIEs*	1	Number	This field is not used by DWAs and must be set to 0.

Table 8-21. Wire Adapter Class Descriptor (cont.)

Offset	Field	Size	Value	Description
13	*DeviceRemovable*	Variable	Bitmap	Indicates if a port has a removable device attached. This field is reported on byte-granularity. Within a byte, if no port exists for a given location, the field representing the port characteristics returns 0. Bit value definition: 0B Device is removable. 1B Device is non-removable This is a bitmap corresponding to the individual ports on the hub: Bit 0 Reserved. Bit 1 Port 1 Bit 2 Port 2 .... Bit n Port n (implementation-dependent, up to a maximum of 127 ports).

8.4.3.8 Notification Endpoint Descriptor

This endpoint is used to report port status change, Wire Adapter status and transfer completion notifications.

Table 8-22. Notification Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	03H	Normal power interrupt endpoint
4	*wMaxPacketSize*	2	200H	Maximum packet size of this endpoint
6	*bInterval*	1	6	Interval for polling endpoint for data transfers. See Section 7.4.3.

8.4.3.9 Notification Endpoint Companion Descriptor

Table 8-23. Notification Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes
1	*bDescriptorType*	1	17	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	1	The max burst size of this endpoint
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32.
4	*wMaxStreamDelay*	2	00H	Maximum supported stream delay. The field is reserved and not used for Interrupt endpoints.

Table 8-23. Notification Endpoint Companion Descriptor (cont.)

Offset	Field	Size	Value	Description
6	*wOverTheAirPacketSize*	2	00H	Maximum packet size this endpoint is capable of receiving over the air. The field is reserved and not used for Interrupt endpoints.
8	*bOverTheAirInterval*	1	00H	Interval for polling endpoint for data transfers. The field is reserved and not used for Interrupt endpoints.
9	*bmCompAttributes*	1	00H	The field is reserved and not used for Interrupt endpoints.

8.4.3.10 Data Transfer Write Endpoint Descriptor

Table 8-24. Data Transfer Write Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	X0000010b. Bit 7 is set if Data packet size adjustment supported on this Wire Adapter
4	*wMaxPacketSize*	2	Number	Maximum packet size of this endpoint
6	*bInterval*	1	0	Polling not supported.

8.4.3.11 Data Transfer Write Endpoint Companion Descriptor

Table 8-25. Data Transfer Write Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes
1	*bDescriptorType*	1	17	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	Number	The max burst size of this endpoint
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32.
4	*wMaxStreamDelay*	2	00H	Maximum supported stream delay. The field is reserved and not used for Bulk endpoints.
6	*wOverTheAirPacketSize*	2	00H	Maximum packet size this endpoint is capable of receiving over the air. The field is reserved and not used for Bulk endpoints.
8	*bOverTheAirInterval*	1	00H	Interval for polling endpoint for data transfers. The field is reserved and not used for Bulk endpoints.
9	*bmCompAttributes*	1	00H	The field is reserved and not used for Bulk endpoints.

8.4.3.12 Data Transfer Read Endpoint Descriptor

Table 8-26. Data Transfer Read Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	X0000010b. Bit 7 is set if Data packet size adjustment supported on this Wire Adapter
4	*wMaxPacketSize*	2	Number	Maximum packet size of this endpoint
6	*bInterval*	1	0	Polling not supported.

8.4.3.13 Data Transfer Read Endpoint Companion Descriptor

Table 8-27. Data Transfer Read Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes
1	*bDescriptorType*	1	17	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	Number	The max burst size of this endpoint
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32.
4	*wMaxStreamDelay*	2	00H	Maximum supported stream delay. The field is reserved and not used for Bulk endpoints.
6	*wOverTheAirPacketSize*	2	00H	Maximum packet size this endpoint is capable of receiving over the air. The field is reserved and not used for Bulk endpoints.
8	*bOverTheAirInterval*	1	00H	Interval for polling endpoint for data transfers. The field is reserved and not used for Bulk endpoints.
9	*bmCompAttributes*	1	00H	The field is reserved and not used for Bulk endpoints.

8.4.3.14 Isochronous Streaming Interface Descriptor

This interface and its associated descriptors are optional. They are only present on Device Wire Adapters that support isochronous endpoints on downstream devices.

Table 8-28. Isochronous Streaming Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	4	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	1	Number of this interface.
3	*bAlternateSetting*	1	0	Value used to select this alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	Number	Number of endpoints used by this interface. Legal values are 1 through 28.
5	*bInterfaceClass*	1	E0H	Wireless Controller
6	*bInterfaceSubclass*	1	02H	Wireless USB Wire Adapter
7	*bInterfaceProtocol*	1	03H	Device Wire Adapter Isochronous Streaming Interface
8	*iInterface*	1	Index	Index of String Descriptor describing this interface

8.4.3.15 Isochronous Streaming OUT Endpoint Descriptor

Table 8-29. Isochronous Streaming OUT Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	01H	Isochronous endpoint.
4	*wMaxPacketSize*	2	Number	The logical *MaxPacketSize* must be set to the same value as the-over-the air *MaxPacketSize*.
6	*bInterval*	1	0	The logical service Interval must be set to the same value as the over-the-air service interval.

8.4.3.16 Isochronous Streaming OUT Endpoint Companion Descriptor

Table 8-30. Isochronous Streaming OUT Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes
1	*bDescriptorType*	1	17	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	Number	The value of this field must adhere to the rules specified in Section 4.7.1
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32.
4	*wMaxStreamDelay*	2	Number	Maximum supported stream delay. The actual stream delay is determined by the DWA driver.
6	*wOverTheAirPacketSize*	2	Number	Maximum packet size this endpoint is capable of receiving over the air.
8	*bOverTheAirInterval*	1	00H	Interval for polling endpoint for data transfers. Since this is a dynamic switching capable endpoint, this endpoint supports all valid values from 4 through 255.
9	*bmCompAttributes*	1	02H	This endpoint supports continuously scalable dynamic switching.

8.4.3.17 Isochronous Streaming IN Endpoint Descriptor

Table 8-31. Isochronous Streaming IN Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	01H	Isochronous endpoint.
4	*wMaxPacketSize*	2	Number	The logical MaxPacketSize must be set to the same value as the over the air MaxPacketSize.
6	*bInterval*	1	0	The logical service Interval must be set to the same value as the over-the-air service interval.

8.4.3.18 Isochronous Streaming IN Endpoint Companion Descriptor

Table 8-32. Isochronous Streaming IN Endpoint Companion Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes
1	*bDescriptorType*	1	17	WIRELESS_ENDPOINT_COMPANION Descriptor Type
2	*bMaxBurst*	1	Number	The value of this field must adhere to the rules specified in Section 4.7.1
3	*bMaxSequence*	1	Number	The maximum sequence used for data bursting. Valid values are in the range 2 to 32.
4	*wMaxStreamDelay*	2	Number	Maximum supported stream delay. The actual stream delay is determined by the DWA driver.
6	*wOverTheAirPacketSize*	2	Number	Maximum packet size this endpoint is capable of sending over the air.
8	*bOverTheAirInterval*	1	00H	Interval for polling endpoint for data transfers. Since this is a dynamic switching capable endpoint, this endpoint supports all valid values from 4 through 255.
9	*bmCompAttributes*	1	02H	This endpoint supports continuously scalable dynamic switching.

8.4.3.19 Wire Adapter RPipe Descriptor

The Wire Adapter RPipe descriptors are not returned as part of the configuration descriptor for a DWA. Host software can get each RPipe descriptor by sending a Get RPipe Descriptor (See Section 8.3.1.4) request to the DWA. The format of the Wire Adapter RPipe descriptor and the description of the fields is given in Table 8-33.

Table 8-33. Wire Adapter RPipe Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	1CH	Size of this descriptor in bytes, including this field. This is a read only field.
1	*bDescriptorType*	1	22H	Wire Adapter RPipe Descriptor Type. This is a read only field.
2	*wRPipeIndex*	2	Number	Number of this RPipe. Zero-based value identifying the index in the array of concurrent RPipes supported by this Wire Adapter. This is a read only field.
4	*wRequests*	2	Number	The number of concurrent requests that can be assigned to this RPipe. This is a read only field.
6	*wBlocks*	2	Number	The number of buffer blocks assigned to this RPipe. If the value in this field is zero then the Set RPipe Descriptor request can be used to set the number of blocks to be assigned to this RPipe. If the value in this field is not zero then the number of blocks assigned to this RPipe cannot be changed by host software. This field may be modified by host software if this is set to zero.

Table 8-33. Wire Adapter RPipe Descriptor (cont.)

Offset	*Field*	Size	Value	Description
8	*wMaxPacketSize*	2	Number	Maximum packet size that this RPipe will use to communicate with attached device.
10	*bHSHubAddress*	1	Number	This is the address of the attached device's parent high speed hub. This number is used only when a full/low device is connected to the DWA through a high speed hub. This field must be set to Zero if the full/low speed device is directly connected to the DWA. This field is reserved if the *bSpeed* field is set to High Speed.
11	*bHSHubPort*	1	Number	This is the Port number on which the attached device is connected to its parent high speed hub. This number is used only when a full/low device is connected to the DWA through a high speed hub. This field must be set to Zero if the full/low speed device is directly connected to the DWA. This field is reserved if the *bSpeed* field is set to High Speed.
12	*bSpeed*	1	Number	The speed of device to be targeted by RPipe. **Value** **Description** 00B Full-Speed (12Mbs) 01B Low-Speed (1.5Mbs) 10B High-Speed (480 Mbs) 11B Reserved
13	*bDeviceAddress*	1	Number	Address to be used with attached device
14	*bEndpointAddress*	1	Number	Endpoint Address to be used with this RPipe. **Bit** **Description** 3:0 The endpoint number 6:4 Reserved; set to zero 7 Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint
15	*bDataSequence*	1	Number	Current data sequence. This is the next data sequence value to be used when sending data to the endpoint that this RPipe is targeted at.
16	*dwCurrentWindow*	4	Number	Reserved and must be set to zero.
20	*bMaxDataSequence*	1	Number	Reserved and must be set to zero.
21	*bInterval*	1	Number	Polling interval to be used by this RPipe in downstream communications
22	*bOverTheAirInterval*	1	Number	If the transfer type is Isochronous, then this is the interval at which the upstream wireless endpoint is polled. See *bOverTheAirInterval* in Table 7-32 for the encoding of this field. This field is Reserved and must be set to zero for all other transfer types.

Table 8-33. Wire Adapter RPipe Descriptor (cont.)

Offset	*Field*	Size	Value	Description
23	*bmAttribute*	1	Bitmap	**Bit** / **Description** 1:0 — **Value** / **Transfer Type** 00B Control 01B Isochronous 10B Bulk 11B Interrupt 5:2 — If the transfer type is Isochronous, then this field indicates the associated upstream Isochronous endpoint on the DWA. This field is Reserved and must be set to zero for all other transfer types. 7:6 — Reserved
24	*bmCharacteristics*	1	Bitmap	Transfer types supported on this RPipe **Bit** / **Description** 0 — 1: Control Transfer supported 0: Control Transfer NOT supported 1 — 1: Isochronous Transfer supported 0: Isochronous Transfer NOT supported 2 — 1: Bulk Transfer supported 0: Bulk Transfer NOT supported 3 — 1: Interrupt Transfer supported 0: Interrupt Transfer NOT supported 7:4 — Reserved This is a read only field.
25	*bmRetryOptions*	1	Bitmap	**Bit** / **Description** 2:0 — The maximum number of times a transaction must be retried before the transfer request is failed. The valid values are 0 through 3. A value of zero in this field indicates that the DWA must not count errors and there is no limit on the retries. For Isochronous transfers this field is set to zero. 7:3 — Reserved.
26	*wNumTransactionErrors*	2	Number	The DWA increments this field when it encounters an error while performing transactions to the downstream endpoint targeted by this RPipe. The host is responsible for resetting this field.

All fields that are not marked read only may be changed by host software by using the Set RPipe Descriptor request.

8.4.4 DWA Specific Requests

This section describes all the DWA specific requests.

Table 8-34. DWA Specific Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
Clear Port Feature	00100100B	CLEAR_FEATURE	Feature Selector	Selector and Port Index	Zero	None
Get Port Status	10100100B	GET_STATUS	Zero	Port Index	4	Port Status and Change Status
Set ISOEP Attributes	00100010B	SET_EP_ATTRIB	Zero	Endpoint Address	6	Endpoint Attributes
Set Port Feature	00100100B	SET_FEATURE	Feature Selector	Selector and Port Index	Zero	None

Table 8-35. DWA Specific Request Codes

bRequest	Value
SET_EP_ATTRIB	30

8.4.4.1 Clear Port Feature

This request resets a value in the reported port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100100B	CLEAR_FEATURE	Feature Selector	Selector and Port Index	Zero	None

The *wIndex* field contains a Selector type in the upper byte and a Port Index in the lower byte.

Clearing a feature disables that feature; see Table 11-17 in the USB 2.0 specification for the feature selector definitions that apply to a port as a recipient. This request format is used to clear the following features:

- PORT_ENABLE
- PORT_SUSPEND
- PORT_POWER
- PORT_INDICATOR
- C_PORT_CONNECTION
- C_PORT_RESET
- C_PORT_ENABLE
- C_PORT_SUSPEND
- C_PORT_OVER_CURRENT

See Section 11.24.2.2 of the USB 2.0 Specification for a detailed description on the usage of the Selector in *wIndex*.

It is a Request Error if *wValue* is not a feature selector listed in Table 8-6, if *wIndex* specifies a port that does not exist, or if *wLength* is not as specified above.

If the Device Wire Adapter is not configured, the Device Wire Adapter's response to this request is undefined.

8.4.4.2 Get Port Status

This request returns the current port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100100B	GET_STATUS	Zero	Port Index	4	Port Status and Change Status

The *wIndex* field contains a Port Index. The port index must be a valid port index for that Device Wire Adapter, greater than zero.

The first word of data contains *wPortStatus* (refer to Table 11-21 in the USB 2.0 specification). The second word of data contains *wPortChange* (refer to Table 11-20 in the USB 2.0 specification). The bit locations in the *wPortStatus* and *wPortChange* fields correspond in a one-to-one fashion.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies a Port that does not exist.

If the Device Wire Adapter is not configured, the Device Wire Adapter's response to this request is undefined.

8.4.4.3 Set Isochronous Endpoint Attributes

This request sets the *wMaxStreamDelay* and *wOverTheAirPacketSize* for the continuously scalable isochronous endpoint specified on the DWA.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100010B	SET_EP_ATTRIB	Zero	Endpoint Address	6	Endpoint Attributes

On reception of this request, the DWA will expect to receive or send data as per the *wOverTheAirPacketSize* specified. The lower byte of *wIndex* specifies the target endpoint. The format of the endpoint attributes structure is given in Table 8-36.

Table 8-36. Endpoint Attributes Buffer Format

Offset	Field	Size	Value	Description
0	*wMaxStreamDelay*	2	Number	The actual stream delay as determined by the host. This field indicates the amount of delay in 128 microsecond units. See Table 7-32 for details.
2	*wOverTheAirPacketSize*	2	Number	New Maximum packet size this endpoint is capable of sending or receiving over the air This must be less than or equal to the original over-the-air Maximum packet size.
4	*wReserved*	2	Zero	Reserved for future use, must be zero.

The host must ensure that the buffering allocated to the RPipe is sufficient to support the *wMaxStreamDelay* value in this request.

It is a Request Error if *wValue* or *wLength* are other than as specified above.

If the endpoint specified does not exist, then the device responds with a Request Error.

If the Device Wire Adapter is not configured, the Device Wire Adapter's response to this request is undefined.

8.4.4.4 Set Port Feature

This request sets a value in the reported port status.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100100B	SET_FEATURE	Feature Selector	Selector and Port Index	Zero	None

The *wIndex* field contains a Selector in the upper byte and a Port Index in the lower byte. The port index must be a valid port index for that Device Wire Adapter, greater than zero.

Setting a feature enables that feature or starts a process associated with that feature; Table 11-17 in the USB 2.0 specification for the feature selector definitions that apply to a port as a recipient. Features that can be set with this request are:

- PORT_RESET
- PORT_SUSPEND
- PORT_POWER
- PORT_TEST
- PORT_INDICATOR

See Section 11.24.2.2 of the USB 2.0 Specification for a detailed description on the usage of the Selector in *wIndex*.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if *wIndex* specifies a Port that does not exist.

If the Device Wire Adapter is not configured, the Device Wire Adapter's response to this request is undefined.

8.4.5 DWA Notification Information

Device Wire Adapters must send back notifications on its Notification Endpoint for Remote Wake and Port Status Changes. The format of each notification is detailed below.

8.4.5.1 Remote Wake

When the Device Wire Adapter detects a remote wake from any of its downstream connected devices and it is armed for remote wake then it should send a Remote Wake notification to the host. The format of this notification is shown in Table 8-37.

Table 8-37. Remote Wake Notification

Offset	Field	Size	Value	Description
0	*bLength*	1	2	Length of this block of data
1	*bNotifyType*	1	91H	NOTIFY_TYPE_RWAKE

8.4.5.2 Port Status Change

A Device Wire adapter must send a Port Status Change notification when the status of a downstream port on the Device Wire Adapter changes. The format of this notification is shown in Table 8-38.

Table 8-38. Port Status Change Notification

Offset	Field	Size	Value	Description
0	*bLength*	1	3	Length of this block of data
1	*bNotifyType*	1	92H	NOTIFY_TYPE_PORTSTATUS
2	*bPortIndex*	1	Number	The Index of the Port on which a Port change occurred.

8.4.6 DWA Isochronous Transfers

When host software wants to get data or send data to an isochronous endpoint connected downstream of a DWA, it will map a wireless USB isochronous endpoint on the DWA isochronous streaming interface to the wired USB endpoint. This mapping is accomplished via the *bmAttribute* field in the RPipe descriptor. The DWA uses the *bInterval* field in the RPipe descriptor to determine the frequency with which to service the downstream wired endpoint. The host will service the upstream wireless Isochronous endpoint on the DWA as specified in the *bOverTheAirInterval* field in the RPipe descriptor. The DWA will accept Wireless USB isochronous data packets on its isochronous endpoints. The DWA interacts with the downstream wired isochronous endpoints such that the upstream wireless isochronous endpoints they are associated with behave like native Wireless USB isochronous endpoints. Note that there is no transfer result or packet status for an isochronous transfer. Data that was read from a downstream connected Isochronous IN endpoint is sent back via its associated Wireless Isochronous endpoint on the DWA.

8.4.6.1 DWA Isochronous OUT Responsibilities

The DWA parses packets received on one of its Wireless USB isochronous OUT endpoints using the Wireless USB isochronous packet header format. The DWA sends the data to the downstream wired isochronous endpoint at the specified (micro)frame based on the information in the packet header and the *bInterval* value specified in the RPipe descriptor.

Figure 8-8 illustrates a High-speed Isochronous OUT data stream through a DWA. The illustration is organized with time flowing from left to right and data flow from top to bottom, where the top illustrates a packet payload of a Wireless USB transaction to a DWA isochronous endpoint, down through a DWA RPipe buffer and finally over the USB 2.0 bus to the recipient endpoint.

Starting from the top, the host sends an isochronous packet to the isochronous function endpoint on the DWA associated with the particular isochronous stream. The format of the packet is the standard isochronous data format (see Table 5-1and Table 5-2 for the definition details). In this example, it contains 32 isochronous data segments (*Data – 1* through *Data – 32*), each of which translates to a micro-frame data payload over the wire.

The DWA divides the associated RPipe buffer into interval segments, as illustrated in this example into 4ms segments. All of the data received during an interval is placed into the RPipe buffer, organized into data groups of eight isochronous segments. Each group is also annotated with the appropriate presentation time, which is the USB 2.0 frame value during which the DWA must transmit the data group over the wire. It is the responsibility of the host to ensure the isochronous data is sent over the Wireless USB channel in a timely fashion (i.e. before the presentation times of the data become invalid).

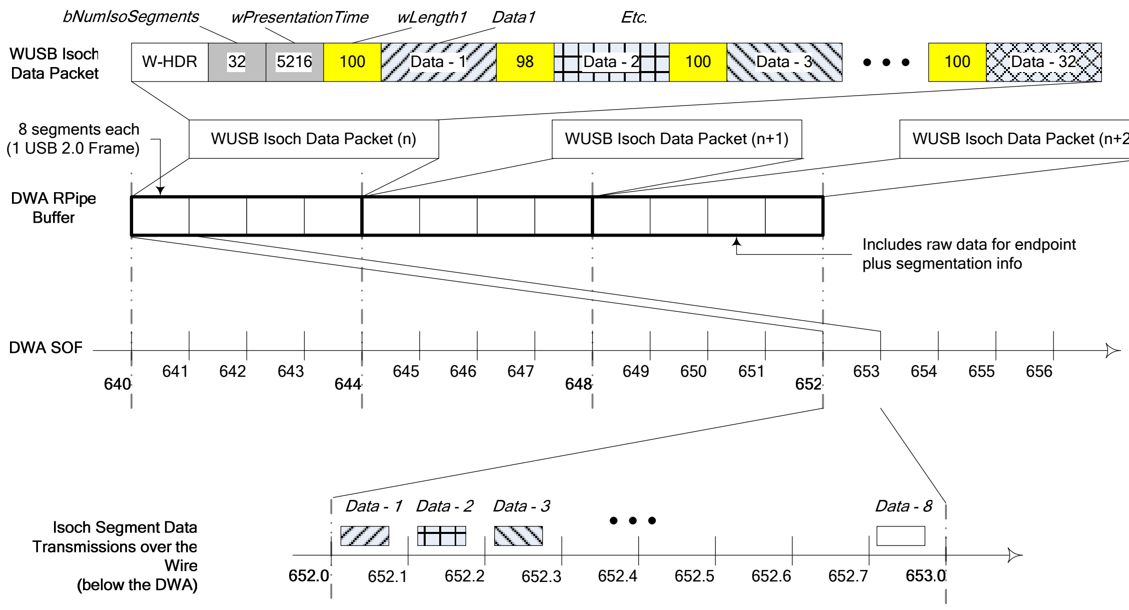

Figure 8-8. High-speed Isochronous OUT Data Stream through a DWA

When the presentation time equals the downstream frame value, the DWA will transmit the isochronous data segments over the wire. As illustrated, the first presentation time was 5216 (Wireless USB Channel time), so the DWA transmits an isochronous OUT transaction during 652.0 (micro-frame 0) and sends *Data -1*. In the next microframe (652.1) it transmits *Data – 2*, and so-on through *Data – 8* in microframe 652.7. In the next frame, the DWA begins transmitting the queued data from the next group. This streaming model continues until the host ceases feeding the stream or errors cause the host to discard and skip late data. Those scenarios are described in detail in Section 4.11.9.

8.4.6.2 DWA Isochronous IN Responsibilities

A DWA performs isochronous IN requests to the downstream wired endpoint every *bInterval* (as specified in the RPipe descriptor) (micro)frames. The first downstream IN is performed when the first Wireless USB Isochronous IN request for the associated upstream Wireless USB isochronous IN endpoint is received.

A DWA must aggregate wired isochronous packet data into the largest packets that can be sent over the air by the associated Wireless USB Isochronous endpoint. It must not split data from a single (micro)frame across multiple over-the-air packets.

Presentation times for over-the-air isochronous packets are determined by the (micro)frame for which the first data segment in the Wireless USB isochronous packet was sampled on the wired downstream bus. The DWA responds with the oldest data in its RPipe buffer for each Wireless USB Isochronous IN request. It only discards data if the buffer associated with the RPipe for that endpoint overflows.

Figure 8-9 illustrates a Full-speed Isochronous IN data stream through a DWA. The top of the figure is a timeline illustrating the SOFs transmitted downstream to the USB 2.0 device connected below the DWA. The SOF values transmitted downstream of the DWA must match the Wireless USB channel times on the bus between the DWA and its host.

The illustration is organized with time flowing from left to right and data flowing top to bottom. The DWA will not begin generating IN Tokens to the downstream device until the host begins polling for data on the associated DWA isochronous data stream function endpoint. The left-hand most W_{DT}CTA represents the first time the host begins polling the DWA isochronous function endpoint. The DWA does not have any data to respond to this transaction token, so will respond with a NAK handshake. This directs the host to discontinue polling the endpoint until the next service interval (designated by the vertical dotted lines).

In response to this first poll attempt by the host (which occurs before frame 642), the DWA begins transactions to the downstream USB 2.0 function endpoint in the next frame (i.e. 643). In each frame after 642, the DWA conducts isochronous IN transactions to the USB 2.0 function endpoint. The DWA continually stores all of the data received during the interval into the RPipe buffer and records the frame time of the first data received from the USB 2.0 device during the interval (i.e. the interval *N*).

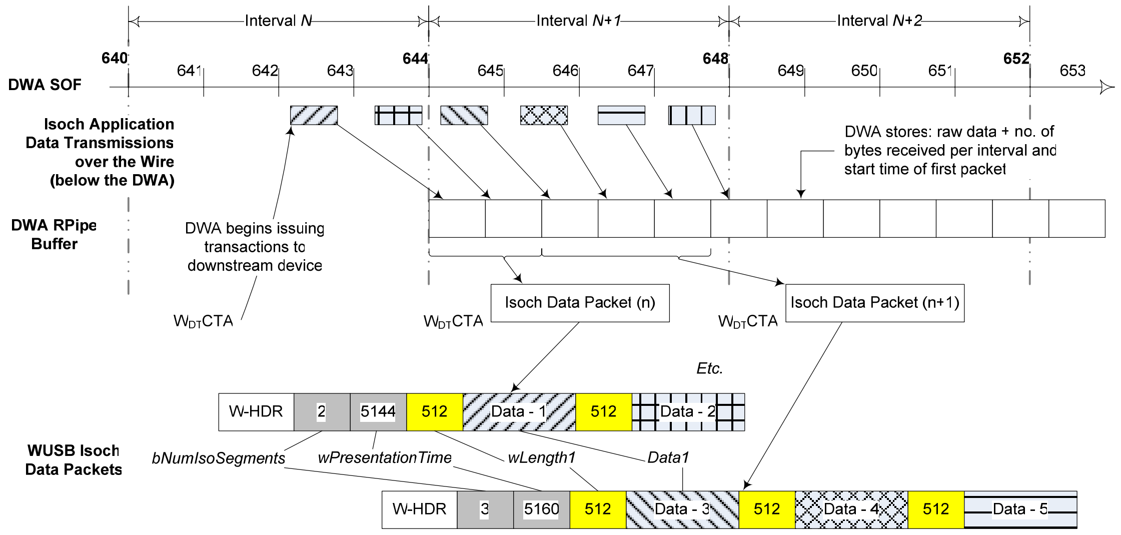

Figure 8-9. Full-speed Isochronous IN Data Stream through a DWA

In the next interval (*N+1*), the host polls the DWA's isochronous function endpoint and the DWA returns the data it has ready to transmit. At the time the W_{DT}CTA during interval *N+1* is transmitted by the host, the DWA has not completely received any data from the USB 2.0 function endpoint during the interval, so the DWA only transmits the complete data it does have (i.e. two isochronous data segments, beginning at 643). The format of the resultant data packet is illustrated. Since 2 packets were received from the USB 2.0 function endpoint in interval *N*, in response to the second W_{DT}CTA poll, the DWA transmits a packet that includes the two samples received and the presentation time (5144 – the Wireless USB Channel time corresponding to the DWA SOF value of 643) of when data was first received. The host will attempt to pull data from the DWA's endpoint until the DWA function endpoint NAKs (not shown in interval *N+1*).

In the next interval (*N+2*), the host polls the DWA's isochronous function endpoint and receives the next set of isochronous segments ready (Data 3, 4 and 5), with the frame number where the first packet (i.e. Data-3 at 645) was received from the USB 2.0 function endpoint. Each interval, the host will pull data from the endpoint until either the DWA function endpoint NAKs or the host uses up all of the allowed number of transaction attempts allowed for the service interval. This streaming model continues until the host ceases feeding the transaction stream or errors cause the DWA to discard and skip late data. Those scenarios are described in detail in Section 4.11.9 of the Data Flow chapter.

Note, this is a rather simple example to illustrate the general flow. There is no intent to imply that a DWA implements a 2-deep interval pipeline. Depending on wireless channel condition, it may be several intervals behind the USB 2.0 channel.

8.5 HWA Interfaces, Descriptors and Control

This section provides details on the HWA specific interfaces, includes all the descriptors an HWA should present to host software, HWA class specific control transfers, additional notifications that an HWA can return to host software and describes how isochronous streaming is supported on an HWA.

8.5.1 HWA Isochronous Streaming Overview

On a Host Wire Adapter the basic data flow model for Isochronous pipes is identical to the Bulk streaming model. Host software sends "buffer oriented" transfer requests to the Wire Adapter. The Host Client model is already based on a client device driver queuing multiple requests to the Wired or Wireless USB Stack in order to achieve streaming. The host software can propagate the request stream without modification to the HWA. It is expected that host software will only have to subdivide client transfer requests into a series of smaller transfer requests depending on the buffering available on that RPipe. Note that although the transfer request model is 'buffer oriented' the transfer requests coming from the client already have per-frame annotations. Host software will include the packetization and *when to send/receive* (timestamp) information in the transfer request to the HWA. This is very important as only the client has knowledge of how per-frame annotations should be applied to a stream. For instance, a client may change the per-frame data amount in order to maintain synchronization. Note that a relative (as opposed to explicit, per packet) timestamp is sufficient.

For IN transfers, it is important that the feedback/response pipe provide sufficient information for the host software to map data packets received in specific frames to the appropriate client request buffer area. This means the HWA must annotate the received data stream with packet boundaries, upon receipt (timestamp) and correctly identify bad and/or missing packets. An isochronous transfer result is always followed by an isochronous packet status array. Note that relative timestamp information is sufficient. See Section 8.5.5 for details.

8.5.2 HWA Descriptors

Host Wire Adapters must support a class specific security descriptor, identical to the security descriptor returned by all wireless USB devices, to be used by the host to identify the encryption types supported on the host wire adapters logical down stream ports. It must return the security descriptor as part of its configuration descriptor. A Host Wire Adapter returns different descriptors based on whether it is operating at high-speed or full speed.

8.5.2.1 Device Descriptor

Table 8-39. Device Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	12H	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	1	DEVICE Descriptor Type.
2	*bcdUSB*	2	200H	USB Specification Release Number in Binary-Coded Decimal. This field identifies the release of the USB Specification with which the device and its descriptors are compliant.
4	*bDeviceClass*	1	EFH	Miscellaneous
5	*bDeviceSubClass*	1	02H	Common Class
6	*bDeviceProtocol*	1	02H	Wire Adapter Multifunction Peripheral
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero
8	*idVendor*	2	ID	Vendor ID (assigned by the USB-IF)

Table 8-39. Device Descriptor (cont.)

Offset	*Field*	Size	Value	Description
10	*idProduct*	2	ID	Product ID (assigned by manufacturer)
12	*bcdDevice*	2	BCD	Device release number in binary-coded-decimal
14	*iManufacturer*	1	Index	Index of string descriptor describing manufacturer
15	iProduct	1	Index	Index of string descriptor describing product
16	iSerialNumber	1	Index	Index of string descriptor describing product serial number
17	bNumConfigurations	1	1	Number of possible configurations

All Host Wire Adapters have a UWB Radio and hence have to export a Radio Control Interface as well (see Section 8.6). To correctly enumerate the HWA, it must set *bDeviceClass*, *bDeviceSubClass* and *bDeviceProtocol* fields to EFH, 02H and 02H respectively. This class code is defined as the Wire Adapter Multifunction Peripheral (WAMP) class code.

8.5.2.2 Device_Qualifier Descriptor

Table 8-40. Device_Qualifier Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	0AH	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	6	DEVICE_QUALIFIER Type
2	*bcdUSB*	2	200H	USB Specification Release Number in Binary-Coded Decimal. This field identifies the release of the USB Specification with which the device and its descriptors are compliant.
4	*bDeviceClass*	1	EFH	Miscellaneous
5	*bDeviceSubClass*	1	02	Common Class
6	*bDeviceProtocol*	1	02	Wire Adapter Multifunction Peripheral
7	*bMaxPacketSize0*	1	Number	Maximum packet size for endpoint zero.
8	*bNumConfigurations*	1	1	Number of possible configurations.
9	*bReserved*	1	Zero	Reserved for future use, must be zero.

8.5.2.3 Configuration Descriptor

Table 8-41. Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	2	CONFIGURATION Descriptor Type
2	*wTotalLength*	2	Number	Total length of all descriptors in this configuration
4	*bNumInterfaces*	1	Number	Number of interfaces included in this configuration
5	*bConfigurationValue*	1	Number	Value to use to reference this configuration
6	*iConfiguration*	1	Index	Index of String Descriptor describing this configuration

Table 8-41. Configuration Descriptor (cont.)

Offset	*Field*	Size	Value	Description
7	*bmAttributes*	1	Bitmap	Configuration characteristics D7: Reserved (set to one) D6: Self-powered D5: Remote Wakeup D4...0: Reserved (reset to zero)
8	*bMaxPower*	1	mA	Maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational. Expressed in 2 mA units (i.e., 50 = 100 mA).

8.5.2.4 Other_Speed_Configuration Descriptor

Table 8-42. Other_Speed_Configuration Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	7	Other_Speed_Configuration Descriptor Type
2	*wTotalLength*	2	Number	Total length of all descriptors in this configuration
4	*bNumInterfaces*	1	Number	Number of interfaces supported by this speed configuration.
5	*bConfigurationValue*	1	Number	Value to use to select configuration
6	*iConfiguration*	1	Index	Index of string descriptor
7	*bmAttributes*	1	Bitmap	Same as Configuration descriptor
8	*bMaxPower*	1	mA	Same as Configuration descriptor

8.5.2.5 Security Descriptors

A Host Wire Adapter must return a security descriptor and all its associated encryption descriptors in its configuration descriptor.

Table 8-43. Wire Adapter Class Security Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	5	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	12	Security Descriptor Type
2	*wTotalLength*	2	Number	Total length of this descriptor and all encryption descriptors returned
4	*bNumEncryptionTypes*	1	Number	Number of supported encryption Types

The Host Wire Adapter will return the number of Encryption descriptors as noted in the *bNumEncryptionsTypes* field immediately after the Security descriptor. Since this Encryption Descriptor is only used to inform the host software of the supported encryption methods, the *bAuthKeyIndex* field must be set to 0x0 in every Encryption Descriptor that is returned by the Wire Adapter.

NOTE: The security descriptor and its associated encryption descriptors that are returned as part of the configuration description are used only to determine the supported encryption methods of the Host Wire Adapter device on its logical downstream ports.

8.5.2.6 Data Transfer Interface Descriptor

Table 8-44. Data Transfer Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	4	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	0	Number of this interface.
3	*bAlternateSetting*	1	0	Value used to select this alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	3	Number of endpoints used by this interface.
5	*bInterfaceClass*	1	E0H	Wireless Controller
6	*bInterfaceSubclass*	1	02H	Wireless USB Wire Adapter
7	*bInterfaceProtocol*	1	01H	Host Wire Adapter Control/Data Streaming interface
8	*iInterface*	1	Index	Index of String Descriptor describing this interface

8.5.2.7 Wire Adapter Class Descriptor

This descriptor describes the characteristics of the HWA to host software. This includes but is not limited to the amount of buffering available on the HWA, the number of RPipes, the maximum number of IEs that the HWA has storage for and the number of devices that this HWA can have connected to it at the same time.

Table 8-45. Wire Adapter Class Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	21H	Wire Adapter Descriptor Type
2	*bcdWAVersion*	2	0100H	WA Class Specification Release Number in Binary-Coded Decimal. This field identifies the release of the WA Class Specification with which this interface is compliant.
4	*bNumPorts*	1	Number	The maximum number of simultaneous devices that this HWA can support.
5	*bmAttributes*	1	Bitmap	Reserved, must be set to zero.
6	*wNumRPipes*	2	Number	The number of RPipes supported by this Wire Adapter
8	*wRPipeMaxBlock*	2	Number	The maximum number of buffer blocks assignable to all RPipes.
10	*bRPipeBlockSize*	1	Number	The size of an RPipe buffer block, expressed In the form $2^{bRpipeBlockSize-1}$ bytes per block. For example, a value of 10 would be 512.
11	*bPwrOn2PwrGood*	1	Number	For Host Wire Adapters, this field must be set to 0.
12	*bNumMMCIEs*	1	Number	This field specifies the number of MMC IE blocks that a HWA can support at the same time. Each block must have at least 255 bytes of storage. Valid values are in the range of 1H to FFH. A zero in this field is undefined for an HWA.
13	*DeviceRemovable*	1	Bitmap	For Host Wire Adapters, this field is of length 1 and all bits are set to 0.

8.5.2.8 Notification Endpoint Descriptor

This endpoint is used to report all the Wireless USB Device Notifications received by the HWA, Wire Adapter status and transfer completion notifications.

Table 8-46. Notification Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	Interrupt endpoint of 00000011b.
4	*wMaxPacketSize*	2	40H	Maximum packet size for this endpoint
6	*bInterval*	1	1	Interval for polling endpoint for data transfers. Expressed in frames or microframes depending on the device operating speed (i.e., either 1 millisecond or 125 μs units).

8.5.2.9 Data Transfer Write Endpoint Descriptor

Table 8-47. Data Transfer Write Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	BULK endpoint of 00000010b.
4	*wMaxPacketSize*	2	Number	Maximum packet size this endpoint
6	*bInterval*	1	0	Polling not supported.

8.5.2.10 Data Transfer Read Endpoint Descriptor

Table 8-48. Data Transfer Read Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	BULK endpoint of 00000010b.
4	*wMaxPacketSize*	2	Number	Maximum packet size this endpoint
6	*bInterval*	1	0	Polling not supported.

8.5.2.11 Wire Adapter RPipe Descriptor

The Wire Adapter RPipe descriptors are not returned as part of the configuration descriptor for an HWA. Host software can get each RPipe descriptor by sending a Get RPipe Descriptor (See Section 8.3.1.4) request to the HWA. The format of the Wire Adapter RPipe descriptor and the description of the fields are given in Table 8-49.

Table 8-49. Wire Adapter RPipe Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	1CH	Size of this descriptor in bytes, including this field. This is a read only field.
1	*bDescriptorType*	1	22H	Wire Adapter RPipe Descriptor Type. This is a read only field.
2	*wRPipeIndex*	2	Number	Number of this RPipe. Zero-based value identifying the index in the array of concurrent RPipes supported by this Wire Adapter. This is a read only field.
4	*wRequests*	2	Number	The number of concurrent requests that can be assigned to this RPipe. This is a read only field.
6	*wBlocks*	2	Number	The number of buffer blocks assigned to this RPipe. If the value in this field is zero then the Set RPipe Descriptor request can be used to set the number of blocks to be assigned to this RPipe. If the value in this field is not zero then the number of blocks assigned to this RPipe cannot be changed by host software. This field may be modified by host software if this is set to zero.
8	*wMaxPacketSize*	2	Number	Maximum packet size that this RPipe will use to communicate with attached device.
10	*bHSHubAddress*	1	Number	Reserved and must be set to zero.
11	*bHSHubPort*	1	Number	This field specifies the device index where the device information buffer is present.
12	*bSpeed*	1	Number	The PhyRate at which to communicate with the endpoint targeted by RPipe. The value and the associated rate is given in Section 5.6.
13	*bDeviceAddress*	1	Number	Address of the attached device
14	*bEndpointAddress*	1	Number	Endpoint Address to be used with this RPipe. __Bit__ __Description__ 3:0 The endpoint number 6:4 Reserved; set to zero 7 Direction, ignored for control endpoints 0 = OUT endpoint 1 = IN endpoint

Table 8-49. Wire Adapter RPipe Descriptor (cont.)

Offset	*Field*	Size	Value	Description
15	*bDataSequence*	1	Number	Current data sequence. This is the next data sequence value to be used when sending data to the endpoint that this RPipe is targeted at.
16	*dwCurrentWindow*	4	Number	Current Window used for data sequence management and burst transfers.
20	*bMaxDataSequence*	1	Number	Maximum sequence number that the endpoint supports. Valid values are 1 through 31.
21	*bInterval*	1	Number	For Interrupt transfers this is the polling interval to be used by this RPipe in downstream communications For Isochronous transfers this is the logical service interval. See *bInterval* in Table 7-31
22	*bOverTheAirInterval*	1	Number	If the transfer type is Isochronous, then this field is the interval for polling the downstream endpoint. This field is Reserved and must be set to zero for all other transfer types. See *bOverTheAirInterval* in Table 7-32 for the encoding of this field.
23	*bmAttribute*	1	Bitmap	**Bit** **Description** 1:0 Value Transfer Type 00B Control 01B Isochronous 10B Bulk 11B Interrupt 4:2 Transmit Power See Section 5.2.1.2 for details on the use of this field. 7:5 Data Burst Preamble Policy See Table 5-7 for the encoding of this field.

Table 8-49. Wire Adapter RPipe Descriptor (cont.)

Offset	*Field*	Size	Value	Description
24	*bmCharacteristics*	1	Bitmap	Transfer types supported on this RPipe **<u>Bit</u>** **<u>Description</u>** 0: 1: Control Transfer supported; 0: Control Transfer NOT supported 1: 1: Isochronous Transfer supported; 0: Isochronous Transfer NOT supported 2: 1: Bulk Transfer supported; 0: Bulk Transfer NOT supported 3: 1: Interrupt Transfer supported; 0: Interrupt Transfer NOT supported 7:4: Reserved This is a read only field.
25	*bmRetryOptions*	1	Bitmap	**<u>Bit</u>** **<u>Description</u>** 3:0: **Max Retry Count.** The maximum number of times a transaction must be retried before the transfer request is failed. The valid values are 0 through 15. A value of zero in this field indicates that the HWA must not count errors and there is no limit on the retries. For Isochronous transfers this field is set to zero. 6:4: **Reserved** 7: **Low Power Interrupt.** If this bit is set, then this is a low power interrupt endpoint and the MaxRetryCount field is ignored.
26	*wNumTransactionErrors*	2	Number	The HWA increments this field when it encounters an error while performing transactions to the downstream endpoint targeted by this RPipe. The host is responsible for resetting this field.

All fields that are not marked read only may be changed by host software by using the Set RPipe Descriptor request.

8.5.3 HWA Specific Requests

This section describes all the HWA specific requests.

Table 8-50. HWA Specific Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
Add MMC IE	00100001B	ADD_MMC_IE	Interval and Repeat Count	IE Handle and Interface Number	IE Length	IE Block
Get BPST Adjustment	10100001B	GET_TIME	TIME_ADJ	Interface Number	1	Adjustment Value
Get BPST Time	10100001B	GET_TIME	TIME_BPST	Interface Number	3	WUSB Channel Time
Get WUSB Time	10100001B	GET_TIME	TIME_WUSB	Interface Number	3	WUSB Channel Time
Remove MMC IE	00100001B	REMOVE_MMC_IE	Zero	IE Handle and Interface Number	Zero	None
Set Device Encryption	00100001B	SET_ENCRYPTION	Encryption Value	Device Index and Interface Number	Zero	None
Set Device Info	00100001B	SET_DEVICE_INFO	Zero	Device Index and Interface Number	36	Device Information Buffer
Set Device Key	00100001B	SET_DESCRIPTOR	Descriptor Type and Key Index	Device Index and Interface Number	Key Descriptor Length	Key Descriptor
Set Group Key	00100001B	SET_DESCRIPTOR	Descriptor Type and Key Index	Interface Number	Key Descriptor Length	Key Descriptor
Set Num DNTS Slots	00100001B	SET_NUM_DNTS	Interval And Number of DNTS Slots	Interface Number	Zero	None
Set WUSB Cluster ID	00100001B	SET_CLUSTER_ID	Cluster ID	Interface Number	Zero	None
Set WUSB MAS	00100001B	SET_WUSB_MAS	Zero	Interface Number	32	WUSB MAS
Set WUSB Stream Index	00100001B	SET_STREAM_IDX	Stream Index	Interface Number	Zero	None
WUSB Channel Stop	00100001B	WUSB_CH_STOP	WUSB Channel Time Offset	Interface Number	Zero	None

Table 8-51. HWA Specific Request Codes

bRequest	Value
ADD_MMC_IE	20
REMOVE_MMC_IE	21
SET_NUM_DNTS	22
SET_CLUSTER_ID	23
SET_DEVICE_INFO	24
GET_TIME	25
SET_STREAM_IDX	26
SET_WUSB_MAS	27
WUSB_CH_STOP	28

Table 8-52. WUSB Channel Time Type

Channel Time Types	Value
TIME_ADJ	0
TIME_BPST	1
TIME_WUSB	2

8.5.3.1 Add MMC IE

This request is used to add an Information Element to subsequent MMCs.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	ADD_MMC_IE	Interval and Repeat Count	IE Handle and Interface Number	IE Length	IE Block

Upon receipt of this request, the HWA will store the Information Element block of data internally. The upper byte of the *wValue* field (Interval) specifies the rate at which the HWA must include this Information Element block in an MMC. This field is expressed in milliseconds. The lower byte of *wValue* (Repeat Count) specifies the number of consecutive MMCs that this Information Element must be sent in during each interval. The WA operational requirements when it gets this request are given in Table 8-53.

Table 8-53. WA Add MMC IE Request Operational Requirements

Interval	Repeat Count	Operational Requirement
0	Number	The HWA must send this IE block in every MMC. The value in the Repeat Count field must be ignored.
1-254	Number	The HWA must send this IE block every *Interval* period. The *Repeat Count* field specifies the number of consecutive MMCs that this IE block must be sent in per Interval.
255	Number	This value must be treated as an infinite period. The HWA must only send this IE block the number of times specified in the *Repeat Count* field.

The upper byte of *wIndex* field uniquely identifies an Information Element block. This is the handle to be used by the host software when it needs to modify or remove this IE block from subsequent MMCs. The total number of IE blocks supported by the host controller is specified by *bNumMMCIEs* specified in the Wire Adapter descriptor.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wIndex* specifies an IE Handle that is larger than *bNumMMCIEs.*

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.2 Get BPST Adjustment

This request returns the current adjustment value of the BPST.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_TIME	TIME_ADJ	Interface Number	1	Adjustment Value

The lower byte of *wIndex* specifies the target interface number. The returned value gives the adjustment value in microseconds.

It is a Request Error if *wValue* or *wLength* are other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.3 Get BPST Time

This request returns the WUSB channel time at the BPST of the next super frame.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_TIME	TIME_BPST	Interface Number	3	WUSB Channel Time

The lower byte of *wIndex* specifies the target interface number. The returned value gives the 24-bit WUSB channel time value at the Beacon Period Start Time (BPST) of the current MBOA MAC superframe.

It is a Request Error if *wValue* or *wLength* are other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.4 Get WUSB Time

This request returns the current WUSB channel time.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_TIME	TIME_WUSB	Interface Number	3	WUSB Channel Time

The lower byte of *wIndex* specifies the target interface number. The returned value gives the current 24-bit WUSB channel time value.

It is a Request Error if *wValue* or *wLength* are other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.5 Remove MMC IE

This request is used to remove an Information Element from subsequent MMCs.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	REMOVE_MMC_IE	Zero	IE Handle and Interface Number	Zero	None

Upon receipt of this request, the HWA will erase the Information Element block of data that is uniquely identified by the IE Handle. The HWA must stop sending this Information Element block in subsequent MMCs. If the IE Handle does not exist, then the HWA is still required to complete the request successfully; however it must not remove any Information Element blocks that were added by any previous Add MMC IE commands.

The upper byte of the *wIndex* field (IE Handle) uniquely identifies an Information Element block. The total number of IE blocks supported by the HWA is specified by *bNumMMCIEs* specified in the Wire Adapter descriptor.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wValue* or *wLength* are other than as specified above or if the upper byte of *wIndex* species an IE Handle that is larger than *bNumMMCIEs.*

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.6 Set Device Encryption

This request sets the encryption type to be used when sending/receiving data to/from the device connected downstream of the HWA.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ENCRYPTION	Encryption Value	Device Index and Interface Number	Zero	None

Encryption Value comes from one of the Encryption Type descriptors contained in the Security Descriptor returned in the configuration descriptor. A value of Zero always represents UNSECURE or no encryption. The upper byte of *wIndex* (Device Index) specifies the Device Index

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if Encryption Value does not represent a valid encryption type.

It is a Request Error to attempt to set WIRED as the current encryption type.

It is a Request Error if *wLength* is other than as specified above or if the Device Index greater than or equal to the bNumPorts field in the Wire Adapter descriptor.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.7 Set Device Info

This request sets the device information buffer that is associated with the device connected downstream of the HWA.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_DEVICE_INFO	Zero	Device Index and Interface Number	36	Device Information Buffer

On reception of this request, the HWA will store the device information buffer for the downstream connected device. The format of the device information buffer is given below. The number of devices that an HWA can support at the same time is specified by the bNumPorts field in the Wire Adapter descriptor. The upper byte of *wIndex* (Device Index) can be any value between 0 and bNumPorts – 1.

The lower byte of *wIndex* specifies the target interface number.

Table 8-54. Device Information Buffer Format

Offset	Field	Size	Value	Description
0	*bmDeviceAvailabilityInfo*	32	Bitmap	This bitmap specifies the MAS slots in which the WUSB device can communicate with the host.
32	*bDeviceAddress*	1	Number	Address of the attached device
33	*wPHYRates*	2	Bitmap	Describes the PHY-level signaling rates capabilities of this device implementation represented as a bit-mask. See Section 7.4.1.1
35	*bmDeviceAttribute*	1	Bitmap	**Bit** **Description** 6:0 **Reserved** 7 Disable. **If this bit is set to a 1B, then the HWA must not perform any transactions to this device.**

It is a Request Error if *wLength* is other than as specified above or if the Device Index is greater than or equal to the bNumPorts field in the Wire Adapter descriptor.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.8 Set Device Key

This request sets the key to be used to encrypt/decrypt data when the HWA is sending/receiving data to/from the device connected downstream of the HWA.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_DESCRIPTOR	Descriptor Type and Key Index	Device Index and Interface Number	Key Descriptor Length	Key Descriptor

When the HWA receives this command, it uses the key data in the accompanying key descriptor to update its copy of the key to be used when sending/receiving data from this device. Host Wire Adapters are only required to support one key per device.

The upper byte of the *wIndex* field specifies the Device index. The Device Index must be less than the bNumPorts field in the Wire Adapter descriptor.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wValue* is other than as specified above or if the Device index greater than or equal to the bNumPorts field in the Wire Adapter descriptor.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.9 Set Group Key

This request sets the Group key to be used to encrypt data when the HWA is sending data to the WUSB cluster.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_DESCRIPTOR	Descriptor Type and Key Index	Interface Number	Key Descriptor Length	Key Descriptor

When the HWA receives this command, it uses the key data in the accompanying key descriptor to update its copy of the Group key to be used when sending data to the WUSB cluster.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.10 Set Num DNTS Slots

This request sets the interval and raw number of notification message time slots available in the DNTS.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_NUM_DNTS	Interval And Number of DNTS Slots	Interface Number	Zero	None

Upon receipt of this request, the HWA will schedule DNTS time slots in subsequent transaction groups. The upper byte of the *wValue* field (Interval) specifies the rate at which the HWA must schedule a DNTS time slot. If the Interval value is set to zero, then the HWA must schedule a DNTS in every transaction group. This field is expressed in milliseconds.

The lower byte of the *wValue* field specifies the number of slots that must be available.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.11 Set WUSB Cluster ID

This request sets the WUSB Cluster ID.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CLUSTER_ID	Cluster ID	Interface Number	Zero	None

This request sets the WUSB Cluster ID for this HWA. The *wValue* field specifies the Cluster Id.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.12 Set WUSB MAS

This request sets the MAS that the HWA can perform transaction in.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_WUSM_MAS	Zero	Interface Number	32	WUSB MAS

This request is used to set/update the currently available Media Access Slots that an HWA can use. WUSB MAS is an array of 256 entries, each of which corresponds to one of the 256 MAS within a superframe. The zero entries identify MAS that cannot be used by the HWA while nonzero entries identify MAS in which an HWA may perform WUSB transactions.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.13 Set WUSB Stream Index

This request sets the WUSB Stream Index.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_STREAM_IDX	Stream Index	Interface Number	Zero	None

This request sets the WUSB Stream Index for this HWA. The *wValue* field specifies the Stream Index.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.3.14 WUSB Channel Stop

This request is used to stop the WUSB channel.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	WUSB_CH_STOP	WUSB Channel Time Offset	Interface Number	Zero	None

On reception of this request, the HWA must stop the Wireless USB channel as described in Section 4.16.2.1 of the Data Flow chapter. The *wValue* field (WUSB Channel Time Offset) specifies the offset in microseconds from the current time when the HWA must stop the channel. If remote wake is enabled on the HWA, then it must support the remote wakeup mechanisms specified in Section 4.16.2.2. The HWA must transition into the **Disabled** state after the Wireless USB channel is stopped.

If WUSB Channel Time Offset is zero, then the HWA must cancel the Wireless USB Channel Stop operation.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wLength* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the Host Wire Adapter is not configured, the Host Wire Adapter's response to this request is undefined.

8.5.4 HWA Notification Information

Host Wire Adapters must send back other notifications for BPST Adjustment Change and any device notification received from a Wireless USB device. The format of each notification is detailed below.

8.5.4.1 BPST Adjustment Change

If the adjustment value to the BPST has changed from the previous superframe to the current superframe then the wire adapter must send a BPST Adjustment Change notification to the host. The format of this notification is shown in Table 8-55

Table 8-55. BPST Adjustment Change Notification

Offset	Field	Size	Value	Description
0	*bLength*	1	3	Length of this block of data
1	*bNotifyType*	1	94H	NOTIFY_TYPE_BPST_ADJ
2	*bAdjustment*	1	Number	New adjustment value in microseconds

8.5.4.2 DN Received Notification

When a host wire adapter receives a device notification from a Wireless USB device it must send that notification to host software. The notification must be sent to host software as shown in Table 8-56.

Table 8-56. DN Received Notification

Offset	Field	Size	Value	Description
0	*bLength*	1	Number	Length of this block of data
1	*bNotifyType*	1	95H	NOTIFY_TYPE_DN_RECEIVED
2	*bmAttributes*	1	Bitmap	<u>Bit</u> <u>Description</u> 6:0 Reserved 7 This bit is set if this notification was received as a secure frame.
3	*NotificationSpecific*	Variable	Raw Data	The device notification received. The HWA is responsible for decrypting the notification if it was a secure frame. See Section 7.6 for the various notifications that an HWA may receive.

The only notification that an HWA must locally process is a DN_EPRdy notification.

NOTE: An HWA must send back a DN_Alive notification (see Section 7.6.7) each time it receives a NAK handshake from a low power interrupt in endpoint.

8.5.5 HWA Isochronous Transfers

To start an isochronous transfer to a Wireless USB device connected downstream of a Host Wire Adapter, host software uses the same basic transfer request mechanism described for bulk, control and interrupt requests. The only additional information sent is the isochronous packet Information. Isochronous transfer requests to an HWA use the Isochronous Transfer Request as shown in Table 8-57. This request type allows large transfers to be segmented into multiple smaller transfers to avoid RPipe buffer overflow on the Host Wire Adapter. The host will send the number of service intervals that this transfer request describes in *dwNumOfPackets* field. The host will send the amount of data to be transferred in each service interval in a Packet Length array immediately after the Transfer Request. The format of this Isochronous Packet Information is shown in Table 8-58.

Further, in the case of OUT transfers, the request, packet information and the data are sent as consecutive transactions. This allows the Host Wire Adapter to receive and interpret the request first and prepare for data allocation.

Table 8-57. Isochronous Transfer Request

Offset	Field	Size	Value	Description
0	*bLength*	1	14H	Length of this request
1	*bRequestType*	1	82H	REQUEST_TYPE_ISOCHRONOUS – indicates a Isochronous transfer
2	*wRPipe*	2	Number	RPipe this transfer is targeted to
4	*dwTransferID*	4	Number	Host-assigned ID for this transfer
8	*dwTransferLength*	4	Number	Amount of data following for an OUT transfer or the maximum amount of returned data for an IN transfer
12	*bTransferSegment*	1	Bitmap	<u>Bit</u> <u>Description</u> 6:0 Segment Number 7 Last Segment
13	*bReserved*	1	Number	Reserved
14	*wPresentationTime*	2	Number	For OUT transfers, this is the *wPresentationTime* of the first Wireless USB packet sent by the HWA. See Table 5-2 For IN transfers, this is the Wireless USB Channel time by which the transfer must be completed.
16	*dwNumOfPackets*	4	Number	Number of Packet Lengths following

Table 8-58. Isochronous Packet Information

Offset	Field	Size	Value	Description
0	*wLength*	2	Number	Length of this block of data
1	*bPacketType*	1	A0H	ISO_PACKET_INFORMATION_TYPE
2	*bReserved*	1	Zero	Reserved for future use, must be zero.
4	*PacketLength[n]*	Variable	Array	The length of data to be sent/received in each service interval. Each array element is a word in size.

A Host Wire Adapter uses the isochronous data header format as described in Section 5.1 in the protocol chapter to communicate with Wireless USB isochronous endpoints.

Note: A DWA uses native isochronous wireless endpoints to support downstream wired isochronous endpoints. Therefore, the HWA performs no special handling for a DWA that supports an isochronous endpoint.

8.5.5.1 HWA Isochronous OUT Responsibilities

An HWA must aggregate HWA Isochronous Packet data into the largest packets that can be sent to the Wireless USB isochronous endpoint. It must not split data from a single HWA Isochronous Packet across multiple over-the-air packets.

Presentation times for over-the-air isochronous packets are calculated based on the *wPresentationTime* value in the transfer request and the *bInterval* value in the RPipe descriptor. The *bOverTheAirInterval* specified in the RPipe descriptor determines the rate at which the Wireless USB Isochronous endpoint must be serviced.

Figure 8-10 illustrates a Wireless USB Isochronous OUT data stream through an HWA. The illustration is organized with time flowing from left to right and data flow from top to bottom, where the top illustrates a Isochronous transfer request to the HWA Data Transfer Write endpoint, down through the HWA RPipe buffer and finally over the Wireless USB channel to the recipient endpoint.

The host sends an Isochronous Transfer Request (containing the Wireless USB presentation time), the Packet Information and the data destined for the Wireless USB endpoint. Once it has all the data for this transfer request, the HWA will start sending the data to the downstream endpoint. The format of the Wireless USB packet is the standard Wireless USB isochronous data format. In this example, the rate at which the Wireless USB endpoint is serviced is set to 4.096 ms and the interval between the segments in that packet is set to 1ms. The Wireless USB endpoint has a *wMaxPacketSize* of 1000 bytes (corresponds to the Packet length in the Packet Information) with *wOverTheAirPacketSize* set to 2048 (*wMaxPacketSize* in the RPipe Descriptor). The Isochronous Transfer Request describes a buffer with 32ms worth of data.

In this example, the HWA starts sending the data to the Wireless USB packets before the presentation for that data. It sends at least 2 Wireless USB packets every 4.096ms. Each packet contains 2 segments of a 1000 each as per the packetization information present in the Isochronous Transfer Request. The HWA uses the *wPresentationTime* in the Isochronous Transfer Request as the *wPresentationTime* in the first Wireless USB Packet. The HWA uses the *bInterval* field in the RPipe descriptor to calculate the *wPresentationTime* in subsequent Wireless USB Packets sent to the endpoint. Once the HWA has sent all the data to the Wireless USB endpoint it sends a Transfer Result along with Packet Status information back to the host on the Data Transfer Read endpoint.

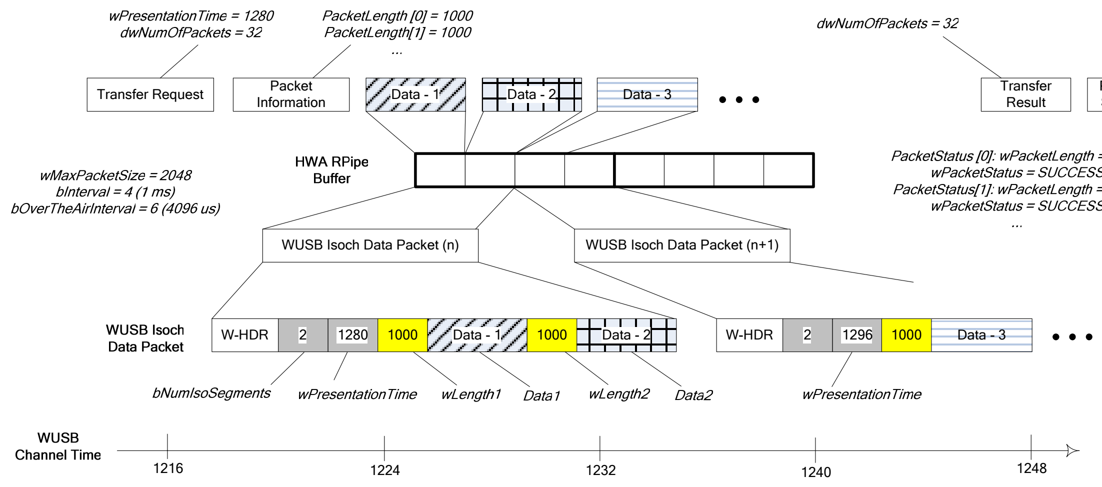

Figure 8-10. Wireless USB Isochronous OUT Data Stream through an HWA

The HWA is responsible for discarding packets as described in Section 4.11.9 if the current Wireless USB Channel time has exceeded the presentation time of a packet that the HWA has been unable to transmit. The HWA must not attempt to transmit a packet whose presentation time has expired.

The HWA must set the HWA Isochronous Packet Status for each HWA Isochronous Packet.

8.5.5.2 HWA Isochronous IN Responsibilities

The HWA is required to process received wireless isochronous data packets and return the information over the wired interface using the transfer result and HWA isochronous packet status format. The HWA parses the Wireless USB isochronous packet header information and places the data only in the data buffer that will be returned to the host software.

The HWA must start performing IN transaction to the downstream Wireless USB endpoint as soon as possible after receiving the Isochronous Transfer request. The first Wireless USB packet is placed in the first location in the RPipe buffer. The presentation time in this Wireless USB Packet along with the *wPresentationTime* in the subsequent Wireless USB packets and the *bInterval* in the RPipe descriptor is used to determine the location in the RPipe buffer for the data in those Wireless USB packets. The HWA must update the Packet Status length information for each Wireless USB packet received based on the segment length information present in the Wireless USB Isochronous header. For a complete transfer, the *dwNumOfPackets* field in the Isochronous Transfer request must be equal to the total number of segments in the Wireless USB packets received from the Wireless USB endpoint.

Once the HWA has received all the data from the Wireless USB endpoint or the Wireless USB channel time exceeds the *wPresentationTime* specified in the Isochronous Transfer request, it must retire the Isochronous Transfer request and send a Transfer Result along with the Packet Status information back to the host on the Data Transfer Read endpoint.

8.5.5.3 HWA Isochronous Transfer Completion

The data and result of an isochronous transfer to a Wireless USB device connected downstream of a Host Wire Adapter are returned to host software on the Data transfer read endpoint. On completion of an Isochronous transfer on an HWA, the HWA will send a transfer completion notification on its notification endpoint and transfer result will be available to host software on its Data Transfer Read endpoint. The Transfer Result will contain the number of Packet Status records to be expected immediately after the Transfer Result. The format of the Isochronous Packet Status is shown in Table 8-59. If this was an IN transfer request, then the data read from the device will be sent after the packet status information.

Table 8-59. HWA Isochronous Packet Status

Offset	Field	Size	Value	Description
0	*wLength*	2	Number	Length of this block of data
1	*bPacketType*	1	A1H	ISO_PACKET_STATUS_TYPE
2	*bReserved*	1	Zero	Reserved for future use, must be zero.
4	*PacketStatus[n]*	Variable	Array	This is an array of Packet Length and Packet Status tuples. wPacketLength — Actual length of data sent or received in the frame wPacketStatus — Status of this Packet

8.6 Radio Control Interface

An HWA must expose a Radio Control interface so that host software can control the radio. This section describes the Radio Control interface required to control the UWB Radio. This interface consists of one interrupt endpoint. This endpoint along with the default control endpoint of the device is used to control the UWB Radio in the Device.

Control Endpoint	This is the default USB control endpoint. All radio control commands are sent to the device through this endpoint.
Radio Control Interrupt Endpoint	This Interrupt IN endpoint is used to return status and results of the radio control commands sent on the default control endpoint. Asynchronous UWB Radio notifications are also sent back to the host

software via this endpoint.

The device must always send a short packet to terminate transfers on this endpoint.

The Radio Control interface allows the host software to control and configure the UWB Radio by using several standard commands. The UWB Radio must be configured before Wireless USB data and notifications can be exchanged. For example, the minimal setup required for a host to configure the radio to be ready for Wireless USB data and notifications is:

- The host initiates scanning on a UWB channel using the Scan command (see Section 8.6.2.5) with the scan state set to SCAN_ONLY.
- During the scan if the device receives one or more beacons from other devices on the same channel and within radio range, a Beacon Received notification (see Section 8.6.3.2) is generated for every received beacon.
- The scan will continue until the host uses the Scan command (see Section 8.6.2.5) with the scan state set to SCAN_DISABLED which will disable the scan. The host may scan additional channels by repeating above steps again. After scanning on one or more channels the host will determine which channel it would like to use for the UWB Radio communication.
- The host will get the Capabilities IE from the device by sending it a Get IE command (see Section 8.6.2.3). The host can modify any of the optional capabilities reported by the device and set any other IEs that it wants the device to send in the beacon using the Set IE command (see Section 8.6.2.8).
- Next the host will instruct the UWB Radio to start sending beacons by using the Start Beaconing command (see Section 8.6.2.12).
- The host will then create a reservation consisting of one or more MASs by using the Set DRP IE command (see Section 8.6.2.7). The UWB radio is now configured and is ready for WUSB initialization and communication.

8.6.1 Radio Control Descriptors

A device must expose the following interface so that host software can properly control the UWB Radio in that device.

8.6.1.1 Radio Control Interface Descriptor

Table 8-60. Radio Control Interface Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	9	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	4	INTERFACE Descriptor Type
2	*bInterfaceNumber*	1	Number	Number of this interface
3	*bAlternateSetting*	1	0	Value used to select this alternate setting for the interface identified in the prior field
4	*bNumEndpoints*	1	1	Number of endpoints used by this interface.
5	*bInterfaceClass*	1	E0H	Wireless Controller
6	*bInterfaceSubclass*	1	01H	RF Controller
7	*bInterfaceProtocol*	1	02H	UWB Radio Control Interface
8	*iInterface*	1	Index	Index of String Descriptor describing this interface

8.6.1.2 Radio Control Interface Class Descriptor

This descriptor describes the characteristics of the Radio Control Interface to host software.

Table 8-61. Radio Control Interface Class Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	4	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	23H	Radio Control Descriptor Type
2	*bcdRCIVersion*	2	0100H	Radio Control Interface Version number in Binary-Coded Decimal.

8.6.1.3 Radio Control Interrupt Endpoint Descriptor

This endpoint is used to report status and results of the radio control commands. It is also used to send UWB Radio notifications back to the host software.

Table 8-62. Radio Control Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	*bLength*	1	7	Size of this descriptor in bytes, including this field.
1	*bDescriptorType*	1	5	ENDPOINT Descriptor Type
2	*bEndpointAddress*	1	Number	The address of this endpoint
3	*bmAttributes*	1	Bitmap	Interrupt endpoint of 00000011b.
4	*wMaxPacketSize*	2	200H	Maximum packet size this endpoint
6	*bInterval*	1	1	Interval for polling endpoint for data transfers. Expressed in frames or microframes depending on the device operating speed (i.e., either 1 millisecond or 125 µs units).

8.6.2 Radio Control Command

To control the UWB Radio only one control transfer command is defined. All UWB Radio Control commands are encapsulated within a Command Block defined in Table 8-65 and sent using this control transfer request. The result of the command is sent back on the Radio Control Interrupt endpoint using the Event Block defined in Table 8-66.

Table 8-63. Execute Radio Control Command

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	EXEC_RC_CMD	Zero	Interface Number	Command Length	Radio Control Command Block

Table 8-64. Radio Control Request Codes

bRequest	Value
EXEC_RC_CMD	40

Table 8-65. Radio Control Command Block (RCCB)

Offset	Field	Size	Value	Description
0	*bCommandType*	1	Number	The type of Command.
1	*wCommand*	2	Number	The actual command to be performed
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH. A value of FFH in this field indicates that this is a RESPONSE to an earlier notification. A value of 00H in this field is invalid.
4	*Parameter0*	Var0	Number	First parameter for this command. The size and value of this parameter is specific to the actual command,
4 + Var0	*Parameter1*	Var1	Number	Second parameter for this command. The size and value of this parameter is specific to the actual command,
…				
Var	*ParameterN*	VarN	Number	Last parameter for this command. The size and value of this parameter is specific to the actual command,

Table 8-66. Radio Control Event Block (RCEB)

Offset	Field	Size	Value	Description
0	*bEventType*	1	Number	The type of Event
1	*wEvent*	2	Number	The event that occurred. If this event was a result of a host issued command then this should match the *wCommand* in the RCCB.
3	*bEventContext*	1	Number	If this event was a result of a host issued command then this should match the *bCommandContext* in the RCCB. A value of Zero indicates an Event that occurred which is not a direct result of a Radio Control Command. A value of FFH in this field is invalid.
4	*Parameter0*	Var0	Number	First parameter for this event. The size and value of this parameter is specific to the actual event
4 + Var	*Parameter1*	Var1	Number	Second parameter for this event. The size and value of this parameter is specific to the actual event
…				
Var	*ParameterN*	VarN	Number	Last parameter for this event. The size and value of this parameter is specific to the actual event

Table 8-67. Command or Event Type

Command/Event Type	Value
GENERAL	0
RESERVED	01-EFH
VENDOR_SPECIFIC	F0-FFH

Table 8-68. Command or Event

	Command/Event	Value
NOTIFICATIONS	AS_PROBE_IE_RECEIVED	0
	BEACON_RECEIVED	1
	BEACON_SIZE_CHANGE	2
	BPOIE_CHANGE	3
	BP_SLOT_CHANGE	4
	BP_SWITCH_IE_RECEIVED	5
	DEV_ADDR_CONFLICT	6
	DRP_AVAILABITY_CHANGE	7
	DRP	8
	Reserved	9-15
COMMANDS	CHANNEL_CHANGE	16
	DEV_ADDR	17
	GET_IE	18
	RESET	19
	SCAN	20
	SET_BEACON_FILTER	21
	SET_DRP_IE	22
	SET_IE	23
	SET_NOTIFICATION_FILTER	24
	SET_TX_POWER	25
	SLEEP	26
	START_BEACONING	27
	STOP_BEACONING	28

Table 8-69. Result Codes

Result Codes	Value
SUCCESS	0
FAILURE	1
FAILURE_HARDWARE	2
FAILURE_NO_SLOTS	3
FAILURE_BEACON_TOO_LARGE	4
FAILURE_INVALID_PARAMETER	5
FAILURE_UNSUPPORTED_PWR_LEVEL	6
TIME_OUT	7

This request is used to control the UWB Radio MAC in a device.

Upon receipt of this request, the device will perform the command indicated in the RCCB. The format of the RCCB is given in Table 8-65.

The lower byte of *wIndex* specifies the target interface number.

It is a Request Error if *wValue* is other than as specified above.

If the interface specified does not exist, then the device responds with a Request Error.

If the device is not configured, the device's response to this request is undefined.

Each of the commands and their parameters are detailed below.

8.6.2.1 Channel Change

This command is used to inform other devices that this device is going to change the channel. The device must send a Channel Change IE in its beacons for the number of times specified in the command.

The RCCB for this command is given below.

Table 8-70. Channel Change RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	16	CHANNEL_CHANGE Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bChannelChange Countdown*	1	Number	Number of superframes before the device changes to the new channel.
5	*bNewChannel Number*	1	Number	New channel number to which the device moves. The encoding of the channel number is specified in Table 5-12.

The *bChannelChangeCountdown* indicates the number of superframes until the device actually changes to the new channel. The device must modify the DRP Availability it sends in its beacon along with the Channel Change IE. The device must send the IE in the beacon until it completes the channel change operation. The device sets the *Channel Change Countdown* field in the Channel Change IE to the specified value in the first beacon, and it must decrement the countdown value by one in each subsequent superframe.

The *bNewChannelNumber* specifies the channel number of the new channel and the device sets the *New Channel Number* field in the Channel Change IE to this value.

The device will confirm that it has sent the last beacon with its *Channel Change Countdown* value set to zero successfully on the current channel or that the operation has failed by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

Table 8-71. Channel Change RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	16	Result of CHANNEL_CHANGE Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the channel change operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure. The STC (see Section 8.6.2.4) will revert back to a free running timer until the device starts beaconing again.

8.6.2.2 Device Address Management

This command is used to query or set the 16-bit device address or the 64-bit MAC address (EUI-64) that are used by the device.

The RCCB for this command is given below.

Table 8-72. Device Address Management RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	17	DEV_ADDR Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bmOperationType*	1	Bitmap	Specifies the type of the device address management operation. **Bit** **Description** 0 Set. If this bit is one, this command sets the new address to be used by the device. If this bit is zero, this command gets the current address used by the device. 7:1 Address type **Value** **Meaning** 0 16-bit device address 1 64-bit MAC address (EUI-64) 2-127 Reserved
5	*BaAddr*	8	Byte array	This field contains a new address if the *Set* bit is one, otherwise it will be ignored.

The type of operation is indicated by the *bmOperationType*. When the address type is zero and the *Set* bit is zero, the command gets the current 16-bit device address used by the device. The device is not required to generate a device address and this command just returns the address which was previously set by the host.

When the address type is zero and the *Set* bit is one, the command sets a new 16-bit device address to the device. The host software is responsible for generating a device address. The device must use the specified address as the source address of all the outgoing MAC frames including beacons in the subsequent superframes.

When the address type is one and the *Set* bit is zero, the command gets the current 64-bit MAC address (EUI-64) used by the device. The address is either statically preconfigured or temporally assigned by the host.

When the address type is one and the *Set* bit is one, the command sets a new 64-bit MAC address (EUI-64) to the device. The device will use this address as the *Device Identifier* field in the *Beacon Parameters* of its beacons in the subsequent superframes. The host is also required to set the 16 bit Device Address if it changes the 64-bit MAC address.

The *baAddr* field contains the address information that is set to the device in the case of set operation, i.e. the *Set* bit is one. The size of the address information depends on the address type. If the address type is zero (16-bit device address) then the device address will be stored in the first 2 bytes of the *baAddr* field.

The RCEB for this command is given below.

Table 8-73. Device Address Management RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	17	Result of DEV_ADDR Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*baAddr*	8	Byte array	This field contains the returned address information if the *Direction* bit in the associated RCCB is one, otherwise it will be ignored.
12	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the device address management operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

The address information is returned to the host in the *baAddr* field in the case of query operation, i.e. the *Set* bit in the previous RCCB is set to a zero. The size of the address information depends on the address type specified in the associated RCCB. If the address type is zero (16-bit device address) then the device address will be stored in the first 2 bytes of the *baAddr* field.

8.6.2.3 Get IE

If the device is beaconing then the device must return all the IEs that are being transmitted in the beacon by the device.

If the device is not beaconing then it must return the IEs that have been set by the host by a previous Set IE command. If the host has not set any IEs then the device must return the local device Capabilities IE.

The RCCB for this command is given Table 8-74.

Table 8-74. Get IE RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	18	GET_IE Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.

The device will respond with the result of the operation to get the IEs by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

Table 8-75. Get IE RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	18	Result of GET_IE Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*wIELength*	2	Number	The length of the IE data to be returned to the host.
6	*IEData*	Var	Raw Data	A variable size array containing IE data.

8.6.2.4 Reset

This command instructs the device to reset the UWB Radio to the default power on state.

The RCCB for this command is given below.

Table 8-76. Reset RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	19	RESET Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.

All pending operations are aborted and all UWB Radio parameters and buffers return to the default power on state. Upon receipt, the device must reset all UWB Radio Control variables and other settings to their initial values. In particular, the device must reset its internal Superframe Time Counter (STC) which is a simple 16-bit counter used to indicate the relative offset in the current superframe from the Beacon Period Start Time (BPST), to zero and must be a value between 0 and 65535. The STC is a free running timer until the device starts beaconing. This timer value is used as the wBPSTOffset in any beacon received notification sent to the host.

The device will confirm the result of the reset operation started by the previous command by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given Table 8-77.

Table 8-77. Reset RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	19	Result of RESET Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the reset operation was successful. If a reset is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.5 Scan

This command instructs the device to start/stop a scan operation.

The RCCB for this command is given below.

Table 8-78. Scan RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	20	SCAN Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bChannelNumber*	1	Number	The physical channel to be scanned. The encoding of the channel number is specified in Table 5-12.
5	*bScanState*	1	Number	Specifies the type of scan. **Value** — **Description** 0 — SCAN_ONLY Scan only. No other transmit or receive operation is performed until the scan is disabled. 1 — SCAN_OUTSIDE_BP Scan at all times except during the beacon period. 2 — SCAN_WHILE_INACTIVE Scan only when not scheduled to transmit or receive. 3 — SCAN_DISABLED Scanning is disabled. This is the default scanning state on power up or after a RESET command completes successfully.

The scanning state is indicated by the *bScanState* field in the command. The *bChannelNumber* indicates the PHY channel on which the MAC should listen during the scan.

If a host sends a Scan command with *bScanState* set to SCAN_ONLY when the device is actively sending beacons then the device's response to this command is undefined.

If *bScanState* is set to SCAN_OUTSIDE_BP then the device will scan on the specified channel except during its beacon period. This command is invalid if the device is not currently beaconing.

If *bScanState* is set to SCAN_WHILE_INACTIVE then the device will scan on the specified channel when it is not actively transmitting or receiving. The device must not scan during its beacon period as well. This command is invalid if the device is not currently beaconing.

The device will confirm the completion, successfully or in error, of a scanning state change initiated by this command by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

The device will send back any beacons it receives during the scan using the Beacon Received notification, see Section 8.6.3.2. The beacon filter (see Section 8.6.2.6) is ignored while the device is Scanning.

Table 8-79. Scan RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	20	Result of SCAN Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the operation to change the scan state has successfully completed. If changing the scan state is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.6 Set Beacon Filter

This command is used to set the Beacon filter. This filter is based on the beacon slot number.

The RCCB for this command is given below.

Table 8-80. Set Beacon Filter RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	21	SET_BEACON_FILTER Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bmBeaconSlots*	6	Bitmap	48-bit bitmap specifying beacon slots that the host does not need beacon information on.
10	*bEnableState*	1	Number	Specifies whether to enable or disable the filter. Value — Description 0 — Filter DISABLED Filter is disabled. This is the default filter state on power up or after a RESET command completes successfully. 1 — Filter ENABLED

If the device detects a beacon in a slot whose bit is not set in *bmBeaconSlots* or the Device Address of the device sending the beacon in a slot changes from one superframe to the next then it must send a Beacon Received notification to the host.

If a beacon is not detected in a slot whose bit is set in *bmBeaconSlots* then the device must generate a BPOIE Change notification (See Section 8.6.3.3).

This filter information is only valid when the device is actively sending beacons and during the beacon group which the device has joined.

The RCEB for this command is given below.

Table 8-81. Set Beacon Filter RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	21	Result of SET_BEACON_FILTER Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the set beacon filter operation was successful. If the operation is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.7 Set DRP IE

This command is used to set a DRP IE being transmitted in the beacon by the device. The host can also remove all DRP IEs that are being transmitted by the device using this command. The host must set all the DRP IEs that it wants the device to include in its beacon at the same time e.g. if the host includes three DRP IEs in the first Set DRP IE and then includes two DRP IEs in the next Set DRP IE command then the device will send the DRP IEs specified in the second Set DRP IE command only.

The RCCB for this command is given below.

Table 8-82. Set DRP IE RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	22	SET_DRP_IE Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bExplicit*	1	Flag	Controls whether implicit or explicit DRP negotiation is used. When this flag is set, then Explicit DRP negotiation is used.
5	*wIELength*	2	Number	The length of the IE data to be added to the beacon.
7	*IEData*	Var	Raw Data	A variable size array containing IE data.

The *IEData* field in this command is the DRP IE data to be added to the beacon if the *bExplicit* flag is not set. The length of the *IEData* field is specified by *wIELength*. The device must replace the DRP IEs that it is currently sending with the DRP IEs specified in this command. If *wIELength* is set to zero, then the device will remove any existing DRP IEs from its beacon.

The *IEData* field in this command is the DRP IE data to be used in the DRP reservation command if the *bExplicit* flag is set. The length of the *IEData* field is specified by *wIELength*. The device must send a DRP reservation request command if the *Owner* bit in the DRP IE is set, e.g. this device is the reservation owner. On the other hand, the device must send a DRP reservation response if the *Owner* bit is not set by including the specified DRP IE data along with the current DRP Availability IE.

The device will confirm the result of the operation to set zero or more DRP IEs to the beacon or the status of sending an explicit DRP IE command by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given in Table 8-83.

If a host sends a Set DRP IE with the *bExplicit* flag set when the device does not support explicit DRP negotiations as reported in its Capabilities IE then the device's response to this command is undefined.

Table 8-83. Set DRP IE RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	22	Result of SET_DRP_IE Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*wRemainingSpace*	2	Number	Indicates the number of remaining bytes left in the beacon
6	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The number of remaining bytes left in the beacon is indicated by the value of *wRemainingSpace*. The *bResultCode* field indicates the result of the attempt to add a DRP IE to the beacon if *bExplicit* is not set or indicates if the device successfully sent an explicit DRP IE command if *bExplicit* is set. If an operation to add the DRP IE to the beacon is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.8 Set IE

This command is used to set one or more IEs in the beacon being transmitted by the device.

The RCCB for this command is given below.

Table 8-84. Set IE RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	23	SET_IE Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*wIELength*	2	Number	The length of the IE data to be added to the beacon.
6	*IEData*	Var	Raw Data	A variable size array containing IE data.

The *IEData* field in this command is the data to be added to the beacon. This may contain one or more IEs which are defined by the host. This data replaces any data from a previous Set IE command. The length of the *IEData* field is specified by *wIELength*.

The complete list of Information Elements available is listed in the MAC layer specification (see Reference [3]). The host can set the IEs listed in Table 8-85 using this command.

Table 8-85. Host Settable IEs

IE Name	Notes
BP Switch IE	Used by Wireless USB
Capabilities IE	
Identification IE	
PCA Availability IE	
Application-specific Probe IE	
Master Key Identifier (MKID) IE	
Application Specific IE (ASIE)	

The behavior of the device if other IEs are set using this command is undefined.

The device must not reorder the IEs set by this command. However it must correctly insert the other IEs it generates as well as the DRP IEs that may be set using the Set DRP IE (see Section 8.6.2.7) command in the set of IEs that it transmits in its beacon. The device will confirm the result of the operation to set zero or more IEs in the beacon by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

Table 8-86. Set IE RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	23	Result of SET_IE Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*wRemainingSpace*	2	Number	Indicates the number of remaining bytes left in the beacon
6	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The number of remaining bytes left in the beacon is indicated by the value of *wRemainingSpace*. The *bResultCode* field indicates the result of the attempt to add an IE to the beacon. If an operation to add an IE to the beacon is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.9 Set Notification Filter

This command instructs the device to filter one or more notifications. The RCCB for this command is given below.

Table 8-87. Set Notification Filter RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	24	SET_NOTIFICATION_FILTER Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*wNotification*	2	Bitmap	Each bit in this bitmap identifies the notification to be filtered. If a bit is set then that notification must not be sent to the host. **Bit** **Notification** 0 AS_PROBE_IE_RECEIVED 1 BEACON_RECEIVED 2 BEACON_SIZE_CHANGE 3 BPOIE_CHANGE 4 BP_SLOT_CHANGE 5 BP_SWITCH_IE_RECEIVED 6 DEV_ADDR_CONFLICT 7 DRP_AVAILABITY_CHANGE . 8 . DRP . 15:9 . Reserved
6	*bEnableState*	1	Number	Specifies whether to enable or disable the filter. **Value** **Description** 0 Filter DISABLED Filter is disabled. This is the default filter state on power up or after a RESET command completes successfully. 1 Filter ENABLED

The device will confirm the result of the set notification filter operation by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

Table 8-88. Set Notification Filter RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	24	Result of SET_NOTIFICATION_FILTER Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

8.6.2.10 Set TX Power

This command is used to set the default transmit power for transmissions by this device.

The RCCB for this command is given below.

Table 8-89. Set TX Power RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	25	SET_TX_POWER Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bPowerLevel*	1	Number	Value indicating the number of steps below the highest power level that must be used.

If the power level for transmission of a packet is not set by other means, then the device must use the power level specified in *bPowerLevel* to transmit that packet.

The default state on power up or after a RESET command completes successfully is the highest power level.

The RCEB for this command is given in Table 8-90.

Table 8-90. Set TX Power RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	25	Result of SET_TX_POWER Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

8.6.2.11 Sleep

On reception of this command the device will start hibernation after transmitting the *Hibernation Mode IE* in its beacons for the specified number of times if the device is not enabled for remote wake. If it is enabled for remote wake then it keeps sending beacons in every superframe. In the latter case, the device will wake up the host (if the host is sleeping) if it needs to send a notification that is not filtered.

The RCCB for this command is given below.

Table 8-91. Sleep RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	26	SLEEP Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*bHibernationCount*	1	Number	Number of superframes until the device begins hibernation. This field is only valid when remote wake is disabled on the device.
6	*bHibernationDuration*	1	Number	Number of superframes for which the device intends to hibernate. This field is only valid when remote wake is disabled on the device.

If the remote wake is disabled on the device, it will begin hibernation after transmitting a number of beacons including *Hibernation Mode IE*. The *Hibernation Countdown* field in this IE is set to the *bHibernationCount* for the first beacon and must be decremented by one for each superframe. After sending a beacon with its *Hibernation Countdown* set to a zero, the device stops sending beacons and begins hibernation. The *Hibernation Duration* field in this IE must be set to the value specified by the *bHibernationDuration* field. The device will stay in hibernation during the specified period, but it is not required to wake up by itself after the specified hibernation period. The host is responsible for waking up the device and requesting it to restart beaconing.

If remote wake is enabled on the device, it must keep sending beacons in every superframe. Before executing this command, the host must specify a set of IEs that the device must send. When the device detects a wake event, e.g. DRP conflict, and the event notification is required by the notification filter (see Section 8.6.2.9), it must wake up the host (if it's sleeping) and send the notification to the host.

The RCEB for this command is given in Table 8-92.

Table 8-92. Sleep RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	26	Result of SLEEP Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the sleep operation was successful. If a sleep is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.12 Start Beaconing

This command instructs the device to begin beaconing on the specified channel.

The RCCB for this command is given below.

Table 8-93. Start Beaconing RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	27	START_BEACONING Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FEH.
4	*wBPSTOffset*	2	Number	The offset of a received beacon relative to the BPST of the device measured in microseconds.
6	*bChannelNumber*	1	Number	The physical channel on which beaconing is to occur. The encoding of the channel number is specified in Table 5-12.

The *bChannelNumber* field value is the physical layer channel on which beaconing will begin. The device must join the beacon group that is identified by *wBPSTOffset.* The value in *wBPSTOffset* must be a value that was returned in a Beacon Received notification, see Section 8.6.3.1.

If the device is currently beaconing, then the device's response to this command is undefined.

If the host did not receive any Beacon Received notifications, then the host must set *wBPSTOffset* to zero and the device must start beaconing as soon as possible and establish a beacon group.

The device will confirm that the first beacon has been successfully transmitted or that the operation has failed by sending back an RCEB on the Radio Control Interrupt Endpoint. If the device successfully starts beaconing then it must listen to all the beacons during its beacon period and return any beacon received notifications that are not filtered to the host. In addition the Superframe Time Counter (STC), see Section 8.6.2.4 must be adjusted so that the zero of the STC corresponds to the new BPST.

The RCEB for this command is given in Table 8-94.

Table 8-94. Start Beaconing RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	27	Result of START_BEACONING Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether beacons have begun to be successfully transmitted. If an operation to start beaconing is not succeeding, it is a vendor specific decision when to time out the operation and return failure.

8.6.2.13 Stop Beaconing

This command instructs the device to stop beaconing.

The RCCB for this command is given below.

Table 8-95. Stop Beaconing RCCB

Offset	Field	Size	Value	Description
0	*bCommandType*	1	0	GENERAL Command Type
1	*wCommand*	2	28	STOP_BEACONING Command
3	*bCommandContext*	1	Number	Host assigned ID for this command. Valid values are 1 through FFH.

This command is sent by the host software to stop beaconing that was started with the Start Beaconing command.

The device will confirm that beaconing has been stopped by sending back an RCEB on the Radio Control Interrupt Endpoint. The RCEB for this command is given below.

Table 8-96. Stop Beaconing RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	28	Result of STOP_BEACONING Command
3	*bEventContext*	1	Number	This should match the *bCommandContext* in the RCCB.
4	*bResultCode*	1	Number	Indicates the completion status of the command. See Table 8-69 for a list of result codes.

The *bResultCode* field indicates whether the beaconing operation successfully stopped. A beacon period must have passed without a beacon being transmitted for the confirmation to complete successfully. If ending beaconing is not succeeding, it is a vendor specific decision when to time out the operation and return failure. The STC will revert back to a free running timer until the device starts beaconing again.

8.6.3 Radio Control Notifications

As mentioned in Section 8.6 all radio control notifications are sent back on the Radio Control Interrupt (RCI) endpoint. If the device has multiple notifications pending when this endpoint is polled, then it must send all the notifications back together. The device must follow short packet semantics to complete a transfer. A host can filter the notifications that the device must send by using the Set Notification Filter command (see Section 8.6.2.9). Note that the maximum length of data that this endpoint can send is limited to 4K.

8.6.3.1 Application-specific Probe IE Received Notification

This notification informs the host that the device has received an Application-specific Probe IE from another device. The notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-97. Application-specific Probe IE Received Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	0	AS_PROBE_IE_RECEIVED Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wSrcAddr*	2	Number	Identifies the device which sent the Application-specific Probe IE
6	*w IELength*	2	Number	The length of the Application-specific Probe IE
8	*IEData*	Var	Raw Data	A variable size array containing Application-specific Probe IE data.

Host software will use this notification to update the host information on another device's Application-specific Probe IE.

8.6.3.2 Beacon Received Notification

This notification informs the host that a beacon was received. The notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-98. Beacon Received Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	1	BEACON_RECEIVED Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*bChannelNumber*	1	Number	The physical channel on which the beacon was received. The encoding of the channel number is specified in Table 5-12.
5	*wBPSTOffset*	2	Number	The offset of a received beacon relative to the BPST of the device, measured in microseconds
7	*bLQI*	1	Number	Link quality indication.
8	*bRSSI*	1	Number	Receive signal strength indication.
9	*wBeaconInfoLength*	2	Number	The number of bytes in the Beacon Info.
11	*BeaconInfo*	Var	Raw Data	Variable size array containing all the information in the received beacon.

The *BeaconInfo* field is an array including the MAC Header, Beacon Parameters, and all information elements for the received beacon. The length of the *BeaconInfo* array is specified in the *wBeaconInfoLength* field.

The *bChannelNumber* field is the PHY channel on which the beacon was received.

The relative offset to the Beacon Period Start Time (BPST) of the local device to the received beacon measured in microseconds is returned in the *wBPSTOffset* field and is equal to the value of STC at that time. This must be a value between 0 and 65535.

The Link Quality Indication value associated with the received beacon frame is specified the *bLQI* field. This is a value between 0 and 255.

The Received Signal Strength Indication value associated with the received beacon frame is returned in the *bRSSI* field.

8.6.3.3 Beacon Size Notification

This notification informs the host that the size of the beacon has changed due to a modification of one or more of the IEs handled by the MAC.

The RCEB for this notification is given below.

Table 8-99. Beacon Size Change Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	2	BEACON_SIZE_CHANGE Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wNewBeaconSize*	2	Number	Indicates the new size of the beacon including the MAC Header, Beacon Parameters and all the IEs.

8.6.3.4 BPOIE Change Notification

This notification informs the host that the BPOIE that is being transmitted by the device has changed. The notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-100. BPOIE Change Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	3	BPOIE_CHANGE Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wBPOIELength*	2	Number	The length of the BPOIE sent by this device
6	*BPOIE*	Var	Raw Data	Variable size array containing all the information in the BPOIE send by this device

8.6.3.5 BP Slot Change Notification

This notification informs the host that the local device has changed the slot it is sending out its beacon in. This notification must be sent in the following cases:

- After a sending the device beacon the first time (after a successful Start Beaconing command)
- Every time the device changes the slot it sends it’s beacon in due to contraction/expansion of the beacon period

The RCEB for this notification is given below.

Table 8-101. BP Slot Change Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	4	BP_SLOT_CHANGE Event
3	*bEventContext*	1	0	This is a notification. Hence the Event Context is 0.
4	*bSlotNumber*	1	Number	The new slot that the device is sending its beacon in

8.6.3.6 BP Switch IE Received Notification

This notification informs the host that the device detected a BP Switch IE in a beacon from another device. The notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-102. BP Switch IE Received Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	5	BP_SWITCH_IE_RECEIVED Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wSrcAddr*	2	Number	Identifies the device which sent the BP Switch IE
6	*wIELength*	2	Number	The length of the BP Switch IE.
8	*IEData*	Var	Raw Data	A variable size array containing BP Switch IE data.

Host software will use this notification to update the host information on another device's BP Merge status. The host will decide whether and when to switch Beacon periods after receiving this notification.

8.6.3.7 Device Address Conflict Notification

This notification informs the host that the device has detected a device address conflict.

The RCEB for this notification is given below.

Table 8-103. Device Address Conflict Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	6	DEV_ADDR_CONFLICT Event
3	*bEventContext*	1	0	This is a notification. Hence the Event Context is 0.

8.6.3.8 DRP Availability Changed Notification

This notification informs the host that the local device's DRP availability has changed. This notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-104. DRP Availability Changed Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	7	DRP_AVAILABITY_CHANGE Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wIELength*	2	Number	The length of the DRP Availability IE.
6	*IEData*	Var	Raw Data	A variable size array containing DRP Availability IE data.

Host software will use this notification to update the host information on which MASs are available for reservations.

8.6.3.9 DRP Notification

This notification informs the host that a peer device is requesting to create a new reservation or to modify or release an existing reservation. The device must generate this notification when it has received DRP IEs in either a beacon or an explicit DRP command destined to its own address or any multicast address. When the device receives an explicit DRP reservation response from the peer device, it must include all the DRP IEs and the DRP Availability IE in the *IEData*. Additionally the notification is used to notify the host when there is a conflict. The notification is sent back in an RCEB on the Radio Control Interrupt Endpoint.

The RCEB for this notification is given below.

Table 8-105. DRP Notification RCEB

Offset	Field	Size	Value	Description
0	*bEventType*	1	0	GENERAL Event Type
1	*wEvent*	2	8	DRP Event
3	*bEventContext*	1	0	This is an indication. Hence the Event Context is 0.
4	*wSrcAddr*	2	Number	Identifies the device which sent the DRP IE
6	*bExplicit*	1	Flag	Indicates whether implicit or explicit DRP negotiation was used. When this flag is set, then Explicit DRP negotiations was used.
7	*wIELength*	2	Number	The length of the DRP IE data.
9	*IEData*	Var	Raw Data	A variable size array containing IE data.

Host software will evaluate the reservation request/conflict in terms of the device's availability and conflict and then the host will use the Set DRP IE to generate a DRP response.

This page intentionally left blank

Appendix A
Wireless USB CCM Test Vectors

This chapter provides test vectors for testing CCM encryption and decryption logic. The individual vectors are designed to reflect genuine Wireless USB operations. Multi-byte numerical values are presented with the most significant byte on the left and the least significant byte on the right. Byte streams are presented in order of transmission with the first byte transmitted on the left and the last byte transmitted on the right. All values are presented in hexadecimal notation.

A.1 Key Derivation

Wireless USB defines a mechanism for deriving temporal keys from the pre-shared Connection Key (CK).

Inputs	
Host Address	`9876`
Device Address	`00BE`
TKID	`019876`
Host Nonce	`10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1F`
Device Nonce	`20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F`
CK	`F0 E1 D2 C3 B4 A5 96 87 78 69 5A 4B 3C 2D 1E 0F`
Results	
KCK	`4B 79 A3 CF E5 53 23 9D D7 C1 6D 1C 2D AB 6D 3F`
PTK	`C8 70 62 82 B6 7C E9 06 7B C5 25 69 F2 36 61 2D`

A.2 Handshake MIC calculation

Wireless USB defines a mechanism for generating the MIC values used to protect Handshake2 and Handshake3 messages. This vector provides the input and outputs for the MIC calculation of a Handshake2 request.

Inputs	
Message Number	`2`
Host Address	`9876`
Device Address	`00BE`
TKID	`019876`
Device Nonce	`20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F`
KCK	`4B 79 A3 CF E5 53 23 9D D7 C1 6D 1C 2D AB 6D 3F`
CDID	`30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F`
Results	

Handshake2 Data	`02 00 76 98 01 00 30 31 32 33 34 35 36 37 38 39`
	`3A 3B 3C 3D 3E 3F 20 21 22 23 24 25 26 27 28 29`

	2A 2B 2C 2D 2E 2F	
MIC	75 6A 97 51 0C 8C 14 7B	

A.3 Secure MMC (EO = payload length)

This vector presents a secured MMC containing a Host Information IE and a WdntsCTA.

Inputs		
Host Address	9876	
Device Address	FFFF	
TKID	019876	
KEY	C8 70 62 82 B6 7C E9 06 7B C5 25 69 F2 36 61 2D	
SFC	001122334455	
Packet	40 1C FF FF 76 98 00 00 23 81	(MAC Header)
	00 01 00 23 00 05 04 0F 0E 0D	(MMC Header)
	80 10 00 0C	(WdntsCTA)
	00 00 01 FF	(end of list)
	14 82 49 00	(Host Info IE)
	A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF	
EO	26	(l(m)= 0, l(a) = 34)
Results		
MAC HDR	48 1C FF FF 76 98 00 00 23 81	
Security Hdr	76 98 01 00 26 00 55 44 33 22 11 00	
Payload	00 01 00 23 00 05 04 0F 0E 0D 80 10	
	00 0C 00 00 01 FF 14 82 49 00 A0 A1	
	A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF	
MIC	98 3B 8B A5 C1 A8 F8 39	

A.4 Data IN from device (EO = 2)

This vector is for a Data IN packet from a device. The Wireless USB header is authenticated but not encrypted. The data portion of the payload is fully encrypted.

Inputs

Host Address	`9876`	
Device Address	`0002`	
TKID	`019876`	
KEY	`C8 70 62 82 B6 7C E9 06 7B C5 25 69 F2 36 61 2D`	
SFC	`001122334456`	
Packet	`C0 12 76 98 02 00 00 00 23 C1`	`(MAC Header)`
	`81 00`	`(WUSB Header)`
	`30 31 32 33 34 35 36 37`	`(data)`
	`38 39 3A 3B 3C 3D 3E 3F`	
	`40 41 42 43 44 45 46 47`	
	`48 49 4A 4B 4C 4D 4E 4F`	
EO	`02`	`( l(m)= 20, l(a) = 10 )`

Results

MAC HDR	`C8 12 76 98 02 00 00 00 23 C1`
Security Hdr	`76 98 01 00 02 00 56 44 33 22 11 00`
Payload	`81 00 41 3A 31 85 C9 85 1B F5 46 E7`
	`C5 93 03 11 85 76 47 ED 9D 95 15 A6`
	`99 CF 47 79 CE C8 6E B0 AD 1D`
MIC	`AE 8C 60 D3  BA F2 AD 46`

Appendix B
Wire Adapter Example Descriptor Sets

B.1 Descriptors for DWA

The Wire Adapter class pre-defines certain fields in standard USB descriptors. Other fields are either implementation-dependent or not applicable to this class.

The sample descriptor set below is for a DWA with 4 ports (all exposed) with the ability to support a pair of isochronous streams to devices connected on the DWA's downstream port. This DWA supports one encryption type (AES 128 CCM), has 16 RPipes and 256K of buffer space (block size of 2K and 128 blocks). This DWA does not support data packet size adjustment.

Note: For the descriptors and fields shown below, the bits in a field are organized in a little-endian fashion; that is, bit location 0 is the least significant bit and bit location 7 is the most significant bit of a byte value.

Device Descriptor:

bLength	12H
bDescriptorType	01H (Device)
bcdUSB	0250H (Wireless USB Compliant)
bDeviceClass	EFH (Miscellaneous)
bDeviceSubClass	02H (Common Class)
bDeviceProtocol	02H (Wire Adapter Multifunction Peripheral)
bMaxPacketSize0	FFH
idVendor	Implementation-dependent
idProduct	Implementation-dependent
bcdDevice	Implementation-dependent
iManufacturer	Implementation-dependent
iProduct	Implementation-dependent
iSerialNumber	Implementation-dependent
bNumConfigurations	1

BOS Descriptor:

bLength	05H
bDescriptorType	0FH (BOS)
wTotalLength	10H
bNumDeviceCaps	1

Wireless USB Device Capabilities – UWB:

bLength	0BH
bDescriptorType	10H (Device Capability)
bDevCapabilityType	01H (Wireless USB)
bmAttributes	Implementation-dependent
wPHYRates	Implementation-dependent
bmTFITXPowerInfo	Implementation-dependent
bmFFITXPowerInfo	Implementation-dependent
bmBandGroup	Implementation-dependent
bReserved	0

Security Descriptor (One Encryption Type supported):

bLength	05H
bDescriptorType	0CH (Security)
wTotalLength	0AH
bNumEncryptionTypes	1

Encryption Type Descriptor (AES-128 in CCM mode):

bLength	05H
bDescriptorType	0EH (Encryption Type)
bEncryptionType	02H (AES-128 in CCM mode)
bEncryptionValue	Implementation-dependent
bAuthKeyIndex	Implementation-dependent

Configuration Descriptor

bLength	09H
bDescriptorType	02H (Configuration)
wTotalLength	N (Implementation-dependent)
bNumInterfaces	2
bConfigurationValue	Implementation-dependent
iConfiguration	Implementation-dependent
bmAttributes	Implementation-dependent
bMaxPower	0

Interface Association Descriptor

bLength	08H
bDescriptorType	0BH (Interface Association)
bFirstInterface	0
bInterfaceCount	02H
bFunctionClass	E0H (Wireless Controller)
bFunctionSubClass	02H (Wireless USB Adapter)
bFunctionProtocol	02H (Device Wire Adapter Control/Data Streaming Programming Interface)
iFunction	Implementation-dependent

Interface Descriptor (Data Transfer Interface):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	3
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	02H (Wireless USB Adapter)
bInterfaceProtocol	02H (Device Wire Adapter Control/Data Streaming Programming Interface)
iInterface	Implementation-dependent

Wire Adapter Class Descriptor:

bLength	0EH
bDescriptorType	21H (Wire Adapter Descriptor Type)
bcdWAVersion	100H (WA Class Specification Version)
bNumPorts	4
bmAttributes	Implementation-dependent
wNumRPipes	10H
wRPipeMaxBlock	80H
bRPipeBlockSize	0CH
bPwrOn2PwrGood	Implementation-dependent
bNumMMCIEs	0
DeviceRemovable	1EH

Endpoint Descriptor (for Notification Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	200H
bInterval	06H

Endpoint Companion Descriptor (for Notification Endpoint)

bLength	0AH
bDescriptorType	11H (Wireless Endpoint Companion)
bMaxBurst	1
bMaxSequence	Implementation-dependent
wMaxStreamDelay	0
wOverTheAirPacketSize	0
bOverTheAirInterval	0
bmCompAttributes	0

Endpoint Descriptor (for Data Transfer Write Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = Out(0)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	Implementation-dependent
bInterval	0

Endpoint Companion Descriptor (for Data Transfer Write Endpoint)

bLength	0AH
bDescriptorType	11H (Wireless Endpoint Companion)
bMaxBurst	Implementation-dependent
bMaxSequence	Implementation-dependent
wMaxStreamDelay	0
wOverTheAirPacketSize	0
bOverTheAirInterval	0
bmCompAttributes	0

Endpoint Descriptor (for Data Transfer Read Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	Implementation-dependent
bInterval	0

Endpoint Companion Descriptor (for Data Transfer Read Endpoint)

bLength	0AH
bDescriptorType	11H (Wireless Endpoint Companion)
bMaxBurst	Implementation-dependent
bMaxSequence	Implementation-dependent
wMaxStreamDelay	0
wOverTheAirPacketSize	0
bOverTheAirInterval	0
bmCompAttributes	0

Interface Descriptor (Isochronous Streaming Interface):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	1
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	02H (Wireless USB Wire Adapter)
bInterfaceProtocol	03H (Device Wire Adapter Isochronous Streaming Programming Interface)
iInterface	Implementation-dependent

Endpoint Descriptor (for Isochronous Streaming OUT Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = Out(0)
bmAttributes	Transfer Type = Isochronous (00000001B)
wMaxPacketSize	Implementation-dependent
bInterval	Implementation-dependent

Endpoint Companion Descriptor (for Isochronous Streaming OUT Endpoint)

bLength	0AH
bDescriptorType	11H (Wireless Endpoint Companion)
bMaxBurst	Implementation-dependent
bMaxSequence	Implementation-dependent
wMaxStreamDelay	Implementation-dependent
wOverTheAirPacketSize	Implementation-dependent
bOverTheAirInterval	0
bmCompAttributes	02H (Continuously scalable dynamic switching)

Endpoint Descriptor (for Isochronous Streaming IN Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Isochronous (00000001B)
wMaxPacketSize	Implementation-dependent
bInterval	Implementation-dependent

Endpoint Companion Descriptor (for Isochronous Streaming IN Endpoint)

bLength	0AH
bDescriptorType	11H (Wireless Endpoint Companion)
bMaxBurst	Implementation-dependent
bMaxSequence	Implementation-dependent
wMaxStreamDelay	Implementation-dependent
wOverTheAirPacketSize	Implementation-dependent
bOverTheAirInterval	0
bmCompAttributes	02H (Continuously scalable dynamic switching)

B.2 Descriptors for HWA

The sample descriptor set below is for an HWA operating on a high-speed USB 2.0 bus, which can connect up to 16 devices simultaneously on its downstream Wireless USB bus. This HWA supports one encryption type (AES 128 CCM), has 32 RPipes and 256K of buffer space (block size of 4K and 64 blocks). The HWA also exports a Radio Control Interface to communicate with the UWB radio in the device.

Device Descriptor (high-speed information):

bLength	12H
bDescriptorType	01H
bcdUSB	0200H (USB 2.0 compliant)
bDeviceClass	EFH (Miscellaneous)
bDeviceSubClass	02H (Common Class)
bDeviceProtocol	02H (Wire Adapter Multifunction Peripheral)
bMaxPacketSize0	64

idVendor	Implementation-dependent
idProduct	Implementation-dependent
bcdDevice	Implementation-dependent
iManufacturer	Implementation-dependent
iProduct	Implementation-dependent
iSerialNumber	Implementation-dependent
bNumConfigurations	1

Device_Qualifier Descriptor (full-speed information)

bLength	0AH
bDescriptorType	06H (Device Qualifier)
bcdUSB	0200H (USB 2.0 Compliant)
bDeviceClass	EFH (Miscellaneous)
bDeviceSubClass	02H (Common Class)
bDeviceProtocol	02H (Wire Adapter Multifunction Peripheral)
bMaxPacketSize0	64
bNumConfigurations	1
bReserved	0

Security Descriptor (One Encryption Type supported):

bLength	05H
bDescriptorType	0CH (Security)
wTotalLength	0AH
bNumEncryptionTypes	1

Encryption Type Descriptor (AES-128 in CCM mode):

bLength	05H
bDescriptorType	0EH (Encryption Type)
bEncryptionType	02H (AES-128 in CCM mode)
bEncryptionValue	Implementation-dependent
bAuthKeyIndex	Implementation-dependent

Configuration Descriptor (high-speed information)

bLength	09H
bDescriptorType	02H (Configuration)
wTotalLength	N
bNumInterfaces	2
bConfigurationValue	Implementation-dependent
iConfiguration	Implementation-dependent
bmAttributes	Implementation-dependent
bMaxPower	The minimum amount of bus power the HWA will consume in this configuration

Interface Descriptor (Data Transfer Interface):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	3
bInterfaceClass	E0H
bInterfaceSubClass	02H
bInterfaceProtocol	01H (Host Wire Adapter)
iInterface	Implementation-dependent

Wire Adapter Class Descriptor:

bLength	0EH
bDescriptorType	21H (Wire Adapter Descriptor Type)
bcdWAVersion	100H (WA Class Specification Version)
bNumPorts	16
bmAttributes	0
wNumRPipes	20H
wRPipeMaxBlock	40H
bRPipeBlockSize	0DH
bPwrOn2PwrGood	0
bNumMMCIEs	4
DeviceRemovable	0

Endpoint Descriptor (for Notification Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	40H
bInterval	1

Endpoint Descriptor (for Data Transfer Write Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = Out(0)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	200H
bInterval	0

Endpoint Descriptor (for Data Transfer Read Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	200H
bInterval	0

Interface Descriptor (Radio Control Interface Alternate Setting 0):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	1
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	01H (RF Controller)
bInterfaceProtocol	02H (UWB Radio Control Interface Programming Interface
iInterface	Implementation-dependent

Radio Control Interface Class Descriptor:

bLength	04H
bDescriptorType	23H (Radio Control Descriptor Type)
bcdRCIVersion	0100H (Radio Control Interface Version)

Endpoint Descriptor (for Radio Control Interrupt Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	00H
bInterval	1

Interface Descriptor (Radio Control Interface Alternate Setting 1):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	1
bAlternateSetting	1
bNumEndpoints	1
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	01H (RF Controller)
bInterfaceProtocol	02H (UWB Radio Control Interface Programming Interface)
iInterface	Implementation-dependent

Radio Control Interface Class Descriptor:

bLength	04H
bDescriptorType	23H (Radio Control Descriptor Type)
bcdRCIVersion	0100H (Radio Control Interface Version)

Endpoint Descriptor (for Radio Control Interrupt Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	200H
bInterval	1

Other_Speed_Configuration Descriptor (full-speed information)

bLength	09H
bDescriptorType	07H (Other Speed Configuration)
wTotalLength	N
bNumInterfaces	2
bConfigurationValue	Implementation-dependent
iConfiguration	Implementation-dependent
bmAttributes	Implementation-dependent
bMaxPower	The minimum amount of bus power the HWA will consume in full-speed configuration

Interface Descriptor (Data Transfer Interface):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	3
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	02H (Wireless USB Wire Adapter)
bInterfaceProtocol	01H (Host Wire Adapter Control/Data Streaming Programming Interface)
iInterface	Implementation-dependent

Wire Adapter Class Descriptor:

bLength	0CH
bDescriptorType	21H (Wire Adapter Descriptor Type)
bcdWAVersion	100H (WA Class Specification Version)
bNumPorts	16
bmAttributes	0
wNumRPipes	20H
wRPipeMaxBlock	40H
bRPipeBlockSize	0DH
bPwrOn2PwrGood	0
bNumMMCIEs	4
DeviceRemovable	0

Endpoint Descriptor (for Notification Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	40H
bInterval	1

Endpoint Descriptor (for Data Transfer Write Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = Out(0)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	40H
bInterval	0

Endpoint Descriptor (for Data Transfer Read Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Bulk (00000010B)
wMaxPacketSize	40H
bInterval	0

Interface Descriptor (Radio Control Interface):

bLength	09H
bDescriptorType	04H (Interface)
bInterfaceNumber	1
bAlternateSetting	0
bNumEndpoints	1
bInterfaceClass	E0H (Wireless Controller)
bInterfaceSubClass	01H (RF Controller)
bInterfaceProtocol	02H (UWB Radio Control Interface Programming Interface)
iInterface	Implementation-dependent

Radio Control Interface Class Descriptor:

bLength	04H
bDescriptorType	23H (Radio Control Descriptor Type)
bcdRCIVersion	0100H (Radio Control Interface Version)

Endpoint Descriptor (for Radio Control Interrupt Endpoint):

bLength	07H
bDescriptorType	05H (Endpoint)
bEndpointAddress	Implementation-dependent; Bit 7: Direction = In(1)
bmAttributes	Transfer Type = Interrupt (00000011B)
wMaxPacketSize	64H
bInterval	1