Universal Serial Bus Device Class Definition for Personal Healthcare Devices

Release 1.0
November 8, 2007

Scope of This Release

This document is the 1.0 release of this device class definition.

Contributors

Company	Contributor	Email Address
Cisco	Mark Schnell	MSchnell@Cisco.com
Frontline Test Equipment	David Bean	DBean@FTE.com
Frontline Test Equipment	Matt Miller	MMiller@FTE.com
Intel Corporation	Doug Bogia	Douglas.P.Bogia@Intel.com
Intel Corporation	Julie Fleischer	Julie.N.Fleischer@Intel.com
Intel Corporation	John Keys	John.Keys@Intel.com
Littelfuse	Max Bassler	MBassler@Interacttech.net
Littelfuse	Bill Travis	BTravis@Littelfuse.com
Littelfuse	Steve Whitney	SWhitney@Littelfuse.com
MCCI	Jason Oliver	Jason.Oliver@MCCI.com
Nonin Medical	Robert Hoy	Robert.Hoy@Nonin.com
Welch Allyn	Song Chung	ChungS@WelchAllyn.com

Revision History

Revision	Date	Filename	Description
1.0	November 8, 2007	healthcare_1.0.doc	Initial release

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

Previously, personal healthcare devices such as glucose meters and pulse oximeters were required to implement proprietary methods for sending personal healthcare data to a Universal Serial Bus (USB) host. The vision of the USB Personal Healthcare Device Class (PHDC) is to enable seamless interoperability between personal healthcare devices and USB hosts, which will allow for a variety of new use cases for both devices and hosts.

1.2 Scope

This document describes the architecture, descriptors, and requests that a personal healthcare device and host must support in order to exchange personal healthcare data over USB. This document does not define a data and messaging format; instead, it provides a generic mechanism by which standardized data and messages can be sent over USB. Appendix A provides the requirements necessary to use the PHDC to send the personal healthcare data and messages defined in data and messaging standards. The ISO/IEEE 11073-20601 Base Exchange Protocol™ is the only data and messaging standard referenced by this specification.

1.3 Related Documents

The following documents are referenced in the PHDC.

1. Universal Serial Bus Specification, Revision 2.0, April 27, 2000, http://www.usb.org.
2. ISO/IEEE 11073-20601 Optimized Exchange Protocol™, Revision 1.0, Date TBD, Not yet published.
3. ISO/IEEE 11073-10101, Health Informatices – Point of Care medical device communication™ – Part 10101: Nomenclature, First edition, December 15, 2004.
4. RFC 2119, Key words for use in RFCs to Indicate Requirement Levels: March 1997, http://www.ietf.org/rfc/rfc2119.txt.

1.4 Terms

Table 1 – PHDC Terms

Term	Description
Alarms	An indication of either physiological conditions, equipment conditions, or other conditions that need attention.
Commands	The ability to direct a device to send information, set configurations, and/or execute actions.
Configuration Information	Specific operational data necessary for the correct operation of a device.
Device Specialization	A particular use for a personal healthcare device, such as a glucose meter, blood pressure monitor, or thermometer. Also known as the device's modality or function.
Episodic	Episodic data collection and/or episodic data transfer corresponds to an episode, usually at irregular intervals. The time between samples can vary widely from seconds to weeks or longer.
Events	An event is generated by a component whenever there is a condition that is potentially interesting and may include physiological conditions such as an elevated heart rate or a device condition such as a low battery voltage.
Function	See **Device Specialization**. In the main body of this document, the term function will be used. In the Appendix section, the term **Device Specialization** is used.

Term	Description
Latency	The time needed for a piece of information to travel from one communication end point, through the intervening communication system to another communications end point. In the personal healthcare environment, an analysis of the healthcare data resulted in the definition of four latency categories or bins: ▪ **Low latency**: <20ms ▪ **Medium latency**: <200ms ▪ **High latency**: <2sec ▪ **Very high latency**: <20sec
Modality	See **Device Specialization**.
Multi-Function Device	A personal healthcare device that implements two separate device specializations, such as a blood pressure monitor and a heart rate monitor.
Notifications	See **Events**.
Personal Healthcare Device	A USB device that implements a personal healthcare function, such as a USB blood pressure monitor or a USB weight scale.
Physiological	Dealing with or relating to functions and activities of life, and of the physical and chemical phenomena involved.
Quality of Service (QoS)	A generic term that refers to any number of communication characteristics that are deemed important to an application. In the context of Personal Healthcare applications, the two most important characteristics are latency and reliability.
Reliability	A measure of the probability that a given piece of information will be successfully sent from one communication end point, through the intervening communication system, to another communications end point. In the personal healthcare environment, an analysis of the healthcare data resulted in the definition of three reliability categories or bins: ▪ **Good Reliability**: typically used for physiological waveforms and other such continuous analog functions. An occasional loss of data will not significantly impair the use of the transmitted information. ▪ **Better Reliability**: typically used for episodic measurements. The loss of measurement data is not well tolerated, but the loss of a data measurement is more acceptable than the loss of an alarm event. This reliability category is between **Good** and **Best**. ▪ **Best Reliability**: typically used for alarm and other critical need events where a loss of information must be avoided at nearly all costs.

1.5 Acronyms

The following acronyms are used throughout the document.

Table 2 – Acronyms

Acronym	Description
AKA / aka	Also Known As
APDU	Application Protocol Data Units
DWG	**Device Working Group** – USB-IF parent working group for device class working groups (WGs) such as the Personal Healthcare WG.
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
PDU	Protocol Data Units
PHD	**Personal Health Devices** – The ISO/IEEE 11073 PHD specifications are the

Acronym	Description
	first data and messaging standards supported by this device class.
PHDC	**Personal Healthcare Device Class** – This specification defines how personal healthcare devices seamlessly interact with USB hosts.
PH WG	**Personal Healthcare Working Group** – The DWG working group responsible for writing the Personal Healthcare Device Class.
QoS	Quality of Service

1.6 Requirements Terminology

In this document, the terms **shall**, **should**, and **may** are used to designate normative text as defined in RFC 2119, as referenced in Section 1.3. When these terms are used normatively, they are denoted by bold font. Any text that does not contain **shall**, **should**, or **may** is informative.

Shall statements define requirements on devices or hosts implementing the PHDC. These features must be part of the implementation in order for the implementation to be considered compliant with the PHDC.

Should statements define requirements where there may exist valid reasons for not implementing the requirement; however, the implications need to be carefully weighed before making this choice.

May statements define optional requirements on devices or hosts implementing the PHDC. These requirements are optional to implement; however, if they are implemented, they must be implemented as defined within the device class.

1.7 Styles & Formatting

In this document, *italics* are used to denote descriptor and preamble fields when mentioned outside of the descriptor or preamble table. *Italics* are also used to denote special features such as the *Meta-Data Message Preamble* feature. **Bold** text is also used for emphasis.

2 Management Overview

2.1 Personal Healthcare Use Cases

The Personal Healthcare Device Class (PHDC) defines the functionality necessary for personal healthcare devices to send standardized data and messages, such as those in ISO/IEEE 11073-20601 Base Exchange Protocol™, to hosts over USB. This enables a variety of use cases where personal healthcare data can be gathered from a device, sent via wired USB to a USB host, and then sent via the internet to a healthcare provider, loved one, or into personal health data storage as shown in Figure 1 – Use Cases Enabled by PHDC.

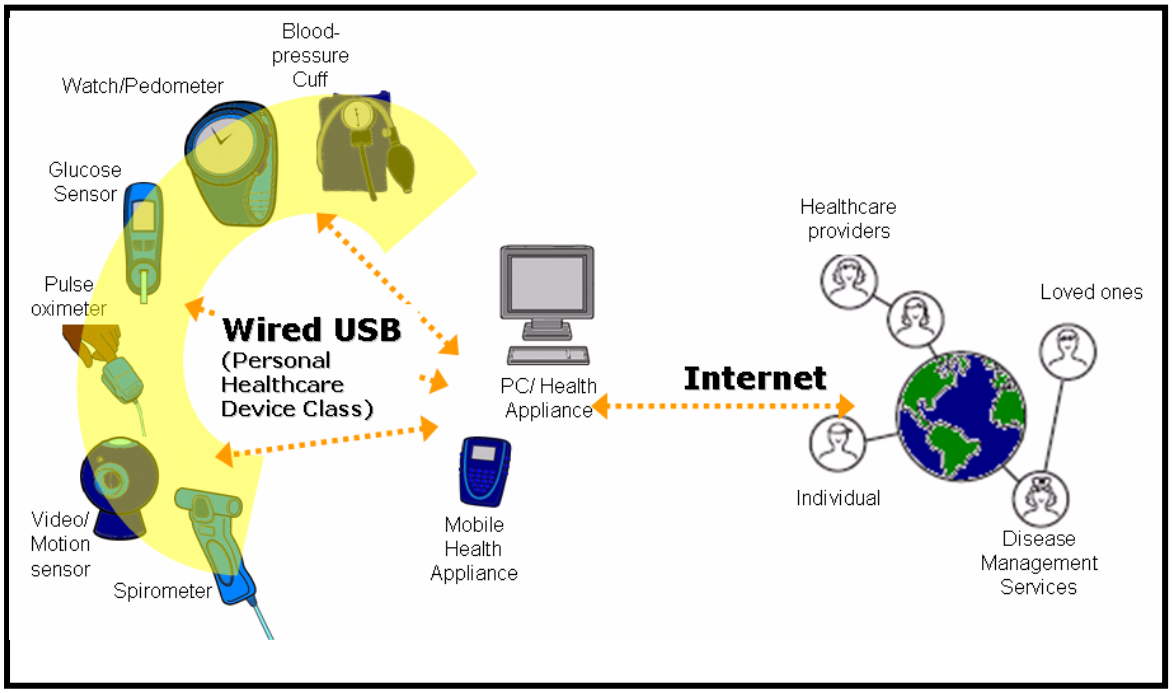

Figure 1 – Use Cases Enabled by PHDC

These use cases fall into the three theme areas defined in Table 3 – PHDC Theme Areas.

Table 3 – PHDC Theme Areas

Theme	Description
Health and Wellness	This theme focuses on enabling generally healthy people to stay fit by transferring data from USB fitness devices or general healthcare devices, such as exercise watches, heart rate monitors, exercise bikes, and blood pressure monitors, to devices like PCs. The uploaded information can then be forwarded to a coach for advice and assessment or utilized in a community for competition and motivation. It could also be stored in a personal healthcare database which the individual could access.
Disease Management	This theme focuses on enhancing the monitoring of chronic conditions by transferring data from USB healthcare devices, such as blood pressure monitors, glucose monitors, weight scales, pulse oximeters, and heart rate monitors, to PCs or health appliances. The uploaded information can then be forwarded to a healthcare provider for analysis and action.
Aging	This theme prolongs independent living by sending information from USB

Theme	Description
Independently	motion sensors, pill monitors, or other devices that monitor activities of daily living on to health appliances or PCs. This information can then be forwarded to healthcare providers for analysis and action or remote caregivers and family for social interaction.

3 Functional Characteristics

3.1 Data Transfer Characteristics

The personal healthcare devices described in Section 2.1 can send data to hosts in three basic ways:

1. Data can be sent in an **episodic** fashion to the host. For example, a user may have a USB weight scale connected to a health appliance. Each time the user steps on the weight scale, her weight may be transferred to the appliance. These can be extremely simple devices that need to transmit their data sample and disconnect; the devices may queue messages to the endpoint hardware upon reaching configured state and disconnect/sleep upon sending that data.
2. Data may be **stored and forwarded** to the host as a large group of data. These devices are ones that perform their functions when not connected to the USB bus. For example, a user may take a fitness watch out for a jog, collect data throughout the exercise, and connect it up to his PC when he returns.
3. Data may be sent to the host in a **continuous** fashion. For example, a user may connect a pulse oximeter to her PC via wired USB and then place the pulse oximeter on her finger so that she can continuously monitor her oxygen saturation levels.

Hosts may also send data back to the device in an **episodic** fashion. Generally, this will be small amounts of data; however, it could be large amount of configuration or other data for the device.

3.2 Personal Healthcare Data QoS Bins

The types of data transfers previously defined can be more finely described in terms of latency and reliability QoS requirements. For example, data that is sent to a host in a continuous fashion would be considered to have Good reliability and Low latency. Figure 2 – Mapping of Latency and Reliability to PHDC Data Types displays latency and reliability characteristics. The cells containing graphics are the QoS combinations that occur in the personal healthcare use cases defined in Section 2.1 and need to be supported by the PHDC.

App Data Type *(& typical use)*		Relative reliability		
		Good	**Better**	**Best**
Max. latency tolerance (transport latency budget)	**Very high** (<20s)			*VHL/Best* 
	High (<2s)			*HL/Best* 
	Medium (<200ms)	*ML/Good* 	*ML/Better* 	*ML/Best* 
	Low (<20ms)	*LL/Good* 		

Figure 2 – Mapping of Latency and Reliability to PHDC Data Types

Table 4 – QoS Bin Descriptions provides additional detail on the six QoS combinations, aka the QoS "bins," that need to be supported by the PHDC. USB can easily satisfy all six combinations using bulk and interrupt endpoints.

Table 4 – QoS Bin Descriptions

Typical Description	QoS Bins (Latency. Reliability)
▪ Raw information rates are in the **50 bit/sec to 1.2M bit/sec** range ▪ The analog sampling rates in the 1 ms to 50 ms range ▪ The sampled analog data can easily be grouped together; packets of grouped analog samples could easily have a **packet period** in the **10s of millisecond** range ▪ Data sent **TO** host ▪ **Good** relative reliability needs	**Low. Good** 
▪ Raw information rates are in the **50 bit/sec to 1.2M bit/sec** range. ▪ Latency allowed to be as high as **200ms**. ▪ Data sent **TO** host ▪ **Good** relative reliability needs	**Medium. Good** 
▪ A measured parameter (ex., BP, SpO2, HR); Data may be collected off-line and replayed or sent real-time ▪ Typical raw information payload in the **10s of bytes** range ▪ The **fastest** communications rate is in **1 sec** per measurement range ▪ Data sent **TO** host ▪ **Better** relative reliability needs	**Medium. Better** 
▪ AKA get/set device parameters; aka Events and/or notifications; aka **request/response** ▪ Control/status of both physiological and equipment functionality ▪ Typical raw information payload in the **10s of bytes** range ▪ **Human response time** sort of **delay** metrics (<1 sec) ▪ Data sent **TO** or **FROM** host ▪ **Best** relative reliability needs	**Medium. Best** 
▪ Both physiological driven alarms and equipment issued alarms ▪ Supports multiple alarm level such as HIGH, MEDIUM, LOW, NOTE with multiple latency needs ▪ **Latency requirements differ by situation** such as clinical to city to rural ▪ Typical raw information payload in the **10s of bytes** range ▪ Worst case is clinical maximum @ 10sec per AAMI ECG-13 onset-to-alarm time ▪ Data sent **TO** or **FROM** host ▪ **Best** relative reliability needs	**High. Best** 
▪ Print or transfer summaries, reports or histories or **Off line collection** and delayed transfer of episodic type measurements in a group ▪ Vast range of overall payload size (**10s of bytes to mega or even gigabytes** of data) ▪ **Very delay tolerant** data category (can live with 10s of seconds) ▪ Data sent **TO** or **FROM** host (**TO** only for off-line data collection) ▪ **Best** relative reliability needs	**VeryHigh. Best** 

4 Operational Model

4.1 Device & Host Interactions

Figure 3 – USB Personal Healthcare Device and Host Interactions defines how devices and hosts that implement the PHDC will interact.

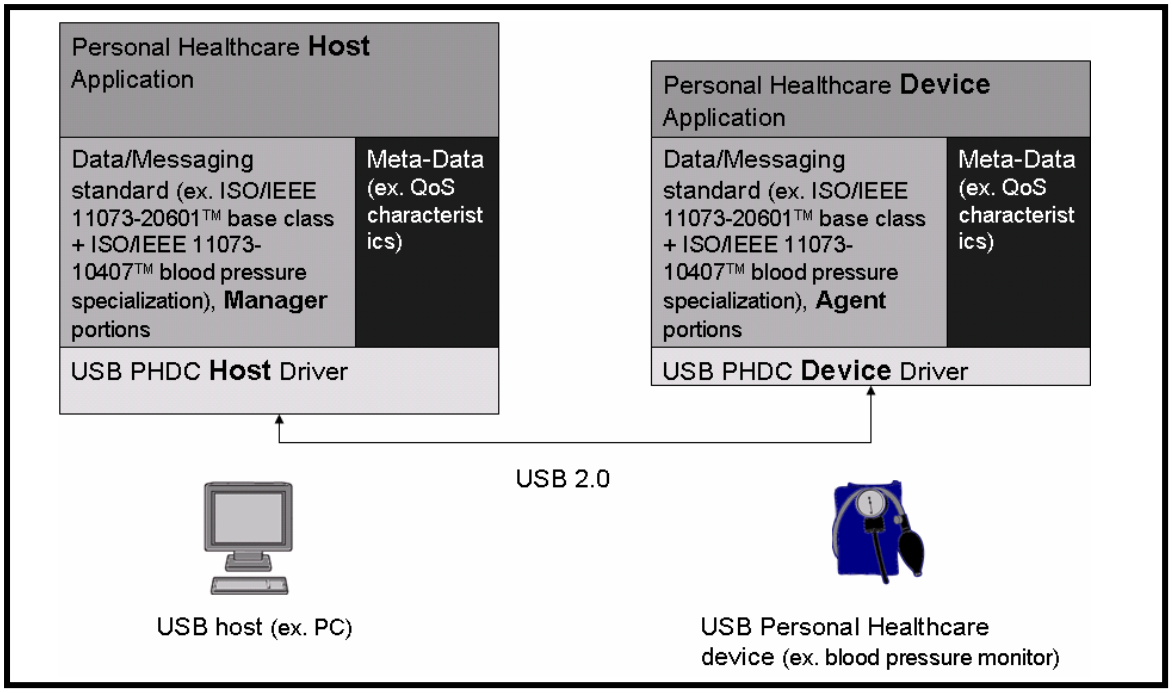

Figure 3 – USB Personal Healthcare Device and Host Interactions

The USB PHDC driver in Figure 3 – USB Personal Healthcare Device and Host Interactions is a generic driver that transports standardized data and messages, designated in the "Data/Messaging standard" box, over the USB transport. The architecture for the PHDC was defined by analyzing the personal healthcare use cases and device characteristics defined in Sections 2.1. This gave rise to the latency and reliability characteristics from Table 4 – QoS Bin Descriptions, which were then mapped into USB endpoints.

Appendix A provides information on how to configure PHDC to send data and messages from supported data and messaging standards. The ISO/IEEE 11073-20601 Optimized Exchange Protocol™ is the initial target for standardized data and messages.

Both personal healthcare hosts and devices have an application layer above the data and messaging layer. This application layer is responsible for formatting and parsing exchanged messages. It is also responsible for providing any meta-data the PHDC driver needs which is not included in the message data. For example, the application layer may need to send QoS information along with message so that the PHDC driver understands how to prioritize a message queue. Further, it may be necessary to send this information over USB along with the personal healthcare data in order to enable broader use cases such as instances when the application layer is replaced with a bridging mechanism that transfers the healthcare data to another transport. This process is further described in Section 4.3.

Other requirements that must be supported by PHDC include: when a personal healthcare device is connected to a USB host, there must be a standard mechanism for recognizing the device as belonging to the USB PHDC, as well as for recognizing the device function such as blood pressure monitor or thermometer. There also must be a mechanism for dealing with devices that have more than one personal healthcare function such as blood pressure monitor and heart rate monitor. Sometimes, these functions may have different data QoS requirements. A final requirement on the PHDC is that a wired USB transport is assumed.

4.2 Personal Healthcare Application & Data Layer Assumptions

As described in Section 4.1, the USB PHDC is one piece of a personal healthcare device solution. A personal healthcare device also needs to implement a data layer and an application layer. Because of this dependency on a standardized data layer with an application layer above, the PHDC needs to support the requirements of the personal healthcare application and data layer above it. This segmentation between the application and data layer and the transport layer follows the classic Layer 7-5 and Layer 4-1 split of the ISO 7 layer model as displayed in Figure 4 – Application and Data Layer and Transport Layer Segmentation.

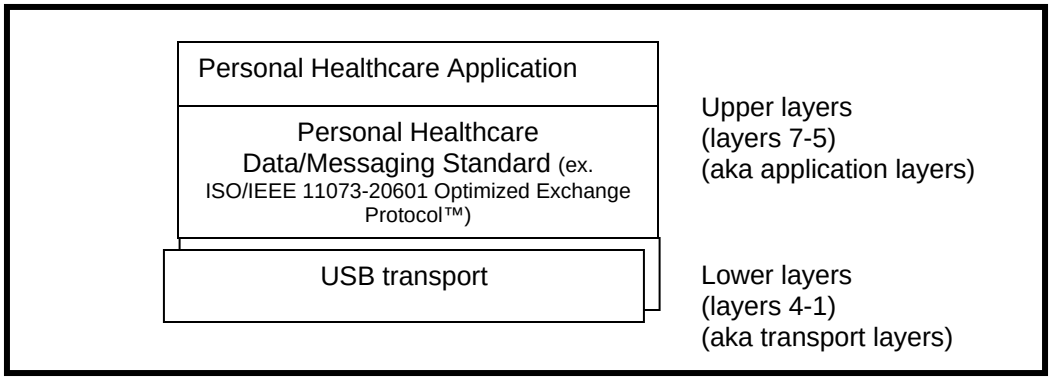

Figure 4 – Application and Data Layer and Transport Layer Segmentation

In order to create a PHDC that sends standardized data and messages, the PHDC defines the following assumptions on the application and data layers. All data and messaging standards in Appendix A support these assumptions.

The PHDC expects the personal healthcare application and data layer to:

1. Send data to the PHDC in message frames such as application PDUs or APDUs no larger than 63KB.
2. Receive data from the PHDC in message frames such as application PDUs or APDUs no larger than 63KB.
3. Enable PHDC to receive and send personal healthcare data as an opaque block of data. If it is necessary for PHDC to know something about the data being sent, this will be sent as meta-data alongside the actual data.
4. Handle ensuring the security and privacy of personal healthcare data. Since a wired USB connection is assumed by the PHDC, the PHDC assumes physical security.
5. If multiple ‘channels’ to a device within a single device/host association are supported, use QoS attributes sent as meta-data to differentiate the characteristics of the ‘channels’.
6. If only a single ‘channel’ to a device within a single device/host association is supported, use QoS attributes sent as meta-data to describe the characteristics of that ‘channel’.
7. Use the following QoS attributes to describe the latency and reliability characteristics of the ‘channel’ (as described in Table 4 – QoS Bin Descriptions):
    a. **Low.Good:** Low latency (<20ms) with ‘good’ reliability.
    b. **Medium.Good**: Medium latency (< 200ms) with ‘good’ reliability.
    c. **Medium.Better:** Medium latency (<200ms) with ‘better’ reliability.

d. **Medium.Best:** Medium latency (<200ms) with 'best' reliability.
e. **High.Best:** High latency (<2sec) with 'best' reliability.
f. **VeryHigh.Best:** Very high latency (<20sec) with 'best' reliability.

NOTE: In the future, additional attributes could be added as meta-data. A non exhaustive list of possible attributes include: bandwidth, host-to-device channel set up time, device-to-host set up time, connection confidence (keep alives) and energy consumption.

4.3 Application Meta-Data

As described in Section 4.1, the application may need to send meta-data along with the message data. Some meta-data, such as the QoS latency/reliability information, is used by the local PHDC component and also transferred over USB to the remote PHDC component and application layer. Additional meta-data, such as opaque meta-data, is transferred over USB without interpretation by PHDC components. Transferring meta-data over USB enables use cases where the application layer is replaced with a bridging mechanism that forwards meta-data and message data to a remote application over another transport. Separation of elements of interest to USB and PHDC, such as QoS reliability/latency information, allows the opaque Meta-Data component to continue to evolve without affecting this specification.

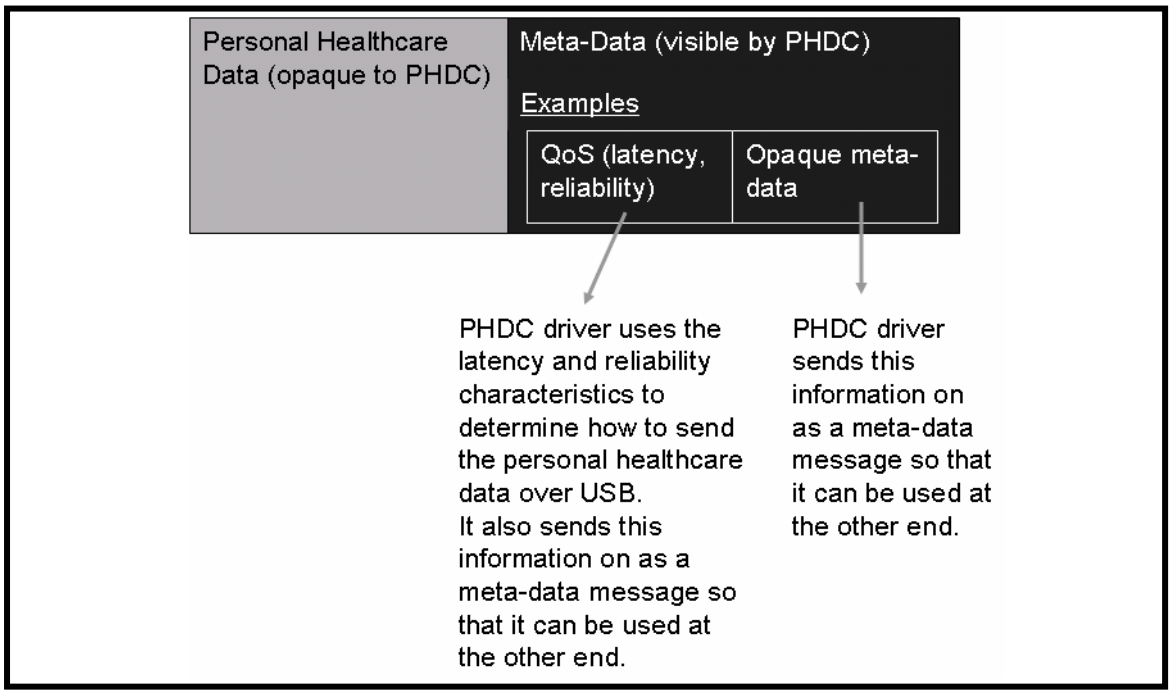

Figure 5 – Sending Meta-Data over PHDC

4.4 Architecture

In order to accomplish the requirements above, the PHDC defines a set of endpoints that implement the QoS criteria defined in Table 4 – QoS Bin Descriptions, and it also defines a mechanism for transferring meta-data.

The personal healthcare architecture defines three required endpoints and a fourth optional endpoint.

1. Control endpoint – Required by USB.
2. Bulk OUT endpoint – This provides a path for transfers from the USB host to the device.
3. Bulk IN endpoint – This provides a path for transfers from the USB device to the host.

4. Interrupt IN endpoint (optional) – This provides a path for transfers from the USB device to the host when it is necessary to send data continuously. For example, the “pleth” waveform in a pulse oximeter.

4.4.1 IN Endpoints

Figure 6 – Map of Application QoS Bins to USB IN Endpoint Types maps the QoS bins defined in Table 4 – QoS Bin Descriptions into IN endpoints.

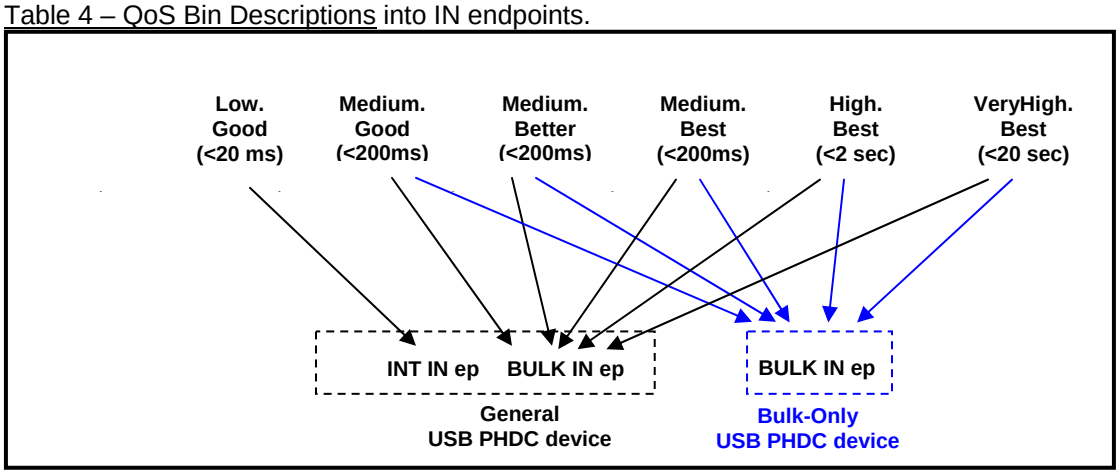

Figure 6 – Map of Application QoS Bins to USB IN Endpoint Types

All PHDC devices **shall** contain a bulk IN endpoint to be used for all reliable data transfers. PHDC devices **may** contain an interrupt IN endpoint. An interrupt IN endpoint **shall** be used if low latency/good reliability data needs to be sent. Since USB interrupt endpoints provide reliable transfers, the device **shall** be responsible for inducing data loss by flushing out stale data from the sending buffer.

For all IN endpoints, the host PHDC component **shall** either queue a read request on the endpoint or utilize the Get Data Status request (Refer to Section 07.1.), or both, to allow for device-driven communication. Note that both of these mechanisms allow silent failures to be reported at the transport level since devices that fail to respond to any Control, Bulk, or Interrupt transaction within three consecutive attempts are marked non-operational by the host controller hardware. If this occurs, an error is returned for the failed transfer, all pipes are halted, and further interaction with the device is blocked.

4.4.2 OUT Endpoints

The QoS characteristics of all data that will be sent from the USB host to device map into a Bulk OUT endpoint as shown in Figure 7 – Map of Application QoS Bins to USB OUT Endpoint Types.

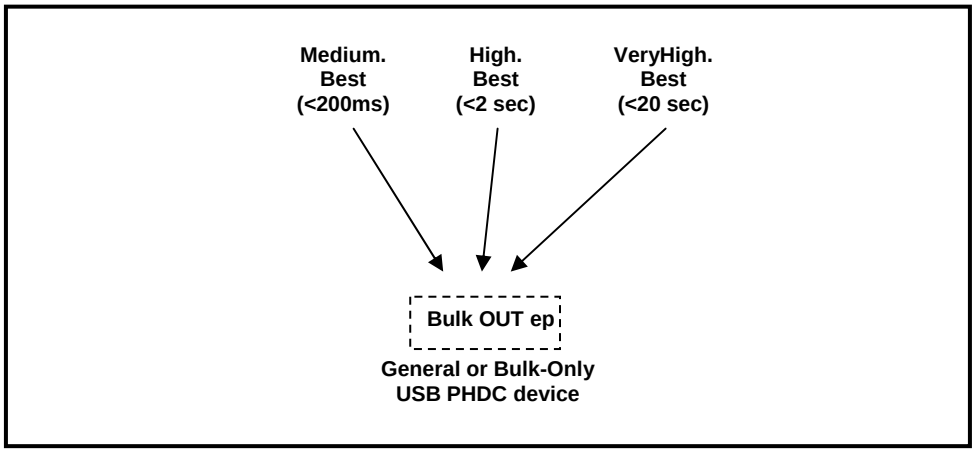

Figure 7 – Map of Application QoS Bins to USB OUT Endpoint Types

All PHDC devices shall contain a bulk OUT endpoint.

4.4.3 Meta-Data Transfer

The PHDC also defines a mechanism for transferring meta-data along with the personal healthcare data. This mechanism involves sending an initial transaction with meta-data information before transferring the data. The meta-data transaction will use short-packet semantics to provide a separation between the meta-data and the actual data. Refer to Section 6 for details.

5 Descriptors

5.1 Standard Descriptors

The following sections describe some standard USB 2.0 descriptors and give device and interface descriptor examples to show how class, subclass, and protocol values are used.

5.1.1 Device Descriptor

Table 5 – PHDC Standard USB Device Descriptor shows an example standard device descriptor for PHDC for a high speed device.

Table 5 – PHDC Standard USB Device Descriptor

Offset (decimal)	Field	Size (bytes)	Value	PHDC Value
0	bLength	1	Number	12h
1	bDescriptorType	1	Constant	01h
2	bcdUSB	2	BCD	0200h – PHDC only supports USB 2.0
4	bDeviceClass	1	Class	This **should** be set to 00h, but it **may** be set to 0Fh, the class code for PHDC. Either *bDeviceClass* or *bInterfaceClass* **shall** be set to 0Fh, the class code for PHDC.
5	bDeviceSubclass	1	SubClass	00h – The PHDC does not assign a subclass.
6	bDeviceProtocol	1	Protocol	00h – The PHDC does not assign a protocol.
7	wMaxPacketSize0	1	Number	64h –The maximum packet size for endpoint 0 on a high speed device is 64h.
8	idVendor	2	ID	Assigned by USB-IF
10	idProduct	2	ID	Assigned by manufacturer.
12	bcdDevice	2	BCD	Assigned by manufacturer.
14	iManufacturer	1	Index	Assigned by manufacturer.
15	iProduct	1	Index	Assigned by manufacturer.
16	iSerialNumber	1	Index	Assigned by manufacturer.
17	bNumConfigurations	1	Number	Typically, 01h; however, a manufacturer can implement multiple configurations so long as the PHDC requirements are upheld.

5.1.2 Configuration Descriptor

The *bNumInterfaces* field of the standard configuration descriptor **may** point to one or more interface descriptors. Refer to Section 5.1.3 for more details.

5.1.3 Interface Descriptors

Table 6 – PHDC Standard USB Interface Descriptor shows an example standard interface descriptor for PHDC.

Table 6 – PHDC Standard USB Interface Descriptor

Offset (decimal)	Field	Size (bytes)	Value	PHDC Value
0	bLength	1	Number	09h
1	bDescriptorType	1	Constant	04h
2	bInterfaceNumber	1	Number	Interface number, starting with 00h. A device **may** implement multiple interfaces (see text below).
3	bAlternateSetting	1	Number	Value for alternate setting. A device **may** implement an alternate setting so long as the PHDC requirements are upheld.
4	bNumEndpoints	1	Number	03h - A device **may** choose to implement only 2 endpoints (bulk only device), or it **may** choose to implement additional bulk or interrupt endpoints. So long as the PHDC requirements are upheld, these are valid configurations.
5	bInterfaceClass	1	Class	This **should** be set to 0Fh, the class code for PHDC. If multiple interfaces are used, it **shall** be set to 0Fh. Either *bDeviceClass* or *bInterfaceClass* **shall** be set to 0Fh, the class code for PHDC.
6	bInterfaceSubclass	1	SubClass	00h – The PHDC does not assign a subclass.
7	bInterfaceProtocol	1	Protocol	00h – The PHDC does not assign a protocol.
8	iInterface	1	Index	Assigned by manufacturer.

If a device implements multiple personal healthcare functions such as a blood pressure monitor and heart rate monitor, one of the following two methods **shall** be used for enabling multi-function devices.

1. **Multiple Interfaces** – The multi-function device **may** expose one interface per device function. The interface would define the endpoints specific to that function.
2. **Combination Device** – The multi-function device **may** implement only one interface for the multi-function device. If this option is used, the data and messaging layer must support a method for sending data for multiple personal healthcare devices through the same data channel.

5.2 Class-Defined Descriptors

Figure 8 – Personal Healthcare Class-Defined Descriptors displays the descriptors used by the PHDC. The black lines show the flow of discovery on the host, starting with the device descriptor.

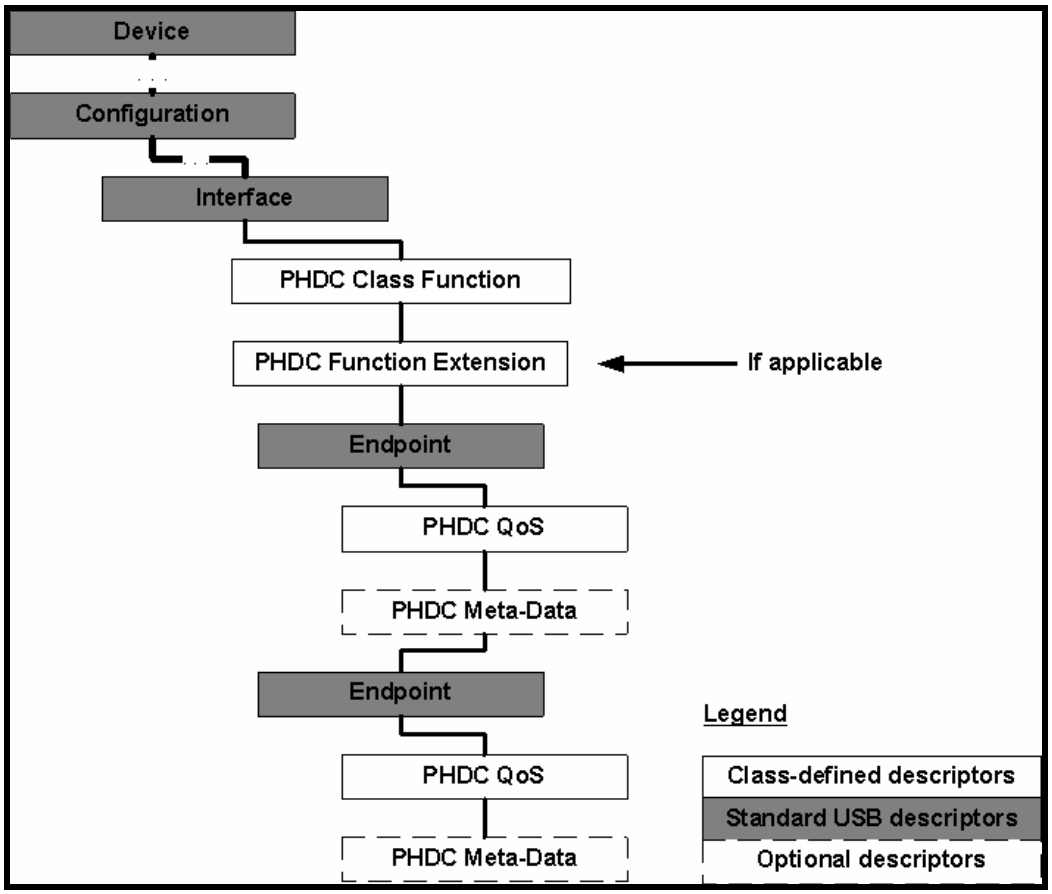

Figure 8 – Personal Healthcare Class-Defined Descriptors

The following sections describe and define the PHDC descriptors shown in Figure 8 – Personal Healthcare Class-Defined Descriptors.

5.2.1 PHDC Class Function Descriptor

The PHDC Class Function descriptor describes items and characteristics that apply at the USB function level. This descriptor specifies the data and messaging protocol that the interface supports, and the format of this descriptor **shall** be as defined below.

This descriptor **shall** follow the interface descriptor with which it is associated and precede the PHDC Function Extension descriptor and the applicable endpoint descriptors.

Table 7 – PHDC Class Function Descriptor

Offset (decimal)	Field	Size (bytes)	Value	Description
0	bLength	1	Number	Descriptor size, in bytes (includes this field)
1	bDescriptorType	1	Constant	PHDC_CLASSFUNCTION_DESCRIPTOR type
2	bPHDCDataCode	1	Constant	Data/Messaging format code, see Table 18 – Personal Healthcare Data and Messaging Formats
3	bmCapability	1	Bitmap	This field defines if certain features are implemented in the interface.

Offset (decimal)	Field	Size (bytes)	Value	Description
				Bit 0: Set to 1 if the *Meta-Data Message Preamble* feature defined in Section 6 is implemented and 0 if it is not. Bits 7..1 are reserved and must be set to 0.

The *bPHDCDataCode* field corresponds to the data and messaging protocol that is being transmitted over the PHDC. Table 18 – Personal Healthcare Data and Messaging Formats lists all available values for the data and messaging protocol and provides the information necessary for sending a given data and messaging protocol over the PHDC. If a descriptor is sent with an invalid *bPHDCDataCode*, the descriptor is invalid.

The *bmCapability* field is a bitmap that **shall** be set as defined in Table 7 – PHDC Class Function Descriptor.

5.2.2 Function Extension Descriptors

It may be necessary for a given data and messaging protocol to add an additional descriptor following the PHDC Class Function descriptor. This descriptor is titled the Function Extension descriptor. Because Function Extension descriptors are specific to a given data and messaging protocol, they are defined on a per data and messaging protocol basis in Appendix A. The Function Extension descriptor **shall** follow the PHDC Class Function descriptor and precede the applicable endpoint descriptors.

Refer to Appendix A for the Function Extension descriptor definition for each data and messaging protocol that requires one.

5.2.3 PHDC QoS Descriptor

Each endpoint descriptor **shall** be followed by a QoS descriptor that defines the latency and reliability characteristics for data being sent over that endpoint. The format of this descriptor **shall** be as defined in Table 8 – PHDC QoS Descriptor.

Table 8 – PHDC QoS Descriptor

Offset (decimal)	Field	Size (bytes)	Value	Description
0	bLength	1	Number	Descriptor size, in bytes (includes this field)
1	bDescriptorType	1	Constant	Constant PHDC_QOS_DESCRIPTOR type
2	bQoSEncodingVersion	1	Number	01h = version 1 QoS information encoding
3	bmLatencyReliability	1	Bitmap	If *bQoSEncodingVersion* = 01h, then this bitmap refers to the latency/reliability bin for the QoS data. Bitmap: Bits 6..7 are reserved and set to 0. Bit 5: VeryHigh.Best latency, reliability bin Bit 4: High.Best latency, reliability bin

Offset (decimal)	Field	Size (bytes)	Value	Description
				Bit 3: Medium.Best latency, reliability bin Bit 2: Medium.Better latency, reliability bin Bit 1: Medium.Good latency, reliability bin Bit 0: Low.Good latency, reliability bin

The *bQoSEncodingVersion* field **shall** contain the version number of the QoS information encoding. This device class defines version 01h QoS information encoding. In order to remain forward compatible, if a host implementing 01h QoS information encoding receives a *bQoSEncodingVersion* field that is not 01h, it **shall** ignore the descriptor. The host can tell the device to use the 01h QoS information encoding when it issues the SET_FEATURE FEATURE_PHDC_METADATA command (refer to Section 7.1.1).

The *bmLatencyReliability* field **shall** include a bitmap indicating the QoS values of the data that will be sent across the endpoint. If *bQoSEncodingVersion* is 01h, then the bitmap **shall** be as defined in Table 8 – PHDC QoS Descriptor. If the PHDC QoS Descriptor is for a bulk endpoint, then a combination of bits *b1-b5* corresponding to the latency and reliability values the endpoint supports **shall** be set in *bmLatencyReliability*. If the PHDC QoS Descriptor is for an interrupt endpoint, then *bmLatencyReliability* bit *b0* **shall** be set (refer to Table 9 – QoS Interrupt Endpoint Descriptor Content).

If the *Meta-Data Message Preamble* feature is not implemented *bmLatencyReliability* **shall** point to only one QoS bin, which **shall** be equal to the QoS for all data on that endpoint. Refer to Section 5.2.4 for information on the feature and Table 7 – PHDC Class Function Descriptor for information on how to indicate if the feature is implemented. This enables a host to obtain a QoS value for an endpoint when the *Meta-Data Message Preamble* feature is not implemented.

Note that all data sent over the interrupt endpoint **shall** be low latency, good reliability data. Thus, the QoS descriptor for the interrupt endpoint **shall** always be as listed in Table 9 – QoS Interrupt Endpoint Descriptor Content.

Table 9 – QoS Interrupt Endpoint Descriptor Content

Field	Value
bLength	4h
bDescriptorType	21h
bQoSEncodingVersion	01h
bmLatencyReliability	01h

5.2.4 PHDC Meta-Data Descriptor

Each PHDC QoS Descriptor **may** be followed by a PHDC MetaData Descriptor that encapsulates opaque meta-data being sent over that endpoint. This descriptor is useful for sending static meta-data or general meta-data to the host at enumeration time. The descriptor is defined such that the implementer has flexibility in the definition of the opaque data. Specifying an exact format for the opaque meta-data is out of scope of the PHDC. The format of this descriptor **shall** be as defined in Table 10 – PHDC Meta-Data Descriptor.

Table 10 – PHDC Meta-Data Descriptor

Offset (decimal)	Field	Size (bytes)	Value	Description
0	bLength	1	Number	Descriptor size, in bytes (includes this field)
1	bDescriptorType	1	Constant	Constant PHDC_METADATA_DESCRIPTOR type
2	bOpaqueData	0 to 253	Variable number	Opaque meta-data.

The *bLength* field **shall** define the size of this meta-data descriptor, based on the size of the opaque data to be sent. If no opaque data is sent and the device chooses to send the PHDC Meta-Data descriptor anyway, this descriptor size **shall** be 2h.

The *bOpaqueData* field **shall** contain the opaque meta-data to be sent to the host at enumeration time. This data could share the same format as the *bOpaqueData* field of the *Meta-Data Message Preamble* (refer to Section 6.1 for details), but it is not required to. The size of this data **should** be less than 32 bytes for best performance on all hosts.

6 Sending Meta-Data Using the PHDC

As described in Section 4.3, it may be necessary for devices and hosts to exchange meta-data such as QoS information using the PHDC. The *Meta-Data Message Preamble* feature defines a mechanism by which a devices and hosts can exchange information on the QoS required for the data being sent as well as additional meta-data which is opaque to the PHDC. This feature is **optional** for a device manufacturer to implement (refer to Table 7 – PHDC Class Function Descriptor for information on how to indicate if the feature is implemented or not). However, a host **shall** always support this feature.

The following requirements **shall** apply to Bulk OUT and Bulk IN endpoints. They **shall not** apply to Interrupt IN endpoints which always send low latency and good reliability data.

The *Meta-Data Message Preamble* feature **shall** initially be disabled. It **shall** be enabled on an interface by the host issuing the SET_FEATURE command on the FEATURE_PHDC_METADATA feature (refer to Section 7.1). When the *Meta-Data Message Preamble* feature is enabled, then all data transfers or sets of data transfers (as per the preamble *bNumTransfers* value) on the Bulk IN and OUT endpoints **shall** be preceded by a *Meta-Data Message Preamble* transfer. Refer to Section 6.4 for information on how the host can determine the QoS level of data being sent, regardless of whether the *Meta-Data Message Preamble* feature is enabled or disabled.

6.1 Meta-Data Message Preamble Transfer

During the *Meta-Data Message Preamble* transfer, a packet containing meta-data information as defined in Table 11 – Meta-Data Message Preamble **shall** be sent.

Table 11 – Meta-Data Message Preamble

Offset (decimal)	Field	Size (bytes)	Value	Description
0	aSignature	16	Constant	Constant used to give preamble verifiability. Set to 16 byte string: “PhdcQoSSignature”
16	bNumTransfers	1	Number	Count of following transfers to which the QoS setting applies
17	bQoSEncodingVersion	1	Number	01h = version 1 QoS information encoding
18	bmLatencyReliability	1	Bitmap	If *bQoSEncodingVersion* = 01h, then this bitmap refers to the latency/reliability bin for the QoS data. Bitmap: Bits 6..7 are reserved and set to 0. Bit 5: VeryHigh.Best latency, reliability bin Bit 4: High.Best latency, reliability bin Bit 3: Medium.Best latency, reliability bin Bit 2: Medium.Better latency, reliability bin Bit 1: Medium.Good latency, reliability bin

Offset (decimal)	Field	Size (bytes)	Value	Description
				Bit 0: Low.Good latency, reliability bin
19	bOpaqueDataSize	1	Number	Size, in bytes, of opaque QoS data or meta-data (if bQoSEncodingVersion = 01h).
20	bOpaqueData	0 to (MaxPacket – 21) bytes	Data	Opaque meta-data (if bQoSEncodingVersion = 01h).

The *aSignature* field is a constant. The host and device **shall** use this constant to ensure that an expected *Meta-Data Message Preamble* data packet is truly a *Meta-Data Message Preamble* data packet (refer to Section 6.3 for behavior if an expected *Meta-Data Message Preamble* is not received.). The value is **shall** be set to the 16 byte string "PhdcQoSSignature."

The *bNumTransfers* field specifies the number of transfers that follow the *Meta-Data Message Preamble*. This feature may be useful when replaying stored data. If the endpoint always sends a specific type of QoS, a large *bNumTransfers* value would ensure efficiency. *bNumTransfers* **shall** never equal zero (refer to Section 6.3 for behavior on an invalid *bNumTransfers*.).

The *bQoSEncodingVersion* field **shall** contain the version number of the QoS information encoding. This device class defines version 01h QoS information encoding.

The *bmLatencyReliability* field **shall** contain a bitmap specifying the QoS information for the transfers that follow. In this case, only one bit **shall** be set in any given *bmLatencyReliability* field (refer to Section 6.3 for behavior on an invalid *bmLatencyReliability* value.).

The *bOpaqueDataSize* field **shall** contain the number of bytes of opaque QoS or meta-data that are to be transferred in the *Meta-Data Message Preamble*.

The *bOpaqueData* field **shall** contain the opaque QoS or meta-data to be transferred. This data could have the format of the *bOpaqueData* field in the *PHDC Meta-Data descriptor* (refer to Section 5.2.4), but this is not required. The data size **shall** be *bOpaqueDataSize*. The amount of opaque data cannot exceed the endpoint maximum packet size minus 21 bytes to insure short-packet reception and request completion upon receipt of the preamble.

6.2 Data Transfer

After the *Meta-Data Message Preamble* transfer, a count of *bNumTransfers* data transfers **shall** be sent. After that, when a new transfer is desired, a new *Meta-Data Message Preamble* **shall** be sent.

6.3 Meta-Data Message Preamble Feature Error Conditions

This section lists mechanisms for dealing with error conditions that occur when the *Meta-Data Message Preamble* feature is enabled.
1) If a *Meta-Data Message Preamble* is expected but not received on the device or it contains an invalid *bmLatencyReliability* value or *bNumTransfers* value the device **shall** STALL subsequent transactions to the receiving endpoint until the host clears the stall condition.

2) If a *Meta-Data Message Preamble* is expected but not received on the host or contains an invalid *bmLatencyReliability* value or *bNumTransfers* value, the host **shall** issue a SET_FEATURE ENDPOINT_HALT request to the device.

After the actions described in condition 1) or condition 2) above occur the host **shall** clear the halt by issuing a CLEAR_FEATURE ENDPOINT_HALT request to the device.

After the host has issued the CLEAR_FEATURE ENDPOINT_HALT, the message sender **shall** send a *Meta-Data Message Preamble* transfer, and the receiver **shall** expect to receive a *Meta-Data Message Preamble* transfer.

This error handling process for *Meta-Data Message Preambles* is displayed in Figure 9 – QoS Error Handling Process.

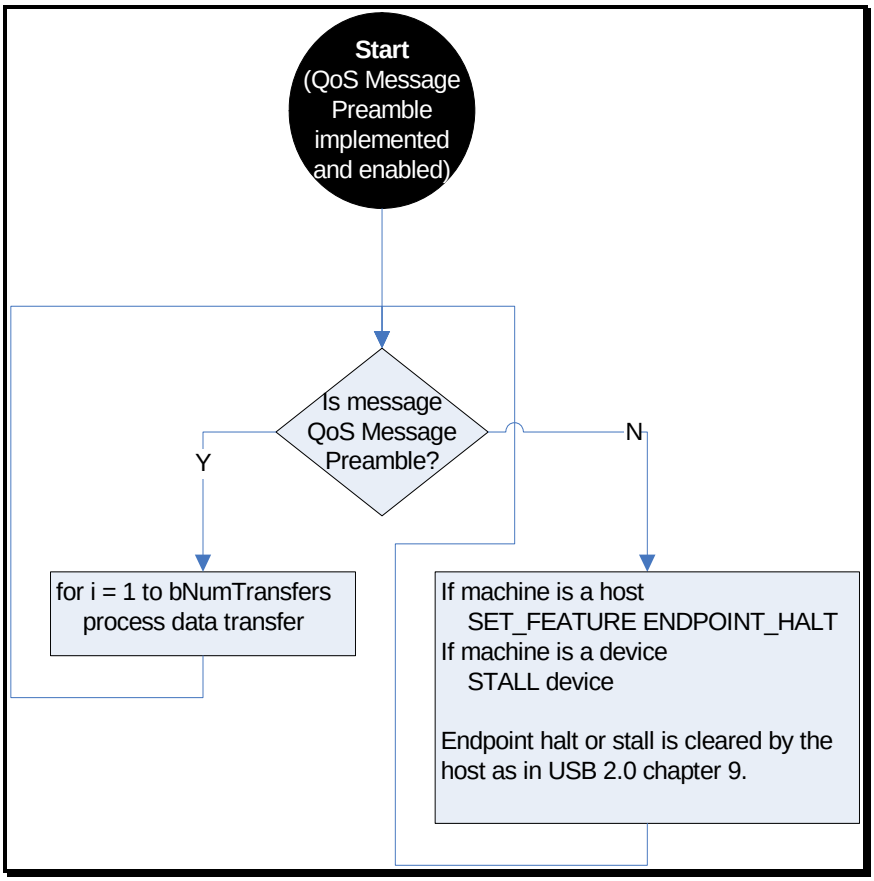

Figure 9 – QoS Error Handling Process

6.4 Determining QoS

It may be necessary for an application on the host side to determine the QoS of the data being sent. Figure 10 – Determining QoS at Host illustrates how QoS information is provided using the *Meta-Data Message Preamble* feature or the PHDC QoS descriptor. The only time QoS information is not available occurs when the *Meta-Data Message Preamble* feature is implemented, but not enabled.

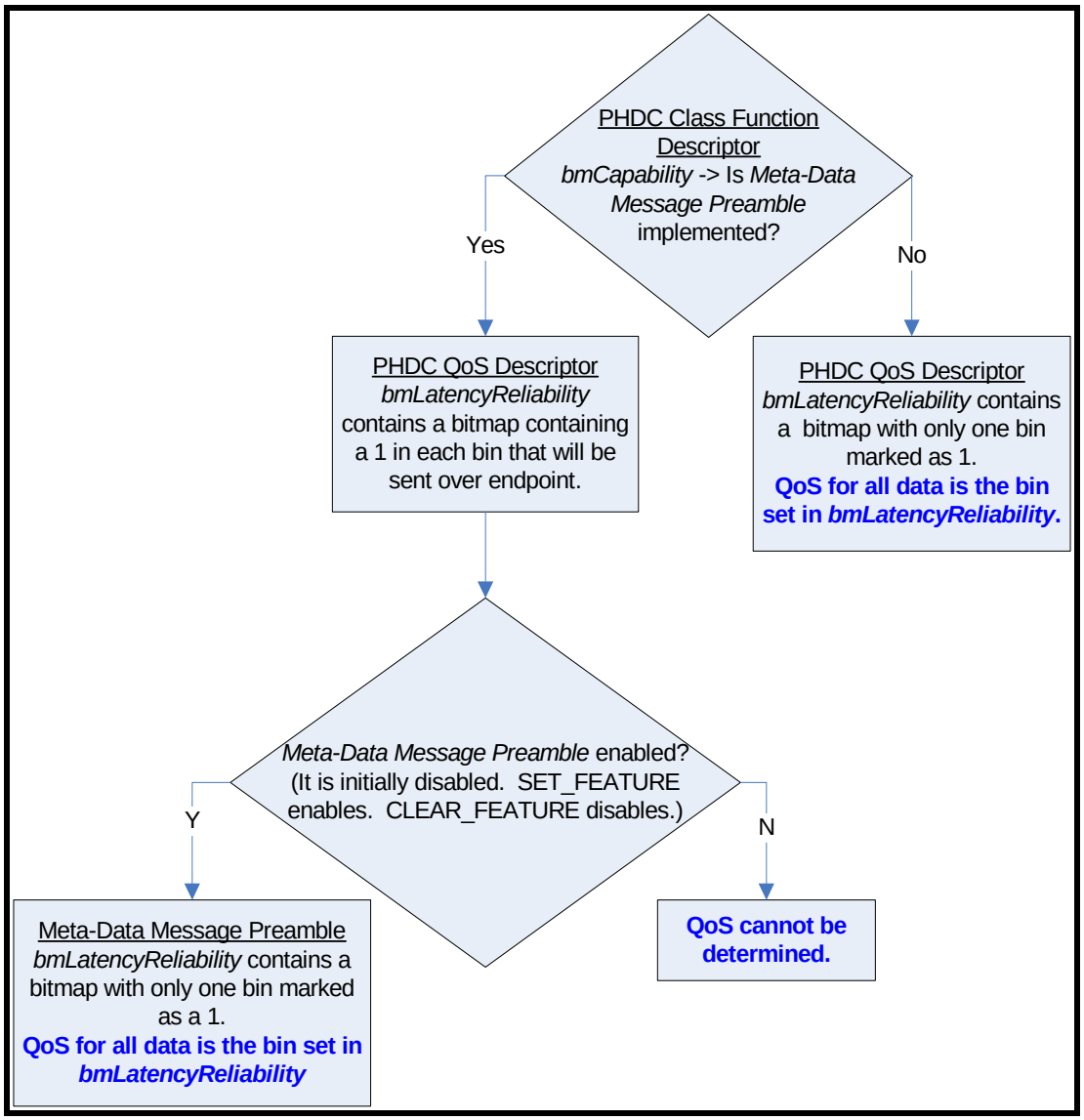

Figure 10 – Determining QoS at Host

7 Requests

7.1 Class-Specific Requests

7.1.1 Set/Clear Meta-Data Message Preamble Feature

The *Meta-Data Message Preamble* feature **shall** be turned on and off via class-specific extensions to the SET_FEATURE and CLEAR_FEATURE requests, respectively.

If the device does not support the *Meta-Data Message Preamble* feature, then it **shall** respond to these requests with a stall, and the host **shall** clear it.

The *Meta-Data Message Preamble* feature **shall** be off by default and **shall** require the SET_FEATURE command shown in Table 12 – Set/Clear PHDC Meta-Data Message Preamble Feature Requests to turn it on.

The host **should** ensure there are no pending transactions on the device before issuing a SET_FEATURE or CLEAR_FEATURE (FEATURE_PHDC_METADATA) command to avoid synchronization issues with the preamble. A device is not expected to modify transactions already queued to hardware. The host can use the class-defined GET_STATUS request (refer to Section 7.1.2) to determine if the device endpoints are idle.

The syntax for these requests **shall** be implemented as shown in Table 12 – Set/Clear PHDC Meta-Data Message Preamble Feature Requests.

Table 12 – Set/Clear *PHDC Meta-Data Message Preamble* Feature Requests

bmRequestType	bRequest	wValue	wIndex	wLength
00100001B	SET_FEATURE	Bits 15..8: *bQoSEncodingVersion* value desired for *Meta-Data Message Preamble* transfers. Bits 7..0: FEATURE_PHDC_METADATA	Target Interface	0
00100001B	CLEAR_FEATURE	Bits 15..8: Reserved, must be zero. Bits 7..0: FEATURE_PHDC_METADATA	Target Interface	0

The high-order byte of *wValue* defines the *bQoSEncodingVersion* version that the host expects to use to encode and decode *Meta-Data Message Preamble* messages (refer to Sections 5.2.3 and 6.1 for a description of the *bQoSEncodingVersion*). The device and host **shall** encode and decode *Meta-Data Message Preamble* messages using this *bQoSEncodingVersion*. This enables forward compatibility. If the host only supports a *bQoSEncodingVersion* of 01h, it **shall** use the SET_FEATURE command to enforce this. If the device only supports a *bQoSEncodingVersion* of 01h, the host will know this from the *PHDC QoS Descriptor* (refer to Section 5.2.3), and the host **shall** use the SET_FEATURE command to enforce a *bQoSEncodingVersion* of 01h.

The *wIndex* value **shall** specify the target interface for the request. If either feature request specifies an interface that doesn't exist, the device **shall** respond with a Request Error.

If *wLength* is non-zero, then the device behavior is not specified.

Default State: Device behavior when these requests are received while the device is in the Default state is not specified.

Address State: Device behavior when these requests are received while the device is in the Addressed state is not specified.

Configured State: These requests are valid when the device is in the Configured state.

Upon receipt of the SET_FEATURE (FEATURE_PHDC_METADATA), the device **shall** turn on the *Meta-Data Message Preamble* feature. Upon receipt of the CLEAR_FEATURE (FEATURE_PHDC_METADATA), the device **shall** turn off the *Meta-Data Message Preamble* feature. Upon the completion of a successful status phase for the set or clear operation, the device **shall** be in the state specified.

7.1.2 Get Data Status

An application may choose to use the Get DataStatus request to determine which endpoints on the device have data. This is useful to do before issuing a SET_FEATURE or CLEAR_FEATURE (FEATURE_PHDC_METADATA) request since these commands **should not** be issued if there are pending transactions on the device (refer to Section 7.1.1). This request is also useful for devices that may submit initial requests upon reaching Configured state. Devices and hosts **shall** support the class-defined Get Data Status request in Table 13 – Get Data Status.

Table 13 – Get Data Status

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_STATUS	0	Target Interface	2	Endpoint bitmap, see Table 14

When a device receives the request, it **shall** return a bitmap corresponding to the endpoints holding data on the device as described in Table 14 – Information Returned from a PHDC Class-defined Get Data Status Request.

Table 14 – Information Returned from a PHDC Class-defined Get Data Status Request

D7	D6	D5	D4	D3	D2	D1	D0
1 if endpoint 07h has data; 0 if not	1 if endpoint 06h has data; 0 if not	1 if endpoint 05h has data; 0 if not	1 if endpoint 04h has data; 0 if not	1 if endpoint 03h has data; 0 if not	1 if endpoint 02h has data; 0 if not	1 if endpoint 01h has data; 0 if not	Reserved, must be zero EP0 not tracked
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
1 if endpoint 0Fh has data; 0 if not	1 if endpoint 0Eh has data; 0 if not	1 if endpoint 0Dh has data; 0 if not	1 if endpoint 0Ch has data; 0 if not	1 if endpoint 0Bh has data; 0 if not	1 if endpoint 0Ah has data; 0 if not	1 if endpoint 09h has data; 0 if not	1 if endpoint 08h has data; 0 if not

The *wIndex* value **shall** specify the target interface for the request. If the request specifies an interface that doesn't exist, the device **shall** respond with a Request Error.

If *wLength* is non-zero, then the device behavior 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: Device behavior when this request is received while the device is in the Addressed state is not specified.

Configured State: This request is valid when the device is in the Configured state.

8 Descriptor, Request & Feature Types Table

The constants for the descriptor types defined in this document are defined in Table 15 – Descriptor Types.

Table 15 – Descriptor Types

Descriptor Type	Value
PHDC_CLASSFUNCTION_DESCRIPTOR	20h
PHDC_QOS_DESCRIPTOR	21h
PHDC_METADATA_DESCRIPTOR	22h

The constants for class-defined request types defined in this document are defined in Table 16 – Request Types.

Table 16 – Request Types

Descriptor Type	Value
GET_STATUS	00h
CLEAR_FEATURE	01h
SET_FEATURE	03h

The constants for feature types are defined in Table 17 – Feature Types. Note that, per USB 2.0 conventions, the feature value **shall** be in the low-order byte of the word.

Table 17 – Feature Types

Descriptor Type	Value
FEATURE_PHDC_QOS	01h

Appendix A

1 Introduction

The USB Personal Healthcare Device Class (PHDC) defines a generic mechanism for sending personal healthcare data. Appendix A defines specific descriptors and additional information that is needed for sending data and messages defined in specific standards using the PHDC.

This Appendix is a normative Appendix to the USB PHDC specification.

2 PHDC Class Function Descriptor Data/Messaging Codes

The PHDC Class Function descriptor *bPHDCDataCode* field points to the data and messaging standard a device expects to use for personal healthcare data. Table 18 – Personal Healthcare Data and Messaging Formats lists the data and messaging standards currently supported by the PHDC.

Table 18 – Personal Healthcare Data and Messaging Formats

Data Format	Value	Description
PHDC_VENDOR	01h	Vendor defined standard
PHDC_11073_20601	02h	ISO/IEEE 11073-20601 Optimized Exchange Protocol™, plus any specifications that utilize this specification as a base

The following sections in this Appendix further define the class-specific extensions, such as the Function Extension descriptor, for each data and messaging standard.

3 Vendor-Defined Extensions

If a device manufacturer decides to send a vendor-defined data and messaging format over the PHDC, then *bPHDCDataCode* **shall** be equal to PHDC_VENDOR. In this case, the vendor **may** define any number of vendor-specific descriptors or other functionality and **shall** enable support for these features in vendor-specific host drivers.

4 11073 PHD Specific Extensions

This section is specific to the PHDC_11073_20601 (aka, 11073 PHD) data and messaging standard and lists extensions to the PHDC, such as additional required descriptors, as well as additional informative text on creating an 11073 PHD extension. Devices and hosts that implement 11073 PHD data and messages **shall** support these extensions. A device or host that does not implement 11073 PHD data and messages is not required to support these extensions.

If a device implements the 11073 PHD extensions, then *bPHDCDataCode* **shall** be equal to PHDC_11073_20601.

The PHDC 11073 PHD Function Extension Descriptor **shall** follow the PHDC Class Function descriptor and precede any endpoint descriptors. This descriptor is a variable length descriptor that provides information on the device specializations that the personal healthcare device supports. The format for this descriptor **shall** be as defined in Table 19 – PHDC 11073 PHD Function Extension Descriptor.

Table 19 – PHDC 11073 PHD Function Extension Descriptor

Offset (decimal)	Field	Size (bytes)	Value	Description
0	bLength	1	Variable number	Descriptor size, in bytes (includes this field)
1	bDescriptorType	1	Constant	Constant PHDC_11073PHD_FUNCTION_DESCRIPTOR type
2	bReserved	1	Zero	Reserved for future use, must be zero
3	bNumDevSpecs	1	Number	Number of *wDevSpecializations* below
4	wDevSpecializations	2 * n	Words	Variable length list that defines the device specialization. The device specialization is informatively contained in the Nomenclature Codes Annex of ISO/IEEE 11073-20601™. The specific codes to use are those listed in the "From Communication Infrastructure (MDC_PART_INFRA)" section. These codes normatively reference the ISO/IEEE 11073-10101™ codes in the Communication Infrastructure Annex for Device Specializations of ISO/IEEE 11073-10101™. In the event of a discrepancy, the ISO/IEEE 11073-10101™ codes **shall** be used.

The *bLength* field lists the length of the variable-sized descriptor. If the *bLength* value is not an even number, greater than or equal to 4, then it is invalid.

The *bNumDevSpecs* field defines the number of device specializations that the interface supports. An interface that implements a single device specialization **shall** have a *bNumDevSpecs* of 1. An interface that implements more than one device specialization **shall** have a *bNumDevSpecs* equal to the number of device specializations that the interface supports. If the device does not support any ISO/IEEE 11073-10101™ defined device specializations, the descriptor **shall** have a *bNumDevSpecs* equal to 0.

The *wDevSpecializations* field is a Word or Words that contain the little endian 16-bit value corresponding to the device specialization(s) supported by the device. These values **shall** be as defined in the ISO/IEEE 11073-20601™ PHD specifications, Nomenclature Codes Annex for device Specializations with the caveat that, if there is a discrepancy between ISO/IEEE 11073-20601™ and ISO/IEEE 11073-10101™ then ISO/IEEE 11073-10101™ **shall** be used. If *bNumDevSpecs* is 0, then there **shall** be no *wDevSpecializations* fields.

5 Descriptor Constants

Table 20 – Appendix A Descriptor Constants lists descriptor constants used in this Appendix.

Table 20 – Appendix A Descriptor Constants

Descriptor Type	Value
PHDC_11073PHD_FUNCTION _DESCRIPTOR	30h