Universal Serial Bus
CDC Subclass Specification for **Wireless Mobile Communications Devices**

Revision 1.1

February 9, 2007

Revision History

Rev	Date	Filename	Comments
1.1	2/9/07	WMC110.doc	Final edits as per February 2007 CDC meeting Specify a reserved value in Table 5-3. Remove “For Review and Discussion Only” footer
1.0	11/23/2001	CDC_WMC10.doc	Remove “For Review and Discussion Only” footer; replace “TBD” placeholders with numeric values.

Please send comments via electronic mail to cdc@usb.org

Contributors to Version 1.0

Mobile Computing Promotion Consortium (MCPC)

Tatsuya Nakatani	Fujitsu Limited
Kenji Oguma	NEC Corporation
Shinji Kamiya	DENSO Corporation
Keiichi Murakami	Fujitsu Limited
Paul E. Berg	MCCI
Terry Moore	MCCI
Minoru Ohwada	MITSUBISHI ELECTRIC CORPORATION
Noriko Fukumasu	NEC Corporation
Noriko Norimatsu	NEC Corporation
Hisayuki Yamanaka	NOKIA Japan Co. Ltd
Chang-Jiang Zhang	NOKIA Japan Co. Ltd
Seiji Abe	NTT DoCoMo, Inc.
Hideo Uchizono	NTT DoCoMo, Inc.
Toshiharu Uchida	Pioneer Corporation
Hajime Shimizu	SHARP BUSINESS COMPUTER SOFTWARE INC.
Kazuhito Yasue	SHARP BUSINESS COMPUTER SOFTWARE INC.
Yoshizane Tanaka	SUNCORPORATION
Takafumi Ito	Toshiba Corporation
Hiroshi Kinugasa	Toshiba Corporation

USB Device Working Group Communications Device Committee

Jim Wilcox	Apple Computer Inc.
Russ Winsper	Apple Computer Inc.
Elena Neira	Ericsson
Morten Christiansen	Ericsson Semafor
Joel Silverman	KLSI
Eric Overtoom	Motorola
Ryota Okazaki	NEC Corporation
Katsuhiko Kobayashi	NEC Engineering, Ltd.
Richard Petrie	Nokia Mobile Phones
James Scales	Nokia Mobile Phones
Julie Bendig	Qualcomm
Hongshi Guo	Qualcomm

Contributors to Version 1.1

USB Device Working Group Communications Device Committee

Russ Winsper	Apple Computer Inc.
Bruce Balden	Belcarra
Jun Guo	Broadcom
Morten Christiansen	Ericsson AB
Alan Berkama	HP
Joel Silverman	K-Micro
Brian Meads	MCCI
Joe Decuir	MCCI
Peter FitzRandolph	MCCI
Terry Moore	MCCI
Ken Taylor	Motorola
Richard Petrie (editor)	Nokia
Janne Rand	Nokia
Tero Soukko	Nokia
Dale Self	Symbian
John Turner	Symbian
Saleem Mohammad	Synopsys

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

The USB interface uses a common and simple cable connector, which supports the connection of various types of equipment. When used with wireless communications devices, USB is a vital interface that handles various media over one type of cable using a unified protocol.

Mobile (cellular phone) wireless service technology is normally characterized by generation:

- First generation (or 1G) services are analog, and provides no special data services.
- Second generation (or 2G) services use TDMA or CDMA, and are loosely characterized as "digital" services. Only limited data service is available on these services. The leading 2G service in Europe is GSM; TDMA and CDMA are common in other geographical locations
- Updated second generation (or 2.5G) services build on the digital services by adding protocols that are more suitable for data. They provide bandwidth of up to about 144 Kbps. The best known 2.5G service is GPRS, based on GSM. 2.5G services typically provide no additional multimedia services, and are based on the existing spectrum and towers.
- Third generation (or 3G) services are a major upgrade to the services offered by 2G equipment.

With the advent of third generation services, various kinds of computing equipment can be connected to wireless communications devices via USB. (This computing equipment is generically referred to as "Mobile Terminal Equipment"). The capabilities of third-generation mobile equipment are much greater than those of second-generation equipment. Multiple calls can be handled on a single communications terminal. "Multimedia communication" (including Internet sessions, voice calls, and special purpose data services such as fax and modem emulation) will take place at the high speeds provided by the third generation wireless networks. To support these services, this document indicates one way of providing USB connectivity for terminal equipment containing multiple functions, for example audio, data communications, and status monitoring functions.

1.2 Scope

This document specifies new device subclasses intended for use with Wireless Mobile Communications devices, based on [USBCDC1.2], [USBPSTN1.2] and [USBECM1.2].

Because this specification is based on several CDC based specifications, the question arises of how to resolve conflicts. The intention of this specification is that all material presented here be upwards-compatible extensions of these other specifications. In cases, where information is repeated from other documents, the originating document shall be treated as the controlling document. New numeric codes for subclass codes, protocol codes, management elements, and notification elements are defined in [USBCDC1.2] and are repeated in this document for clarity.

1.3 Related Documents

Reference	Description
[3GPP27.007]	AT command set for User Equipment (UE), 3rd Generation Partnership Project; Technical Specification Group Terminals, Document 27.007,Version 3.9.0 (June 2001). Available on-line at http://www.3gpp.org/ftp/Specs/2001-06/R1999/27_series/27007-390.zip.
[Bluetooth Reserved Numbers]	The reserved numbers section of official Bluetooth website **www.bluetooth.org**.
[C-S0017-0]	3GPP2 TSG.C Specification C-S0017-0. Specifies AT command sets to be used for cmda2000 mobile terminals. Available on-line from http://www.3gpp2.org/public_html/specs/tsgc.cfm .
[GSM07.07]	ETSI GTS GSM 07.07 V5.0.0 (1996-07) Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment (ME) (GSM 07.07), ETSI. Available on-line at http://www.etsi.org/.
[INTTD]	*USB Telephony Devices: Interfaces for Value-Add Feature Set,* Version 1.00, June 19,2000, Intel Corporation.
[LEACH1998]	*UUIDs and GUIDs,* Paul J. Leach et al, IETF draft draft-leach-uuids-guids-01.txt., February 4, 1998.
[MICBUS]	*Guidelines for Bus and Device Specifications* , Version 1.0a, March 29, 2000, Microsoft Corporation. This specification is available online from Microsoft, http://www.microsoft.com/whdc/archive/specguide.mspx
[MICMFD]	*Designing Multifunction Devices for Windows Operating Systems*, Version 1.0, April 25, 2000, Microsoft Corporation. This specification is available online from Microsoft, http://www.microsoft.com/whdc/device/mf/mfdesin.mspx.
[OBEX1.2]	*IrDA Object Exchange Protocol IrOBEX*, V1.3. This specification is available online from the website **http://www.irda.org**.
[OPENC309]	DCE: Remote Procedure Call, Open Group CAE Specification C309 ISBN 1-85912-041-5 28cm. 674p. pbk. 1,655g. 8/94
[PCCA101]	PCCA STD-101, Data Transmission Systems and Equipment - Serial Asynchronous Automatic Dialing and Control for Character Mode DCE on Wireless Data Services, Portable Computer and Communications Association. Available on-line at http://www.pcca.org/standards.[1] Annexes to this document are:
[PCCA101-A]	common commands, in main document (== TIA/EIA-678 Annex A)
[PCCA101-D]	*Pad Control*, http://www.pcca.org/standards/Annex_d.doc, (== TIA/EIA-678 Annex D)
[PCCA101-F]	*Commands for wireless networks*, http://www.pcca.org/standards/Annex_f.doc (== TIA/EIA-678 Annex B)
[PCCA101-I]	*Commands for Analog Cell Phones*, http://www.pcca.org/standards/Annex_i.doc (== TIA/EIA-678 Annex C)
[PCCA101-L]	*Commands for CDPD modems*, http://www.pcca.org/standards/AnnexI20.PDF (not in TIA/EIA-678)
[PCCA101-O]	*Commands for Wakeup control*, http://www.pcca.org/standards/Annex_o_ballot_version.doc (not in TIA/EIA-678)
[USB2.0]	*Universal Serial Bus Specification*, revision 2.0 (also referred to as the *USB Specification*). This specification is available on the World Wide Web site **http://www.usb.org**.
[USBAUD1.0]	*Universal Serial Bus Device Class Definition for Audio Devices,* Release 1.0. This specification is available on the World Wide Web site **http://www.usb.org**.
[USBCDC1.2]	*Universal Serial Bus Class Definitions for Communications Devices*, Version 1.2. This specification is available on the World Wide Web site **http://www.usb.org**.
[USBDFU1.0]	*Universal Serial Bus Device Class Specification for Device Firmware Upgrade,* Release 1.0. This specification is available on the World Wide Web site **http://www.usb.org**.
[USBHID1.1]	*Universal Serial Bus Device Class Definition for Human Interface Devices,* Version 1.1. This specification is available on the World Wide Web site **http://www.usb.org**.
[USBECM1.2]	Universal Serial Bus USB CDC Subclass Specification for Ethernet Control Model Devices, Release 1.2. This specification is available on the World Wide Web site **http://www.usb.org**.
[USBMASS1.1]	*Universal Serial Bus Mass Storage Class Specification Overview,* Release 1.1. This specification is available on the World Wide Web site **http://www.usb.org**.

[1] PCCA STD-101 was adopted as TIA/EIA-678. The content is identical but has been rearranged.

[USBPSTN1.2] Universal Serial Bus USB CDC Subclass Specification for PSTN Devices, Release 1.2. This specification is available on the World Wide Web site **http://www.usb.org**.

1.4 Terms and Abbreviations

Term	Description
2G	Second generation wireless telecommunication systems
2.5G	Second generation wireless telecommunication systems, with enhanced data services.
3G	Third generation wireless telecommunication systems
3GPP	Third Generation Partnership Project: trade organization planning the migration from GSM to 3G service. (See http://www.3gpp.org/)
3GPP2	Third Generation Partnership Project Two: trade organization planning the migration from CDMA to 3G service. (See http://www.3gpp2.org/).
ACM	Abstract Control Model, a way of representing data/fax modem capabilities in USB devices, defined by [USBCDC1.2].
ARIB	Association of Radio Industries and Businesses
BARB	Bluetooth Architecture Review Board
CDMA	Code Division Multiple Access (see http://www.cdg.org/).
cdmaOne	cdmaOne describes a complete wireless system that incorporates the CDMA/IS-95 air interface.
cdma2000	cdma2000 is Telecommunications Industry Association (TIA) standard for third-generation technology that is an evolutionary outgrowth of cdmaOne. See 3GPP2.
control plane	Communications Class interface used to perform device management and optionally call management. See [USBCDC1.2].
DFU	Device Firmware Upgrade, a standard means of updating firmware over USB. See [USBDFU1.0].
EDGE	Enhanced Data rates for GSM and TDMA/136 Evolution – an enhanced version of GPRS compatible with GSM and TDMA/136
GOEP	Generic Object Exchange Profile
GPRS	General Packet Radio Services, a 2.5G data service for GSM networks
GSM	Global System for Mobile telecommunications, a complete 2G wireless system. See http://www.gsmworld.com/

Term	Description
IMT-2000	International Mobile Telecommunication-2000 – the generic standard name for 3G technology. See http://www.itu.org/imt, and especially http://www.itu.int/imt/what_is/roadto/index.html, which gives a very good overview of how cellular technologies align.
IrDA	Infrared Data Association. (http://www.irda.org/)
MCPC	Mobile Computing Promotion Consortium, http://www.mcpc-jp.org/.
MDLM	Mobile Direct Line Model, a way of migrating some of the protocol functions of wireless terminal adapters to the USB host system. A mechanism for implementing MDLM transport is defined by this document.
ME	Mobile Equipment
MT	Mobile Terminal
OBEX	Object Exchange
PABX	Private Automatic Branch eXchange
PDC	Personal Digital Cellular
PHS	Personal Handy-phone System
TA	Terminal Adapter
TDMA	Time Division Multiple Access, a 2G modulation system
TDMA/136	A complete wireless service based on the TDMA/IS-136 air interface. See http://www.uwcc.org/.
TE	Terminal Equipment, it indicates the communications equipment in the USB host
UUID	Universal Unique Identifier
WHCM	Wireless Handset Control Model
WMC	Wireless Mobile Communications, the abbreviation for this device subclass

2 Management Overview

With 2.5G and 3G cell phone services, manufacturers and network operators need to define a common way for handling multiple data and voice services in a single handset.

This specification represents multi-function communications handset devices as composite devices. Each potential facility is modeled, if possible, using a pre-existing device class. This allows reuse of existing class drivers. Additional Communications Device subclasses are defined for new application areas specific to wireless handsets.

Although some of the capabilities can only be used after a circuit is established, each capability is always explicitly represented in the descriptors. This allows for a simpler usage model for current operating systems.

Because the 3G standards and existing cell phones use AT commands for computer-directed call control, this specification extends the Telephone Control Model by adding a new protocol code that indicates that AT commands are used for call control.

Table 2-1 illustrates the representations. Note that in Table 2-1, the notation "pp" is used to indicate that the protocol code is one of the protocol codes which identifies variants of the AT command set.

Table 2-1: Sample Function Representation

Function	Representation	Endpoints
voice (AT-command call control)	Telephone Control Model (TCM) with AT commands (02-03-*pp*) + Audio	Interrupt IN + audio stream ISO OUT + audio stream ISO IN
voice (AT-command call control, plus ability to use handset's keypad in host, and to use handset's speaker and microphone as host peripherals)	Telephone Control Model (TCM) with AT commands (02-03-*pp*) + Audio + HID	TCM Interrupt IN + audio stream ISO OUT (to network) + audio stream ISO IN (from network) + audio stream ISO OUT (to headset earpiece) + audio stream ISO IN (from headset microphone + HID Interrupt IN
fax/modem	ACM (02-02-*pp*) + Data	Interrupt IN + Data Class Bulk OUT & Bulk IN
OBEX	OBEX (02-0Bh0B-00), + Data Class	Data Class Bulk OUT & Bulk IN
Ethernet Frame (fixed service)	CDC Ethernet Networking Control Model (ECM) (02-06-00) + Data	Interrupt IN + Data Class Bulk OUT & Bulk IN
Device Management	Device Management (02-09h09-*pp*)	Interrupt IN

The remainder of this specification is organized into the following outline.

- Assumptions and Constraints
- Functional Overview
- New class code values
- Functional Characteristics for each subclass, including:
    1. Functional Topology

2. Descriptors
3. Management Elements
4. Notifications

- Device Requests
- Device Descriptors

3 Assumptions and Constraints

This standard assumes the reference model shown in Figure 3-1.

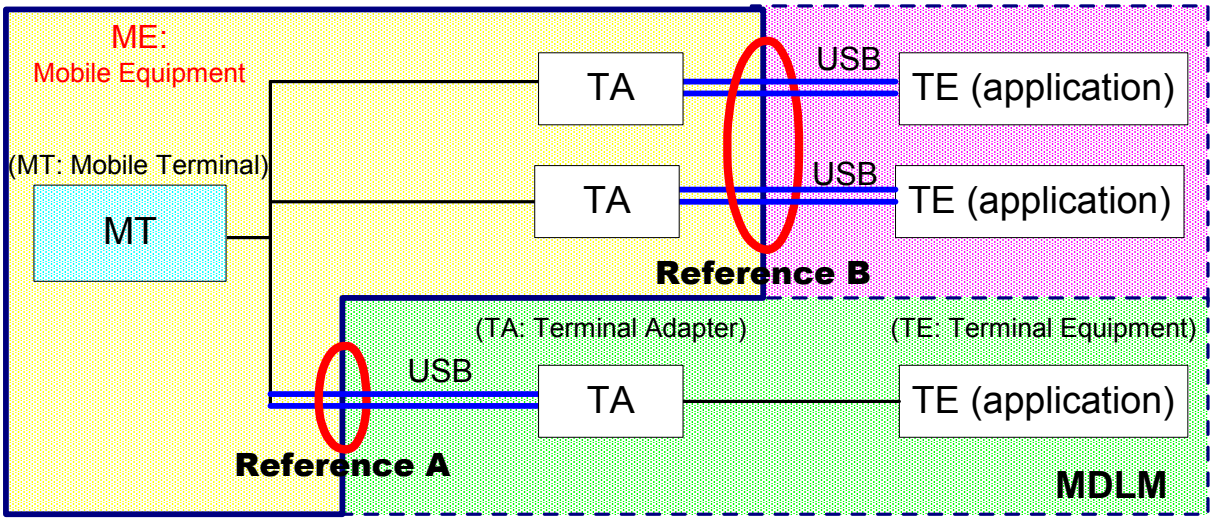


Figure 3-1: Reference Model

In this figure, the handset (or Mobile Equipment, "ME") consists of an MT and one or more TAs. The Mobile Terminal (MT) has only a facility to access the air interface. It can be thought of as the radio transmitter and receiver. Each Terminal Adapter (TA) is provided with one call facility or management capability. If a TA is provided with a call facility, that means it provides the data formatting, media conversions, and protocol capabilities to make that kind of call, to the USB host at Reference B. The presence of a TA reflects an abstract capability of the ME; at any given time, it might not be possible to use the TA to make that kind of call, due to external circumstances.

Within the limits of USB, any number of TAs may be provided. Applications use the kind of TA that provides the device model that meets their requirements. For a Voice Application, the application opens a Telephone Control Model TA and uses USB Audio. For a modem application, the application accesses an Abstract Control Model TA. When multiple applications need to be operated at the same time (i.e., Multi-call), the ME must have sufficiently many TAs to support all the applications or calls concurrently. Of course, the ME and the network interface must also be able to support such concurrency; otherwise, only a subset of the calls can proceed.

The ME can also provide connections to the USB host at Reference A. In this case, some or all of the TA functions can be off-loaded from the ME to the USB host. However, due to the variety of standards and requirements, this specification cannot cover the details of this interface. Instead, this specification provides a standardized way for operators or regional groups to agree on a private interface, while still making it possible for cognizant third-party software to recognize and operate conforming devices.

3.1 Compliance

In addition to meeting all the requirements of the USB core specification [USB2.0], devices conforming to this specification must meet the following requirements. Some of these requirements are testable; others must be enforced by design.

The testable requirements focus on the content of the descriptors.

1. The Device descriptor shall have bDeviceClass set to Communications Class, and bDeviceSubclass and bDeviceProtocol set to zero.

2. Each Communications Class interface must be followed by the appropriate functional descriptors for that interface. These shall include the HEADER functional descriptor.

3. If a Communications Class interface has associated Data Class interface, a Union Functional descriptor must appear, specifying the Communications Class interface as the primary interface, and the Data Class interface(s) as subordinate interfaces.

4. No Data Class interface should appear that is not mentioned in a Union Functional descriptor.

5. Each Data Class interface shall be mentioned in exactly two Union Functional descriptors. It shall be mentioned in the Union Functional descriptor that follows the primary Communications interface for the function to which it belongs. It shall also be mentioned in the Union Functional descriptor for the WHCM interface that describes the handset to which the Data Class interface belongs.

6. If a Communications Class interface appears with multiple alternate settings, all alternate settings for that interface must have the same bInterfaceClass, bInterfaceSubclass and bInterfaceProtocol codes.

7. The class descriptors associated with a given Communications Class interface must appear sequentially in the configuration bundle *after* the interface descriptor to which they apply, and *before* the next interface descriptor in the bundle (even if that interface descriptor is for an alternate setting for the same interface.

8. If a Communications Class interface appears with multiple alternate settings, functional descriptors must be repeated after each such interface and the Union Functional descriptors must be the same for each alternate setting.

9. All Abstract Control Model and Device Management Communications Class interfaces must have an INTERRUPT IN endpoint for transporting notifications to the host.

10. For other interfaces, the interrupt endpoint requirements given in section 4.3 must be met.

11. Bit zero of the first octet of any Ethernet address associated with the device must always be zero. (See section 6.4.2.3.)

Non-testable requirements

1. The device shall handle contention according to the rules presented section 4.2.1.

2. Suppose that SendEncapsulatedCommand may optionally be omitted for a given kind of function (according to the controlling specification). Further suppose that the function, in fact, implements it. In such cases, an Interrupt IN endpoint must be supplied with the controlling Communications interface, for transporting *ResponseAvailable* notifications.

3. The first three octets of any Ethernet address associated with the device shall be the OUI of the organization assigning the Ethernet address, and the last three octets shall be assigned in such a way as to guarantee uniqueness and consistency.

The intent of this specification is that certain handset functions be represented in certain ways. However, this specification is not restrictive. The class-compliant functions of a device are compliant with the governing class specification (and in accordance with the general requirements given above).

4 Functional Overview

4.1 Device Organization

A Communications Device Class Wireless Handset consists of one or more functions (where "function" is used in the sense of multi-function device). The device designer organizes functions within a device into groups. Figure 4-1 shows an example of how functions might be grouped. The example device is modeled as consisting of a logical handset, a mass storage device, and a device firmware update function. In turn, the handset is composed of Modem, LAN, OBEX, HID (for the keypad), Audio, and Device Status functions.

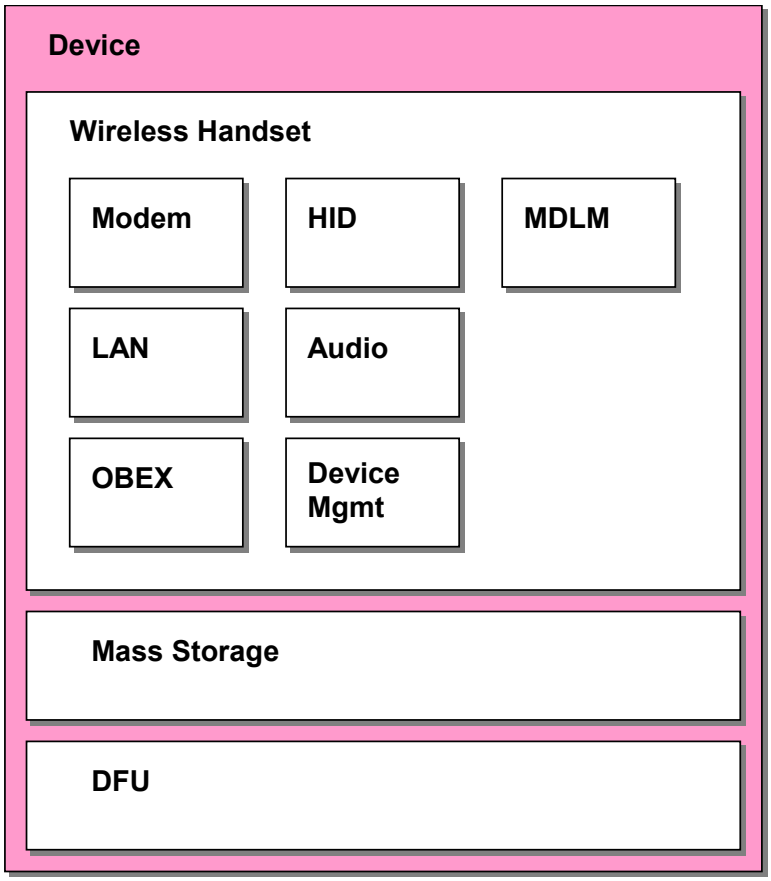


Figure 4-1. Device Structure

The device designer expresses these relationships in the descriptors for the device. Each of the individual functions is modeled using one or more interfaces of the appropriate device classes. In the device shown in Figure 4-1, for example, the functions are represented in the descriptors as follows:

- **Modem**: a Communications Class interface (with subclass "Abstract Control Model") and a Data Class interface, connected by a Union Functional descriptor that follows the Communications Class interface. Described in [USBPSTN1.2], with additional details as given in section 6.2 of this document.

- **LAN**: a Communications Class interface (with subclass "Ethernet Networking Control Model") and a Data Class interface, connected by a Union Functional descriptor that follows the Communications Class interface. Described in [USBECM1.2], with additional details as given in section 6.4 of this document.

- **OBEX**: a Communications Class interface (with subclass "OBEX Model") and a Data Class interface, connected by a Union Functional descriptor that follows the Communications Class interface. Defined in section 6.5 of this document.

- **HID:** a Human-Interface Device class interface, as defined in [USBHID1.1].

- **Audio:** Consists of a Communications Class interface (with subclass "Telephone Control Model", and a protocol code defining the AT command set to be used). The operation of this interface is as defined in section 6.3 of this document. In addition, there must be an Audio Control interface followed by one or more Audio Streaming interfaces, connected as described in [USBAUD1.0].

- **Device Management**: a Communications Class interface (with subclass "Device Management Model"), as defined in section 6.6 of this document.

- **MDLM**: a Communications Class interface (with subclass "Mobile Direct Line Model"), as defined in section 6.6.3 of this document. May be followed by one or more Data Class interfaces, as required. The Data Class interfaces are marked as part of this function because they are listed in a Union Functional descriptor that follows the Communications Class interface.

- **Mass Storage**: a mass storage class interface, as defined by [USBMASS1.1] and associated documents.

- **DFU**: a Device Firmware Upgrade interface (in application mode), as defined by [USBDFU1.0].

The device designer further identifies those portions of the device that make up a logical handset. To do this, the designer adds another Communications Class interface (with subclass "Wireless Handset Control Model"). This interface is followed by a Union Functional descriptor; the Union Functional descriptor identifies each of the interfaces that make up the logical handset.

This organization is discussed in more detail in section 6.1.

4.2 Device Operation

This specification imposes few additional operational requirements on the overall operation of a device containing functions defined herein. However, since all facilities are described statically, but might not be able to be used concurrently (due to network or device implementation restrictions), this specification requires that a specific set of practices be followed for managing this situation.

4.2.1 Contention

A given logical handset may have many logical functions, corresponding to different call facilities. The subscribed services, the local service provider, or the hardware or firmware of the handset may prevent some functions from being used concurrently.

The services available from the network via the handset might also change dynamically, even while a service is in use. For example, the user might reconfigure the handset via the keypad; the user might roam into a cell that cannot provide all services; or the network operator may decide to deny or retract service from the customer because of billing issues.

This specification refers to the process of dynamically determining whether a given call facility is available as "contention". It is so called, because different applications on the host may try to use different functions (modeling call capabilities) concurrently, and the handset is responsible for determining how to resolve any conflicting requests. Normally, the situation is resolved as if the network were unavailable. Any contention-related issues for specific functions are listed with the discussion of that function.

Dynamic retraction of service is also modeled as if the network were unavailable, so it is fundamentally similar to contention.

4.3 Function Models

[USB2.0] defines "function" as a "USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers". Further, in section 5.2.3, it says "Multiple functions may be packaged into what appears to be a single physical device…. A device that has multiple interfaces controlled independently of each other is referred to as a composite device." We therefore adopt the term "function" to describe a set of one or more interfaces which taken together provide a capability to the host.

This document defines the following new kinds of functions:

- Wireless Handset Control Model (WHCM)
- Mobile Direct Line Model (MDLM)
- Object Exchange (OBEX)
- Device Management

This document extends or reuses CDC based control models defined by [USBPSTN1.2] and [USBECM1.2] for specific kinds of WMC functions:

- All models are extended to allow additional protocols for the control plane.
- Telephone Control Model (TCM) is extended to allow use of AT commands for call control. The optional notification endpoint is mandatory in this specification.
- Abstract Control Model (ACM) is used for data/fax purposes. The optional notification endpoint is mandatory in this specification.
- Ethernet Networking Control Model is used for LAN frame exchange. The optional notification endpoint is mandatory in this specification.

This document reuses control models defined by other class specifications for specific kinds of WMC functions.

- The keypad (and, if necessary, the display) are modeled using HID interfaces. The implementation may follow the guidelines given in [INTTD].

- The audio flow to the handset and to the network are modeled using USB Audio class interfaces. The implementation may follow the guidelines given in [INTTD], with suitable modifications for the handset capabilities.

4.4 Interface Definitions

No interface classes are defined by this specification, beyond those defined in [USBCDC1.2] and the other class specifications. However, additional bInterfaceSubClass and bInterfaceProtocol codes are defined for use in constructing Communications Class interfaces. The bInterfaceSubClass codes are extended to represent the new function types. The bInterfaceProtocol codes are extended to allow additional AT commands, and to provide flexibility as these standards evolve.

4.5 Endpoint Requirements

The only additional endpoint requirement imposed by this specification is the general requirement for notification endpoints. This is an extension beyond the requirements of [USBCDC1.2].

4.6 Device Models

This specification targets multi-function devices, and is intended to allow free combination of logical handsets with other functions. Therefore, the device model is that of a composite device, as defined in [USB2.0].

5 Class Specific Codes

5.1 Communications Class Subclass Codes

Table 5-1: Communications Class Subclass Code

Code	Subclass
08h	Wireless Handset Control Model
09h	Device Management Model
0Ah	Mobile Direct Line Model
0Bh	OBEX Model

5.2 Communications Class Protocol Codes

Table 5-2: Communications Class Protocol Codes

Code	Subclass
02h	AT Commands defined by [PCCA101]
03h	AT Commands defined by [PCCA101] + [PCCA101-O]
04h	AT Commands defined by [GSM07.07]
05h	AT Commands defined by [3GPP27.007]
06h	AT Commands defined by [C-S0017-0]
FEh	External Protocol: Commands defined by Command Set functional descriptor

5.3 Communications Class Functional Descriptor Sub-Type Codes

Table 5-3: Communications Class Functional Descriptor Sub-Type Codes

Code	Description
11h	Wireless Handset Control Model Functional Descriptor
12h	Mobile Direct Line Model Functional Descriptor
13h	MDLM Detail Functional Descriptor
14h	Device Management Model Functional Descriptor
15h	OBEX Functional Descriptor
16h	Command Set Functional Descriptor
17h	Command Set Detail Functional Descriptor
18h	Telephone Control Model Functional Descriptor
19h	OBEX Service Identifier Functional Descriptor

5.4 Communications Class Management Element Request Codes

Table 5-4: Communications Class Management Element Request Codes

Request Code	Subclass
60h – 7Fh	MDLM Semantic-Model specific Requests (32 in all)

5.5 Communications Class Notification Element Request Codes

Table 5-5: Communications Class Notification Element Request Codes

Notification Code	Subclass
40h – 5Fh	MDLM Semantic-Model-specific Notifications (32 in all).

6 Functional Characteristics

This section describes how each standard abstract function is represented.

6.1 WHCM Logical Handset

A logical handset is modeled as a WHCM interface, plus a collection of interfaces that model various capabilities of the handset.

6.1.1 Functional Topology

The Logical Handset function has no endpoints of its own, nor does it have any Data Class interfaces directly associated with it. The interface that models the logical handset is a placeholder, which is followed by a Union Functional descriptor, that points to all the other interfaces that are part of functions which are part of the logical handset.

6.1.2 WHCM Descriptors

6.1.2.1 WHCM Interface Descriptor

One interface descriptor with bInterfaceClass == COMM and bInterfaceSubClass == WHCM and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each wireless device. Because cell phones typically have only one handset function, only one such descriptor will normally appear.

Following the WHCM descriptor, a number of functional descriptors appear.

6.1.2.2 Communications Class Header Functional Descriptor

This is as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

Table 6-1: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in Table 25 of [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.1.2.3 WHCM Functional Descriptor

This conveys subclass version information.

Table 6-2: Wireless Handset Control Model Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	ID for Wireless Handset Control Model functional descriptor.
3	bcdVersion	2	BCD number 0x0110	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. (Example: Version 1.1 of the spec is represented as 0x110.) Please note that the value used here is not necessarily the same as the version of [USBCDC1.2], which is 1.20 (0x120).

6.1.2.4 Communications Class Union Functional Descriptor (following WHCM interface)

This descriptor is formatted as a standard CDC Union Functional descriptor, and points to a collection of interfaces. However, the meaning is different.

- The Union Functional descriptor of other Communications Class functions points to other interfaces that are to be managed by the same driver that is bound to the Communications Class interface.

- However, the Union Functional descriptor of the WHCM interface points to other interfaces that are to be managed by drivers that are logically subordinate to the WHCM interface. The WHCM interface defines a handset; every interface that is part of the logical handset must appear in the Union Functional descriptor. This includes (in the case of Communications Class functions) the Communications interfaces and the Data Class interfaces.

For informative purposes, we repeat the definition of the Union Functional descriptor here.

Table 6-3: Union Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	4+n	Size of Descriptor in bytes; n is the count of subordinate interfaces defined below.
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[2]	1	Number	The interface number of this WHCM interface

Offset	Field	Size	Value	Description
4	bSubordinateInterface0[3]	1	Number	The interface number of the first subordinate interface.
…	…	…	…	…
4+n-1	bSubordinateInterface n-1	1	Number	The interface number of the last subordinate interface.

We suggest the following approach to assigning functions to a given logical handset. Consider the (highly hypothetical) case of two complete phones bonded into a single physical device. In this case, there must be two WHCM interfaces, one for each phone. Each WHCM interface is followed by a Union Functional descriptor that points to the USB Interface Descriptors that denote features that are specific to its logical phone.

If there is only one WHCM interface, then its Union Functional descriptor SHALL point to all the other interfaces associated with the WHCM interface. For example, consider the following case:

Interface 0: the master WHCM interface, representing the handset as a whole facility

Interface 1: the Abstract Control Model interface that represents a modem facility of the logical handset

Interface 2: the Data Class interface for interface 1.

Interface 3: the TCM interface that represents a voice-call facility

Interface 4: a DFU interface

Interface 5: a HID interface

Interface 6: an AUDIO CONTROL interface

Interface 7: an AUDIO STREAMING interface

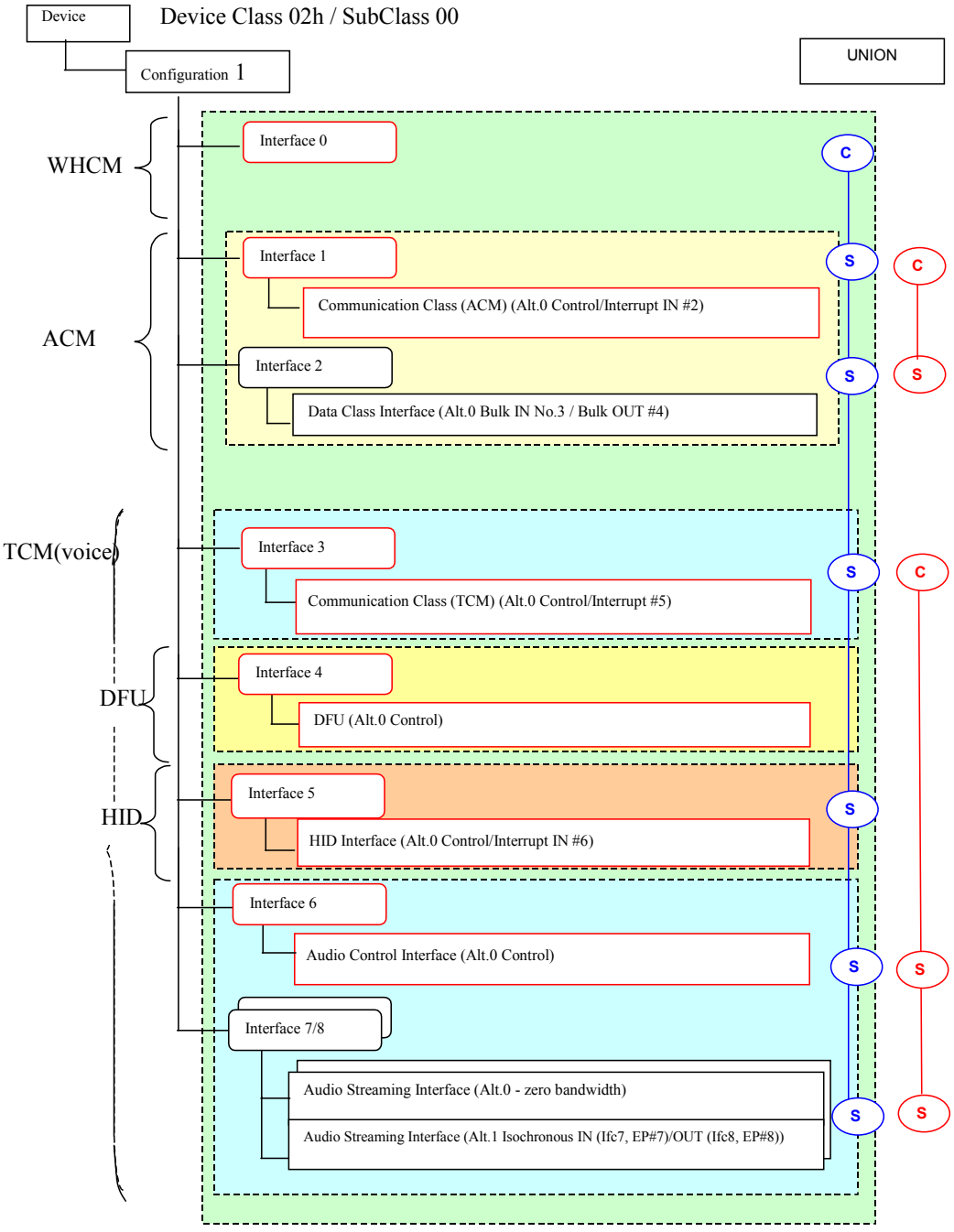
Interface 8: an AUDIO STREAMING interface

Interface 0 would be followed by (among other things) a Union Functional descriptor pointing to subordinate interfaces 1, 2, 3, 5, 6, 7 and 8. Interface 1 would be followed by (among other things) a Union Functional descriptor pointing to subordinate interface 2. Interface 3 would be followed by (among other things) a Union Functional descriptor pointing to subordinate interfaces 6, 7, and 8. Interface 6 would be followed by (among other things) an AUDIO class descriptor identifying interfaces 7 and 8 as subordinate interfaces.

If there are multiple logical handsets in the device, each would be modeled by a top-level WHCM interface (as interface 0). Each WHCM interface's Union Functional descriptor would point only to those interfaces that are logically part of the phone modeled by the WHCM interface. The functions that are global to the physical device would not appear in any WHCM descriptor, but would appear logically parallel to all of the WHCM interfaces.

Figure 6-1, below, is a graphical illustration of these relationships.

Figure 6-1: WHCM Union Functional Descriptor and Device Structure



6.1.3 Management Elements

No management elements are defined for use with the WHCM Interface.

6.1.4 Notifications

No notifications are defined for use with the WHCM interface, so no notification endpoint is defined.

6.2 Data/Fax Modem Functions

6.2.1 Functional Topology

A data/fax call facility consists of:

1. A Communications Class/Abstract Control Model (ACM) interface with a notification endpoint.
2. A Data Class interface with two endpoints, one BULK IN, the other BULK OUT.

Each Data/Fax function is represented by a standard Communications Class/Abstract Control Model interface with an interrupt endpoint. This interface in turn has a Union Functional descriptor pointing to a Data Class interface with two bulk endpoints.

This use is as defined in [USBPSTN1.2].

In order to minimize contention for the default pipe of the handset, all ACM interfaces shall provide a notification endpoint associated with the ACM interface. (This is an additional requirement, compared to [USBPSTN1.2].)

6.2.2 Descriptors

6.2.2.1 ACM Interface Descriptor

One interface descriptor with bInterfaceClass == COMM, bInterfaceSubClass == ACM, and bInterfaceProtocol == (some) AT Command shall be embedded in the configuration bundle for each data/fax.

Table 6-4: Communications Class Abstract Control Model Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant (02)	Abstract Control Model, as defined in [USBCDC1.2].
7	bInterfaceProtocol	1	Constant	Standard or enhanced AT Command set protocol, as defined in [USBCDC1.2]. Enhanced AT command set is repeated in Table 5-2 for reference.

Notice that the interface protocol includes a non-zero protocol code such as “AT command”, rather than 00, which indicates “no protocol”.

Following the ACM interface descriptor, a number of functional descriptors appear.

6.2.2.2 Communications Class Header Functional Descriptor

This is as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

Table 6-5: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.2.2.3 Abstract Control Management Functional Descriptor

This descriptor is mandatory.

Table 6-6: Abstract Control Management Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	4	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Abstract Control Management Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bmCapabilities	1	0x06 (byte)	Specifies the capabilities that this data/fax function supports. A bit value of zero means that the capability is not supported. D7..D4: RESERVED (Reset to zero) D3: Function generates the notification NetworkConnect ION D2: Function supports the management element SendBreak D1: Function supports the management elements GetLineCoding, SetControlLineState, GetLineCoding. Function will generate the notification SerialState. D0: Function supports management elements GetCommFeature, SetCommFeature and ClearCommFeature

Although the capabilities outlined above are based on [USBPSTN1.2], this specification imposes a further restriction. D3 through D0 of bmCapabilities shall always be coded as 0x06 (that is, D3 = 0, D2 = 1, D1 = 1, and D0 = 0). Furthermore, no host driver is required to support functions with bmCapabilities set to a value other than 0x06.

6.2.2.4 Call Management Functional Descriptor

This descriptor is mandatory.

Table 6-7: Call Management Functional Descriptor for Data/FAX Facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Call Management Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bmCapabilities	1	Bitmap	Specifies the capabilities that this data/fax function supports. D7..D2: RESERVED (Reset to zero) D1: 0 - Function sends/receives call management information only over this Communications Class interface 1 – Function can send/receive call management information over the Data Class interface. D0: 0 – Function does not perform call management 1 – Function does perform call management
4	bDataInterface	1	Number	bInterfaceNumber of the Data Class interface

Data/Fax functions shall always set bits D1 and D0 of bmCapabilities to 1, and shall always support sending call management via their Data Class interface.

Data/Fax functions shall always set byte 4 of this descriptor to the same value that is used in bInterfaceNumber of the Data Class interface descriptor (given below).

These constraints effectively mean that all Data/Fax facilities must support call management over the data class interface, and all Data/Fax facilities must perform call management.

6.2.2.5 Communications Class Union Functional Descriptor

This descriptor is formatted as a standard CDC Union Functional descriptor. For informative purposes, we repeat the definition of the Union Functional descriptor here.

This descriptor is mandatory.

Table 6-8: Union Functional Descriptor for Data/Fax Facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE

Offset	Field	Size	Value	Description
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[4]	1	Number	The interface number of this ACM interface
4	bSubordinateInterface0[5]	1	Number	The interface number of the Data Class interface.

6.2.2.6 Notification Endpoint Descriptor

This descriptor describes the INTERRUPT IN endpoint that transports notifications for this function.

This descriptor is mandatory.

6.2.2.7 Data Class Interface Descriptor

One interface descriptor with bInterfaceClass == DATA, bInterfaceSubClass == 0, and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each data/fax facility.

Table 6-9: Data Class Interface Descriptor for Data/FAX Facilities

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (0x0A)	Data Class
6	bInterfaceSubClass	1	Constant (00)	No subclass
7	bInterfaceProtocol	1	Constant (00)	No protocol.

Notice that the interface protocol is 00, which indicates “no protocol”. For embedded call control (AT commands) the particular command set is identified by the protocol code given by the facility’s Communications Class interface descriptor.

Following the Data Class interface descriptor, a number of functional descriptors may appear.

6.2.2.8 Data Class Header Functional Descriptor

This is as described in [USBCDC1.2], and has the format shown in Table 6-10.

This descriptor is optional, but must be first if it appears.

Table 6-10: Data Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.2.2.9 Endpoint Descriptors

Two endpoints must be provided.

1. A bulk IN endpoint
2. A bulk OUT endpoint.

These endpoint descriptors may appear in any order.

6.2.3 ACM Management Elements for Data/Fax

The management elements are described in [USBPSTN1.2], with adjustments given in Table 6-11.

Table 6-11: ACM Management Elements for Data/Fax Functions

Request	Requirements of [USBPSTN1.2]	Requirements of this standard
SendEncapsulatedCommand	Mandatory	mandatory (please see section 7.1 for additional discussion)
GetEncapsulatedResponse	Mandatory	mandatory
SetCommFeature	Optional	optional
GetCommFeature	Optional	optional
ClearCommFeature	Optional	optional
SetLineCoding	Optional	mandatory
GetLineCoding	Optional	mandatory
SetControlLineState	Optional	mandatory
SendBreak	Optional	mandatory

6.2.4 ACM Notifications for Data/Fax

Notifications are as defined in [USBPSTN1.2].

Table 6-12: ACM Notification Elements for Data/Fax Functions

Notification	Requirements of [USBPSTN1.2]	Requirements of this standard
NetworkConnection	Optional	not used
ResponseAvailable	Mandatory	mandatory
SerialState	Optional	mandatory

6.2.5 Contention

Call management is done using AT commands. When the function attempts to go logically off-hook, then the AT command interpreter requests the central resource manager on the handset (logically part of the TA/MT cluster) for the resources needed to place the call. If available, the AT command interpreter proceeds as normal; otherwise it aborts the call with "NO DIAL TONE" or another appropriate response, as determined by the handset vendor. This is analogous to what happens on a PABX when a user requests an outside line but no outside lines are available.

6.3 Voice Functions

6.3.1 Functional Topology

A Voice facility consists of:

1. A Communications Class/Telephone Control Model (TCM) interface with a notification endpoint.

2. An Audio Class Audio Control interface

3. At least two Audio Class Audio Streaming interfaces, to model the interface to the network. Each interface has one isochronous endpoint for transporting audio data.

The top level interface for modeling a voice call facility is a Communications Class/Telephone Control Model (TCM) interface. This interface in turn has a Union Functional descriptor which points to the Audio Control interface and Audio Streaming interfaces. However, as an extension to [USBPSTN1.2], call control is done using AT commands, transported to the interface using SendEncapsulatedCommand / GetEncapsulatedResponse.

In order to minimize contention for the default pipe of the handset, all TCM interfaces shall provide a notification endpoint associated with the TCM interface. (This is an additional requirement, compared to [USBPSTN1.2].)

6.3.2 Descriptors

6.3.2.1 TCM Interface Descriptor

One interface descriptor with bInterfaceClass == COMM, bInterfaceSubClass == TCM, and bInterfaceProtocol == (some) AT Command shall be embedded in the configuration bundle for each voice facility.

Table 6-13: Communications Class Telephone Control Model Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant (03)	Telephone Control Model, as defined in [USBCDC1.2].
7	bInterfaceProtocol	1	Constant	Standard or enhanced AT Command set protocol, as defined in [USBCDC1.2]. Enhanced AT command set protocols repeated in Table 5-2 for reference.

Notice that the interface protocol includes a non-zero protocol code such as “AT command”, rather than 00, which indicates “no protocol”.

Following the TCM descriptor, a number of functional descriptors appear.

6.3.2.2 Communications Class Header Functional Descriptor

This is as described in [USBCDC1.2], and shown in Table 6-5.

This descriptor is mandatory, and must be the first functional descriptor.

Table 6-14: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in Table 25 of [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.3.2.3 Telephone Control Model Functional Descriptor

This descriptor is mandatory. This descriptor is defined by this document, and is an extension to [USBPSTN1.2]

Table 6-15: Telephone Control Model Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	7	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Telephone Control Model Functional Descriptor subtype, as given in Table 5-3.
3	bcdVersion	2	BCD number 0x0110	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. (Example: Version 1.2 of the spec would be represented as 0x120, not 0x102.) Please note that the value used here is not necessarily the same as the version of [USBCDC1.2], which is 1.20 (0x120).
5	wMaxCommand	2	word	Describes the maximum number of characters that can be transported in a single SendEncapsulatedCommand. This shall be at least 256 decimal (0x100).

The host may send up to wMaxCommand characters in a single encapsulated command If the host attempts to send more than this, the device shall send an error response (a STALL PID) during the data or status phase of the transfer.

6.3.2.4 Communications Class Union Functional Descriptor

This descriptor is formatted as a standard CDC Union Functional descriptor. Instead of pointing to each of the Data Class interfaces within this function, it points to the Audio class interfaces which provide the

voice transport. For informative purposes, we repeat the definition of the Union Functional descriptor here.

This descriptor is mandatory.

Table 6-16: Union Functional Descriptor for Voice Call Facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	Number	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[6]	1	Number	The interface number of this TCM interface
4	bSubordinateInterface0[7]	1	Number	The interface number of the audio class audio control interface.
5	bSubordinateInterface1	1	Number	The interface number of the first audio class audio streaming interface
…	…	…	…	…
n+3	bSubordinateInterface n-1	1	Number	The interface number of the last audio-class audio streaming interface.

6.3.2.5 Notification Endpoint Descriptor

This descriptor describes the INTERRUPT IN endpoint that transports notifications for this function.

This descriptor is mandatory.

6.3.3 Management Elements

The following management elements, defined in [USBPSTN1.2], are required.

6.3.3.1 SendEncapsulatedCommand

The host uses this to send encapsulated commands to the device. As described above, the function specifies the maximum amount of data that can be sent with SendEncapsulatedCommand using the wMaxCommand field of the Telephone Control Model Functional Descriptor.

6.3.3.2 GetEncapsulatedResponse

The host uses this to fetch responses from a previous encapsulated command. If no response is currently available, the function shall immediately return a zero-length packet during the data-IN phase. If more

data is available than the host has asked for, the function shall arrange so that a subsequent GetEncapsulatedResponse will get the next portion of the response. SetInterface targeting the Telephone Control Model interface, SetConfiguration and USB Reset shall all cause any pending encapsulated response data to be discarded by the function.

RING indications, dialing progress, and other network notifications are also presented to the host via GetEncapsulatedResponse. Whenever a new such notification is delivered, the function shall also send a ResponseAvailable notification over the notification endpoint associated with the Telephone Control Model interface.

After issuing a complete encapsulated command, the host software must get all the associated responses before issuing the next encapsulated command. The effect of not waiting is not specified.

6.3.4 Notifications

The only notification is ResponseAvailable, as defined in [USBPSTN1.2].

6.3.5 Contention

Call management is done using AT commands. When the host sends an AT command that causes the function to try to use a network resource (for example, a call facility or the radio in the MT), the AT command interpreter must contend for the resources needed to place the call. It does this by asking the central resource manager on the handset (logically part of the TA/MT cluster). If the resources are available, the AT command interpreter proceeds to place the call; Otherwise it aborts the call with "NO DIAL TONE" or another appropriate response, as determined by the handset vendor. This is analogous to what happens on a PABX when a user requests an outside line but no outside lines are available.

6.4 LAN Frame Functions

6.4.1 Functional Topology

A LAN frame traffic facility is consists of:

1. A Communications Class/Ethernet Networking Control Model interface with a notification endpoint.

2. A Data Class interface with two endpoints, one BULK IN, the other BULK OUT.

This is just as defined for a single-function Ethernet adapter in [USBECM1.2]. However, the notification endpoint is required.

6.4.2 Descriptors

6.4.2.1 Ethernet Networking Control Model Interface Descriptor

One interface descriptor with bInterfaceClass == COMM, bInterfaceSubClass == Ethernet Networking Control Model, and bInterfaceProtocol == 00 shall be embedded in the configuration bundle for each data/fax.

Table 6-17: Communications Class Ethernet Networking Control Model Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant (06)	Ethernet Networking Control Model, as defined in [USBCDC1.2].
7	bInterfaceProtocol	1	Constant (00)	No specific protocol.

Notice that the interface protocol is 00, which indicates “no protocol”.

Following the Ethernet Networking Control Model interface descriptor, a number of functional descriptors appear.

6.4.2.2 Communications Class Header Functional Descriptor

This is as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

6.4.2.3 Ethernet Networking Functional Descriptor

This descriptor is mandatory for LAN frame facilities. For informative purposes, the definition is repeated from [USBECM1.2].

Table 6-18: Ethernet Networking Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	Number	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Ethernet Networking Functional Descriptor subtype, as defined in Table 25 of [USBCDC1.2]
3	iMacAddress	1	Index	Index of string descriptor giving the Ethernet MAC address for this facility. Must not be zero. The MAC address must be formatted in UNICODE as specified in [USBECM1.2].
4	bmEthernetStatistics	4	Bitmask	Mask of supported statistics. Stored in little-endian order.
8	wMaxSegmentSize	2	Number	The maximum segment size that the LAN frame facility can support, normally 1514 bytes. Stored in little-endian order
10	wNumberMCFilters	2	Bitmask	Indicates the number of multicast filters supported, as defined by table 41 of [USBECM1.2].
12	bNumberPowerFilters	1	Number	Indicates the number of power filters implemented by the function.

This specification requires that the Ethernet address specified by the string at iMacAddress be the same no matter which (valid) language code is used with GetDescriptor to retrieve it. After conversion, the first three octets of the address must be the OUI assigned by the IEEE to the authority assigning the address. The remaining three octets must be unique to this physical device.

6.4.2.4 Communications Class Union Functional Descriptor

This descriptor is formatted as a standard CDC Union Functional descriptor. For informative purposes, we repeat the definition of the Union Functional descriptor here.

This descriptor is mandatory.

Table 6-19: Union Functional Descriptor for LAN frame facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE

Offset	Field	Size	Value	Description
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[8]	1	Number	The interface number of this Ethernet Networking Control Model interface, as given by bInterfaceNumber.
4	bSubordinateInterface0[9]	1	Number	The interface number of the Data Class interface.

6.4.2.5 Notification Endpoint Descriptor

This descriptor describes the INTERRUPT IN endpoint that transports notifications for this function.

This descriptor is mandatory.

6.4.2.6 Data Class Interface Descriptor, Alternate Setting Zero

One interface descriptor with bAlternateSetting == 0, bInterfaceClass == DATA, bInterfaceSubClass == 0, and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each LAN frame facility.

Table 6-20: Data Class Interface Descriptor for LAN frame facilities, Setting 0

Offset	Field	Size	Value	Description
3	bAlternateSetting	1	Number (0)	Indicates that this descriptor is for alternate setting zero
4	bNumEndpoints	1	Number (0)	Indicates that no endpoints are associated with this alternate setting.
5	bInterfaceClass	1	Constant (0x0A)	Data Class
6	bInterfaceSubClass	1	Constant (00)	No subclass
7	bInterfaceProtocol	1	Constant (00)	No protocol.

The interface protocol is 00, which indicates "no protocol". No endpoints are permitted in alternate setting zero (in accordance with [USBECM1.2]).

Following the Data Class interface descriptor for alternate setting 0, a number of functional descriptors may appear.

6.4.2.7 Data Class Header Functional Descriptor

This is as described above, in Table 6-10.

This descriptor is optional, but must be first if it appears.

6.4.2.8 Data Class Interface Descriptor, Alternate Setting not Zero

At least one additional interface descriptor with bAlternateSetting != 0, bInterfaceClass == DATA, bInterfaceSubClass == 0, and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each LAN frame facility.

Table 6-21: Data Class Interface Descriptor for LAN frame facilities, non-zero Setting

Offset	Field	Size	Value	Description
3	bAlternateSetting	1	Number (non-zero)	Indicates that this descriptor is for alternate setting other than zero
4	bNumEndpoints	1	Number (2)	Indicates that two endpoints are associated with this alternate setting.
5	bInterfaceClass	1	Constant (0x0A)	Data Class
6	bInterfaceSubClass	1	Constant (00)	No subclass
7	bInterfaceProtocol	1	Constant (00)	No protocol.

The interface protocol is 00, which indicates "no protocol". Two endpoints are required in the non-zero alternate setting (in accordance with [USBECM1.2]).

Following the Data Class interface descriptor for the non-zero alternate setting, a number of functional descriptors may appear.

6.4.2.9 Data Class Header Functional Descriptor

This is as described in above, in Table 6-10.

This descriptor is optional, but must be first if it appears.

6.4.2.10 Endpoint Descriptors, Alternate Setting not Zero

Two endpoints must be provided.

3. A bulk IN endpoint
4. A bulk OUT endpoint.

These endpoint descriptors may appear in any order.

6.4.2.11 Additional Alternate Data Class Settings

A designer may provide additional Data Class settings as desired, in order to support alternate encapsulation methods or endpoint types.

6.4.3 Management Elements

The management elements for LAN frame facilities are as defined by [USBECM1.2].

6.4.4 Notifications

The notifications for LAN frame facilities are as defined by [USBECM1.2].

6.4.5 Contention

Contention involving the LAN function is handled by simulating connect and disconnect to the virtual ether. When the host system enables the LAN function, the host driver signals this to the device by selecting a non-zero alternate interface setting on the Data Class interface. The Ethernet Networking Control Model handler on the device then tries to establish a connection, arbitrating with the TA/MT and negotiating with the network. If the connection is successful, the function sends a NetworkConnect (up) notification to the host over the notification pipe; otherwise the function sends a NetworkConnect (down) notification. [Since the handset might or might not be in range of a base station, NetworkConnect (up/down) will happen periodically anyway in response to network changes.]

Provisioning aspects are beyond the scope of this specification.

6.5 OBEX Functions

The OBEX data exchange facility is conceptually modeled using a Communications Class/Abstract Control Model interface. However, OBEX differs from ACM in two ways:

1. There is no communication with a remote device. Therefore, there is no need for the various communication-related commands from ACM (setting baud rate, etc.)

2. There is no separate command language, and therefore no need for SendEncapsulatedCommand or GetEncapsulatedResponse.

The OBEX protocol is fundamentally message oriented; but the OBEX protocol is designed to be layered above TCP-like protocols, which are character oriented. Therefore, OBEX is implemented using a pair of bulk pipes, just as for data plane of ACM modems.

We define a new subclass code for OBEX, and a new set of functional descriptors.

In order to simplify re-use of existing drivers for implementing OBEX, the transport semantics are as close as possible to a subset of ACM.

6.5.1 Functional Topology

The OBEX function is modeled as a Communications Class interface plus a Data Class interface. The Data Class interface shall have two alternate settings. Alternate setting 0 shall have no endpoints; alternate setting 1 shall provide two bulk endpoints, one BULK-IN and the other BULK-OUT. As with the networking control models, placing the interface in alternate setting 0 shall return the data plane to a default state, resetting the OBEX function. See [USBECM1.2] for more information.

6.5.2 OBEX Descriptors

6.5.2.1 OBEX Interface Descriptor

One Communications Class/OBEX interface is embedded in the configuration bundle for each object-exchange locus. This is a Communications Class interface, with subclass OBEX, as defined in Table 5-1.

Table 6-22: Communications Class OBEX Model Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant	OBEX Model, as defined in Table 5-1
7	bInterfaceProtocol	1	Constant (00)	Indicates that no specific command protocol is used. (The OBEX protocol is implied by the use of OBEX model.)

A protocol code of zero is used because OBEX Model interfaces do not support encapsulated commands or any variations on the OBEX protocol.

6.5.2.2 Communications Class Header Functional Descriptor

Following the OBEX interface descriptor is a Communications Class Header Functional Descriptor, as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

Table 6-23: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.5.2.3 OBEX Functional Descriptor

This conveys subclass version information. Because the mode switching logic is opaque to the layer that manages this interface, mode-switching capabilities are not presented here.

Table 6-24: OBEX Control Model Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	ID for OBEX Control Model functional descriptor.
3	bcdVersion	2	BCD number	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. Version 1.2 of the spec is represented as 0x120.

6.5.2.4 Union Functional Descriptor

Table 6-25: Union Functional Descriptor for OBEX Facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[10]	1	Number	The interface number of this OBEX interface

Offset	Field	Size	Value	Description
4	bSubordinateInterface0[11]	1	Number	The interface number of the Data Class interface that transports OBEX data.

A Communications Class OBEX interface must have exactly one Data Class interface. For this reason, the Union Functional descriptor has a fixed length of 5 bytes.

6.5.2.5 OBEX Service Identification Functional Descriptor (Optional)

This optional functional descriptor indicates the mode supported by this OBEX function. This corresponds to an OBEX role (client or server), a particular OBEX service, and an OBEX service version.

The descriptor consists of a fixed length set of fields, incorporating the OBEX role, service identifier and version (see Table 6-26).

Table 6-26: Functional Descriptor for OBEX service identification

Offset	Field	Size	Value	Description
0	bFunctionLength	1	22	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubType	1	Constant	ID for OBEX Service Identification Functional Descriptor
3	bmOBEXRole	1	Bit mask	Represents the OBEX role to be played by the function (See Table 6-27below)
4	bOBEXServiceUUID[16]	16	Byte Constant	A 16 byte UUID value used to indicate the particular OBEX service associated with this function.
20	wOBEXServiceVersion	2	Value	A 16 bit value indicating the version of the OBEX service associated with this function.

Table 6-27: Bitmask values for bmObexRole

Bit of bmObexRole	Meaning
D7–D1	Reserved (set to zero)
D0	If set to 0, this function acts as an OBEX Server when alternate interface setting 1 is activated. If set to 1, this function acts as an OBEX Client when alternate interface setting 1 is activated.

The interpretation of bOBEXServiceUUID depends upon the value of D0 in the corresponding bmOBEXRole field:

- If D0 in bmOBEXRole is set to 1, (meaning OBEX client), bOBEXServiceUUID is used to identify the OBEX service on the Host that this function may request to connect to.
- If D0 in bmOBEXRole is set to 0, (meaning OBEX server), bOBEXServiceUUID is used to identify an OBEX service provided in server mode by this function.

Section 6.5.2.5.1 describes the derivation of these UUID values in more detail.

Table 6-28: UUID values defined by WMC OBEX

UUID name	Value
WMC_DEFAULT_OBEX_SERVER_UUID	02aeb320f64911da974d0800200c9a66[12]

The 16-bit field wOBEXServiceVersion is used to indicate the precise version number of the service identified by bOBEXServiceUUID (Refer to section 6.5.2.5.1).. For OBEX functions presenting an OBEX service identification functional descriptor with a bOBEXServiceUUID value equal to WMC_DEFAULT_OBEX_SERVER_UUID (refer to Table 6-28), the value of wOBEXServiceVersion is considered to be irrelevant, and must be set to zero (Refer to section 6.5.2.5.2).

6.5.2.5.1 Reuse of Bluetooth GOEP based profiles

OBEX was originally defined as a part of the Infra Red Data Association protocol suite, and later adopted by the Bluetooth Special Interest Group (SIG). A number of UUID values have been defined by Bluetooth SIG as service class constants, these being associated with particular Bluetooth Profiles[13]. The

[12] This value is not generated by the Bluetooth SIG and therefore needs to be defined in this document.
[13] : Membership of the Bluetooth SIG at (at least) "Adopter" level is necessary to access current Bluetooth specifications. At time of writing, there are no fees associated with obtaining Adopter membership of the Bluetooth SIG) Any changes to UUIDs and all Assigned Numbers must go through BARB so a request to BARB-chair@bluetooth.org would need to be submitted.

mapping of service class constants to their corresponding profiles are defined in the "Service Classes" section of [Bluetooth Reserved Numbers].

A subset of these service class constants are associated with Bluetooth Profiles based on OBEX - termed as "Generic Object Exchange Profile" (GOEP) based profiles. Reuse of Bluetooth profiles over OBEX functions is restricted to these GOEP based profiles. The introduction of OBEX Service UUIDs to USB offers opportunities for reuse and increased interoperability.

An OBEX USB Service UUID takes the form of a 128-bit value (see [LEACH1998]), uniquely identifying a particular OBEX service. Bluetooth service class constants are usually defined in a 16 bit UUID format. These 16 bit UUIDs are directly transformable into the usual 128 bit UUID format, following rules set down in the Bluetooth core specification. When these service class constants are utilized in bOBEXServiceUUID, they shall be expanded to their full 128 bit UUID format

OBEX functions presenting an OBEX service identification functional descriptor with a bOBEXServiceUUID value matching that of a Bluetooth GOEP based profile shall utilise wOBEXServerVersion to indicate the specific version of the Bluetooth profile being referenced, in the format specified for a profile version number in a Bluetooth Profile Descriptor List. The service shall adhere to the OBEX level characteristics of the corresponding version of the corresponding OBEX based Bluetooth profile.

6.5.2.5.2 OBEX Default Server connections

Where bOBEXServiceUUID is set to WMC_DEFAULT_OBEX_SERVER_UUID (refer to Table 6-28), it is taken that the function provides (if an OBEX server) or requires (if an OBEX client) a connection to an OBEX Default Server.

The OBEX Default Server provides a generic OBEX Inbox service to which objects can be Put, but it can also act as a gateway to other services. For these other services, it is assumed that appropriate OBEX Client to OBEX Server routing is achieved through other means, particularly OBEX command level routing by the use of Target Headers in OBEX Connect packets (see [OBEX1.3]). Where an OBEX Connect is issued without a Target Header being specified, it is taken that the client wishes to connect to the OBEX Default Server.

In practice, the use of functions with this bOBEXServiceUUID value opens up opportunities for a variety of schemes for routing and multiplexing OBEX services. One example would be to have a pool of multi-purpose OBEX pipe bundles available for OBEX transfers, with routing to specific OBEX services occurring within the OBEX Server itself (whether the USB function or host has taken on the OBEX Server role). Such usage schemes depend on the capabilities of the OBEX implementation bound over the OBEX function.

6.5.2.6 OBEX Communications Interface Endpoint Descriptors

No notifications are supported by the OBEX subclass, and therefore no Interrupt endpoint is provided with this interface.

6.5.2.7 Data Class Interface Descriptor, Alternate Setting Zero

One interface descriptor with bAlternateSetting == 0, bInterfaceClass == DATA, bInterfaceSubClass == 0, and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each OBEX facility.

Table 6-29: Data Class Interface Descriptor for OBEX facilities, Setting 0

Offset	Field	Size	Value	Description
3	bAlternateSetting	1	Number (0)	Indicates that this descriptor is for alternate setting zero
4	bNumEndpoints	1	Number (0)	Indicates that no endpoints are associated with this alternate setting.
5	bInterfaceClass	1	Constant (0x0A)	Data Class
6	bInterfaceSubClass	1	Constant (00)	No subclass
7	bInterfaceProtocol	1	Constant (00)	No protocol.

The interface protocol is 00, which indicates "no protocol". No endpoints are permitted in alternate setting zero.

Whenever this setting is selected, the OBEX engine shall abandon all work in progress in an appropriate way, and shall return to an idle state.

Following the Data Class interface descriptor for alternate setting 0, a number of functional descriptors may appear.

6.5.2.8 Data Class Header Functional Descriptor

This is as described above, in Table 6-10.

This descriptor is optional, but must be first if it appears.

6.5.2.9 Data Class Interface Descriptor, Alternate Setting not Zero

At least one additional interface descriptor with bAlternateSetting != 0, bInterfaceClass == DATA, bInterfaceSubClass == 0, and bInterfaceProtocol == 0 shall be embedded in the configuration bundle for each OBEX facility.

Table 6-30: Data Class Interface Descriptor for OBEX facilities, non-zero Setting

Offset	Field	Size	Value	Description
3	bAlternateSetting	1	Number (non-zero)	Indicates that this descriptor is for alternate setting other than zero
4	bNumEndpoints	1	Number (2)	Indicates that two endpoints are associated with this alternate setting.
5	bInterfaceClass	1	Constant (0x0A)	Data Class
6	bInterfaceSubClass	1	Constant (00)	No subclass
7	bInterfaceProtocol	1	Constant (00)	No protocol.

The interface protocol is 00, which indicates "no protocol". Two endpoints are required in the non-zero alternate setting.

When this setting is selected, the OBEX service in the device shall be operational.

Following the Data Class interface descriptor for the non-zero alternate setting, a number of functional descriptors may appear.

6.5.2.10 Data Class Header Functional Descriptor

This is as described in above, in Table 6-10.

This descriptor is optional, but must be first if it appears.

6.5.2.11 Endpoint Descriptors, Alternate Setting not Zero

Two endpoints must be provided.

5. A bulk IN endpoint
6. A bulk OUT endpoint.

These endpoint descriptors may appear in any order.

6.5.3 Management Elements

6.5.3.1 Establishing OBEX transport connection

Since it is always the OBEX Client entity that initiates an OBEX connection, the sequence of events required depending on whether the OBEX client resides in a USB device or in a USB Host (refer to Appendix B: for more details on OBEX connections). However, in both cases it is assumed that the host must activate the OBEX interface using SetInterface to activate the relevant OBEX interface. This corresponds to a running host application which is either an OBEX client or OBEX server as appropriate. This is a basic requirement for valid interactions with any given host.

6.5.3.2 Suspend, Resume and Remote Wakeup

When a device supporting an OBEX function is placed into Suspend mode, the OBEX session shall not be terminated.

- If the Suspend mode is exited through USB Resume, the OBEX session state shall be maintained.
- If the Suspend mode is exited through a USB Reset, power interruption (e.g. VBus removed), cable detach or through any other means, then all OBEX session state shall be abandoned."

6.5.3.3 Session Request Protocol

For SRP capable USB OTG peripherals the end of a session also terminates the OBEX session.

6.5.4 Notifications

No notifications are used by the OBEX class.

6.5.5 Contention

The OBEX interface does not involve the use of the air interface (MT), so contention is not an issue.

6.6 Device Management Functions

6.6.1 Functional Topology

The Device Management data exchange facility allows management information to be exchanged between the logical handset and the USB host, without interfering with any other activities that are in process. The facility is conceptually similar to a Communications Class/Abstract Control Model interface. However, Device Management differs from ACM in two ways:

1. There is no data class interface; all information is exchanged using commands over the control endpoint.

2. The AT command set is the only way of communicating with this interface; there is no "data plane" associated with this kind of interface.

6.6.2 Device Management Descriptors

6.6.2.1 Device Management Interface Descriptor

One Communications Class/Device Management interface is embedded in the configuration bundle for each device management locus. Only one should appear per logical handset.

Since Device Management data exchange is normally low volume and infrequently exchanged, this class has no need for separate data endpoints.

Table 6-31: Communications Class Device Management Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant	Device Management Model, as defined in Table 5-1.
7	bInterfaceProtocol	1	Constant	Standard or enhanced AT Command set protocol, as defined in [USBCDC1.2].;Enhanced AT command set protocols repeated in Table 5-2 for reference.

Notice that the interface protocol includes a non-zero protocol code such as "AT command", rather than 00, which indicates "no protocol".

Following the Device Management interface descriptor, a number of functional descriptors appear.

6.6.2.2 Communications Class Header Functional Descriptor

Following the Device Management interface descriptor is a Communications Class Header Functional Descriptor, as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

Table 6-32: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bcdCDC	2	Number 0x0120	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.6.2.3 Device Management Functional Descriptor

This conveys subclass version information and device design decision information to the host.

Table 6-33: Device Management Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	7	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	ID for Device Management Control Model functional descriptor.
3	bcdVersion	2	BCD number	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. Version 1.2 of the spec is represented as 0x120.
5	wMaxCommand	2	Number	The buffer size allocated in the device for data sent from the host using SendEncapsulatedCommand. If the host attempts to send more than this in a single command, the device shall fail the request. This shall be at least 256 decimal (0x100).

6.6.2.4 Device Management Notification Endpoint

In order not to interfere with other functions, it's important that activities on the default pipe execute quickly. Therefore, the COMM/Device Management interface must also provide a separate notification endpoint to carry notifications back to the host, to indicate that an encapsulated response is available.

The notification endpoint shall be an INTERRUPT IN endpoint.

Note that a notification endpoint is *required*.

6.6.3 Management Elements

The management elements used with the Device Management facility are:

SendEncapsulatedCommand (mandatory)

GetEncapsulatedResponse (mandatory)

6.6.4 Notifications

The only notifications supported by the Device Management class are:

ResponseAvailable (mandatory)

6.6.5 Contention

The Device Management facility does not involve the use of the air interface (MT), so contention is not an issue.

6.7 MDLM Transport Functions

MDLM transport allows a driver with sufficient knowledge to take over direct control of the MT radio component.

Due to regulatory issues, although the MDLM transport effectively has direct control of the radio transmitter and receiver, the entire capability of the radio is not available to the host computer. Command sets and data formats are limited, based on the type of network (GPRS, cdmaOne, cdma2000, W-CDMA, and so forth) for which the handset was designed. These command sets and data formats, while possibly standard within a network, may vary from location to location.

In any given MDLM application, however, the command sets and data formats are necessarily well defined. We collectively refer to a given standard set of data formats, management elements, and notification elements as a *MDLM semantic model*.

Therefore, the MDLM interface descriptor must identify the MDLM semantic model to be used with the interfaces that form the given MDLM function, so that the appropriate host driver can be loaded, and can do the right thing. Rather than establish a central authority for issuing MDLM identifiers, UUIDs (also known as GUIDs) are used to identify the MDLM transport in use. These UUIDs are as defined in [OPENC309] and [LEACH1998].

6.7.1 Functional Topology

MDLM is modeled using a master Communications Class/Mobile Direct Line Model interface. The interface descriptor is followed by a CDC Header Functional Descriptor, a MDLM Functional Descriptor, and a Union Functional descriptor that indicates one or more Data Class interfaces that provide data transport facilities.

6.7.2 Descriptors

6.7.2.1 MDLM Interface Descriptor

One interface descriptor with bInterfaceClass == Communications, bInterfaceSubClass == MDLM, and bInterfaceProtocol set to an appropriate value shall be embedded in the configuration bundle for each instance of an MDLM facility.

Table 6-34: Communications Class Mobile Direct Line Model Interface Descriptor

Offset	Field	Size	Value	Description
5	bInterfaceClass	1	Constant (02)	Communications Class
6	bInterfaceSubClass	1	Constant	Mobile Direct Line Model, as defined in Table 5-1.
7	bInterfaceProtocol	1	Constant	Control plane protocol.

Following the MDLM Interface Descriptor, a number of functional descriptors appear.

6.7.2.2 Communications Class Header Functional Descriptor

This is as described in [USBCDC1.2].

This descriptor is mandatory, and must be first.

Table 6-35: Communications Class Header Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Header Functional Descriptor subtype, as defined in [USBCDC1.2]
3	bcdCDC	2	Number 0x0110	Release number of [USBCDC1.2] in BCD, with implied decimal point between bits 7 and 8. 0x0120 == 1.20 == 1.2.

6.7.2.3 Mobile Direct Line Model Functional Descriptor

This descriptor is mandatory. It conveys the GUID that uniquely identifies the kind of MDLM interface that is being provided.

Table 6-36: Mobile Direct Line Model Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	21 (0x15)	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Mobile Direct Line Model Functional Descriptor subtype, as given in Table 5-3 above
3	bcdVersion	2	BCD number	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. Version 1.2 of the spec is represented as 0x120. This is based on the version of *this document*, not on the version of the transport specification defined by bGUID.
5	bGUID[16]	16	Byte Constant	Uniquely identifies the detailed transport protocol (and therefore the host drivers) provided by this MDLM interface. The GUID is given in wire-order, as defined in [LEACH1998].

6.7.2.4 MDLM Detail Functional Descriptor

This descriptor is optional, and may be repeated as necessary. It conveys any additional information required by the MDLM transport specification identified by the MDLM Functional Descriptor. If present, it must occur after the MDLM Functional Descriptor, in the same group of functional descriptors.

Table 6-37: MDLM Detail Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	4+n	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	MDLM Detail Functional Descriptor subtype, as given in Table 5-3 above
3	bGuidDescriptorType	1	Constant	Discriminator, interpreted according to the semantic model specified by the GUID in the MDLM Functional Descriptor
4	bDetailData[n]	n	Byte Constant	Information associated with this GUID and discriminator, according to the semantic model specified by the GUID in the MDLM Functional Descriptor

This specification allows the MDLM Semantic Model to define up to 256 different kinds of MDLM Detail Functional Descriptors, to convey additional information as needed. bGuidDescriptorType identifies the kind of descriptor that is present.

This specification further allows the device designer complete freedom to provide as many MDLM Detail Functional Descriptors as are needed for a given device and MDLM Semantic Model.

6.7.2.5 Communications Class Union Functional Descriptor

This descriptor is formatted as a standard CDC Union Functional descriptor. It points to the Data Class interfaces that form part of this MDLM function. For informative purposes, we repeat the definition of the Union Functional descriptor here.

This descriptor is mandatory.

Table 6-38: Union Functional Descriptor for MDLM Facilities

Offset	Field	Size	Value	Description
0	bFunctionLength	1	5	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Union Functional Descriptor Subtype as defined in [USBCDC1.2]
3	bControlInterface[14]	1	Number	The interface number of this MDLM interface
4	bSubordinateInterface0[15]	1	Number	The interface number of the first subordinate interface.
…	…	…	…	…
n+3	bSubordinateInterface n-1	1	Number	The interface number of the last subordinate interface.

6.7.2.6 Notification Endpoint Descriptor

This descriptor describes the INTERRUPT IN endpoint that transports notifications for this function.

This descriptor is optional.

6.7.3 Management Elements

The management elements used by the MDLM function are, necessarily, determined by the kind of MDLM interface identified by the MDLM functional descriptor. Therefore, a group of management elements are reserved for use as determined by the semantic model in use.

Table 6-39: MDLM-Specific-Write

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
00100001B	MDLM Semantic-model-specific code (from Table 5-4) ; meaning defined by semantic model of MDLM interface identified by wIndex	as defined by semantic model of MDLM interface identified by wIndex	MDLM Interface	Length of Data	Data for designated operation (sent from host to device); meaning defined by semantic model of MDLM interface identified by wIndex

Table 6-40: MDLM-Specific-Read

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	MDLM Semantic-model-specific code (from Table 5-4) ; meaning defined by semantic model of MDLM interface identified by wIndex	as defined by semantic model of MDLM interface identified by wIndex	MDLM Interface	Length of Data	Data for designated operation (sent from device to host); meaning defined by semantic model of MDLM interface identified by wIndex

6.7.4 Notifications

All data passed over the notification endpoint must be encapsulated as notifications according to [USBCDC1.2]. The notification elements used by the MDLM function are, necessarily, determined by the kind of MDLM interface identified by the MDLM functional descriptor. Therefore, a group of notification elements codes are reserved for use as determined by the semantic model in use.

Table 6-41: MDLM-Specific-Read

bmRequestType	bNotificationCode	wValue	wIndex	wLength	Data
10100001B	MDLM Semantic-model-specific code (from Table 5-5); meaning defined by semantic model of MDLM interface identified by wIndex	as defined by semantic model of MDLM interface identified by wIndex	MDLM Interface	Length of Data	Data for designated operation (sent from device to host following the header); meaning defined by semantic model of MDLM interface identified by wIndex

6.7.5 Contention

Contention handling is defined by the MDLM semantic model.

See MDLM semantic model for discussions of what other facilities in the handset can do when the MDLM facility is in use.

7 Device Requests

No additional class-level requests are defined by this specification. Subclass-specific requests are given as defined in section 6, "Functional Characteristics". However, the use of encapsulated commands for command transport is clarified by this document.

7.1 Encapsulating AT Command Data

[USBPSTN1.2] defines SendEncapsulatedCommand and GetEncapsulatedResponse as mechanisms for transporting control-plane data. Experience has shown that the defined mechanisms are explained with insufficient clarity to ensure interoperability between drivers and firmware designed by different development groups.

For the purposes of this subclass specification, functions within devices shall be implemented to transport AT command data using the assumptions and algorithms outlined in this section.

This specification makes the basic assumption that the WMC device's firmware already has an AT command interpreter, for processing data from other (non-USB) sources. Such interpreters have two kinds of state:

1. state related to the byte-by-byte processing

2. state related to command processing

Type-2 state interacts with type-1 state. (For example, the value of S3 changes the logical <CR> character. This affects when a command is deemed to be completed, as well as affecting how responses are formatted.)

Normally, it is convenient to share the AT interpreter code across all data streams. For this reason, this specification assumes that the AT command interpreter's lower edge is a bidirectional character-oriented interface.

Therefore, when using SendEncapsulatedCommand to transport data, this specification requires that the firmware **ignore** messaging boundaries implied by the SendEncapsulatedCommand sequence. The firmware shall process bytes delivered by SendEncapsulatedCommand in the same way as if the bytes had been delivered via a serial port, or as embedded commands over the Data interface. Therefore, the host sends exactly the same sequence of bytes when using SendEncapsulatedCommand that it would send using any other transport media.

The function shall *not* return STALL in response to errors in the AT commands transported by SendEncapsulatedCommand. The function shall return STALL if the firmware cannot immediately accept the bytes transported by SendEncapsulatedCommand, due to buffer errors. Since "aborts" are signaled by sending characters while dialing is in process, the firmware must provide sufficient buffering to accept (and must not STALL) abort sequences sent while dialing.

The function shall provide a certain minimum amount of input buffering.

- For Abstract Control Model functions, the firmware shall provide a minimum of 256 bytes of buffering.

- For other functions (Device Management, Telephone Control Model), the functional descriptors must reflect the actual amount of buffering provided.

AT detection, echoing and response code formatting are done exactly as they are done when using other character-oriented data transport methods. As with all reliable USB transports, reply data is queued internally by the device firmware until the host requests the data (via an IN-token or a GetEncapsulatedResponse management element).

The firmware shall interpret GetEncapsulatedResponse as a request to read response bytes. The firmware shall send the next wLength bytes from the response. The firmware shall allow the host to retrieve data using any number of GetEncapsulatedResponse requests. The firmware shall return a zero-length reply if there are no data bytes available.

The firmware shall send ResponseAvailable notifications periodically, using any appropriate algorithm, to inform the host that there is data available in the reply buffer. The firmware is allowed to send ResponseAvailable notifications even if there is no data available, but this will obviously reduce overall performance.

For functions that support both SendEncapsulatedCommand and other control methods, this specification requires that the firmware maintain logically separate command and response streams for each potential control method. Commands are not intermixed at the character level. Commands are separately accumulated for each stream.

Because a function may have multiple command and response streams, the host may concurrently send commands over any of the streams. For example, a call may be placed over a TA using AT commands embedded in the data stream, while at the same time management operations are performed over the same TA using AT commands sent via SendEncapsulatedCommand. In order for this to work, firmware shall maintain certain AT command interpreter state variables on a control stream-by-stream basis. The states variables that are maintained for each control stream shall include the values most recently set for E#, Q#, V#, S3, S4, S5, and any other vendor-specific state.

The function shall not return STALL in response to GetEncapsulatedResponse.

If a function has multiple control streams, firmware needs to know which stream shall receive unsolicited responses (for example, RING notifications).

- In case of Abstract Control Model modems (section 6.2), the unsolicited responses shall go to the Data interface, if the Data interface is configured to transport embedded call-control information. Otherwise, the notifications shall go to the GetEncapsulatedResponse queue.

- In other cases defined by this specification, there is only one control stream associated with the function, so this issue doesn't apply.

8 Device Descriptors

8.1 Standard USB Interface Descriptors

8.1.1 Device Descriptor

The device descriptor is a standard device descriptor, as specified in Chapter 9 of the USB 2.0 specification. The device class shall be 0x02 (Communications Class). The device subclass and protocol codes shall be 0x00. The default pipe maximum packet size shall be 64 bytes.

8.1.2 Configuration Bundle

The configuration bundle starts with a configuration descriptor, which is as specified in Chapter 9 of the USB 2.0 specification.

8.1.2.1 Additional Function-Specific Descriptors

The remaining descriptors in the configuration bundle present the interfaces for each of the subordinate functions. The subclass-specific descriptors are given in the appropriate parts of 6, "Functional Characteristics".

8.1.2.2 Command Set Functional Descriptor

Although there is a tendency towards standardization, a large variety of AT command sets are presently defined. Since the 3G standards are still evolving, a designer may need to mark a function's specific command set in a generic way, even though a bInterfaceProtocol code has not been allocated for that command set.

If a Communications Class interface identifies its protocol as "External Protocol", as given in Table 5-2, then host software uses the Command Set functional descriptor to define the command set to be used by this interface. In all cases, the commands are transported using "Send Encapsulated Command" and "Get Encapsulated Response", just as AT commands would be.

This descriptor is mandatory after a Communications Interface, if that Communication Interface's bInterfaceProtocol code is "External Protocol", as defined in Table 5-2. Otherwise, this descriptor is forbidden.

Table 8-1: Command Set Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	22 (0x16)	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Command Set Functional Descriptor subtype, as given in Table 5-3 above
3	bcdVersion	2	BCD number	Version number for this subclass specification. Initially 0x0100. The implied decimal point is between bits 7 and 8. Version 1.2 of the spec is represented as 0x120. This is based on the version of *this document*, not on the version of the command set identified by bGUID.
5	iCommandSet	1	Index	Index of string that describes this command set
6	bGUID[16]	16	Byte Constant	Uniquely identifies the command set to be used when communicating with this interface. The GUID is given in wire-order, as defined in [LEACH1998].

8.1.2.3 Command Set Detail Functional Descriptor

This descriptor is optional, and may be repeated as necessary. It conveys any additional information required by the command set identified by the Command Set Functional Descriptor. If present, it must occur after a Communications Class Interface Descriptor, with bInterfaceProtocol == EXTERNAL_PROTOCOL (as defined in Table 5-2: Communications Class Protocol Codes), and after the Command Set Functional Descriptor, in the same group of functional descriptors.

Table 8-2: Command Set Detail Functional Descriptor

Offset	Field	Size	Value	Description
0	bFunctionLength	1	4+n	Size of Descriptor in bytes
1	bDescriptorType	1	Constant	CS_INTERFACE
2	bDescriptorSubtype	1	Constant	Command Set Detail Functional Descriptor subtype, as given in Table 5-3 above
3	bGuidDescriptorType	1	Constant	Discriminator, interpreted according to the command set model specified by the GUID in the Command Set Functional Descriptor
4	bDetailData[n]	n	Byte Constant	Information associated with this GUID and discriminator, according to the command set model specified by the GUID in the Command Set Functional Descriptor

This specification allows the Command Set Semantic Model to define up to 256 different kinds of Command Set Functional Descriptors, to convey additional information as needed. bGuidDescriptorType identifies the kind of descriptor that is present.

This specification further allows the device designer complete freedom to provide as many Command Set Functional Descriptors as are needed for a given device and Command Set Semantic model.

Appendix A: Windows Driver Architecture

Vendors are free to use whatever driver structure best suit their technical and commercial goals. However, for readers unfamiliar with the details of using multi-function devices under the Windows operating system, this section shows a driver architecture that can be used to integrate the functionality of a WMC handset into Windows. This section is informative, not normative.

For the purposes of discussion, we assume a relatively complicated handset, containing the following functions:

- Two Modem functions (one intended for data calls, one intended for fax calls)
- LAN frame
- OBEX
- Device Management
- Mass storage
- HID Class, Audio Class and Telephone Control Model (for placing voice calls)
- Device Firmware Update

A bus driver performs the operations involved in dividing the device into multiple functions. This bus driver is a replacement for the composite device driver provided by Microsoft, and must be supplied by the vendor or obtained from a third party.

For modem, FAX, mass storage, and Ethernet frame, existing class drivers can be used as they are. (In the case of Ethernet frame, class drivers identical to those used for DOCSIS Cable Modems are appropriate and may be obtained from third parties.)

For device management, minor changes will be needed to extend the existing operating-system ACM drivers, because all communication is using the default pipe.

For OBEX, although this interface is similar to that of a modem, different drivers are needed because most of the semantics of OBEX differ from a standard modem.

For Voice, the same driver architecture can be used as was suggested in Intel's White Paper, "USB Telephony Devices: Interfaces for Value Add Feature Sets", section 3, PSTN Support. The TAPI service provider and the telephone management driver must be provided by the vendor or obtained from a third party; but the audio presentation and the telephone handset functions are implemented using Microsoft standard drivers.

Please refer to Figure A-1 on page 57 for an overview of the driver architecture.

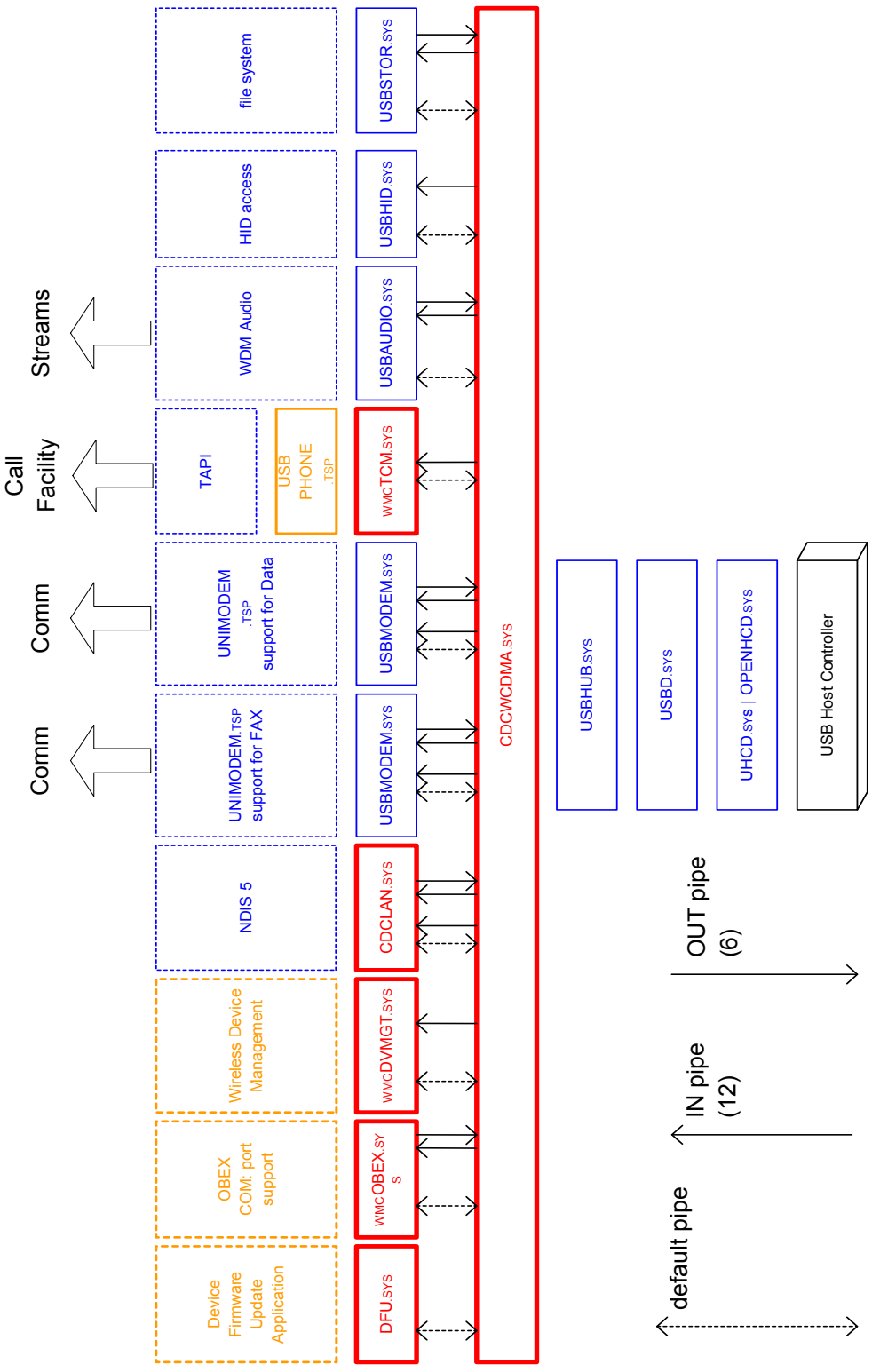


Figure A-1 Windows Driver Architecture

Appendix B: OBEX Connections

The information in this section is purely informative. It introduces the concept of OBEX transport connections, but contains no normative information on how an OBEX function must be implemented. Where concepts introduced in this section are used in normative sections of this specification, they are used to illustrate the interaction between the USB and nominal OBEX entities.

Over any transport medium, before an OBEX protocol exchange can begin, a transport connection must be established over which the OBEX packets can be sent. The OBEX Client, because it sends the first OBEX packet in an exchange, is responsible for initiating the establishment of the transport connection (except where it is already established, for instance where the transport connection has not yet been torn down following a previous OBEX session). Once the transport connection has been established, the OBEX Client initiates the connection at the OBEX level by sending an OBEX Connect packet. The OBEX specification describes the protocol used from this point onwards.

Equally, when an OBEX protocol exchange has been completed, the transport connection must be torn down (following the optional but recommended exchange of an OBEX Disconnect / Disconnect Response). This tearing down is normally done in a controlled sequence of exchanges by software state machines. In practice however, transport connections may be broken at any time. For example, USB cables may be removed, IrDA line of sight may be broken, or Bluetooth devices may go out of range. OBEX implementations must therefore be resilient to abrupt non-controlled transport disconnections.

From the OBEX point of view, any OBEX transport has two basic top level states; “transport up” and “transport down”. When its transport is up, OBEX can exchange data with a peer OBEX entity over a bi-directional data channel associated with that transport. When the transport is down, OBEX cannot exchange data. Clearly, OBEX entities need to affect and monitor the state of their underlying transports. This may be done by means of a small number of request and indication primitives. These primitives are illustrated in Figure B-2 below.

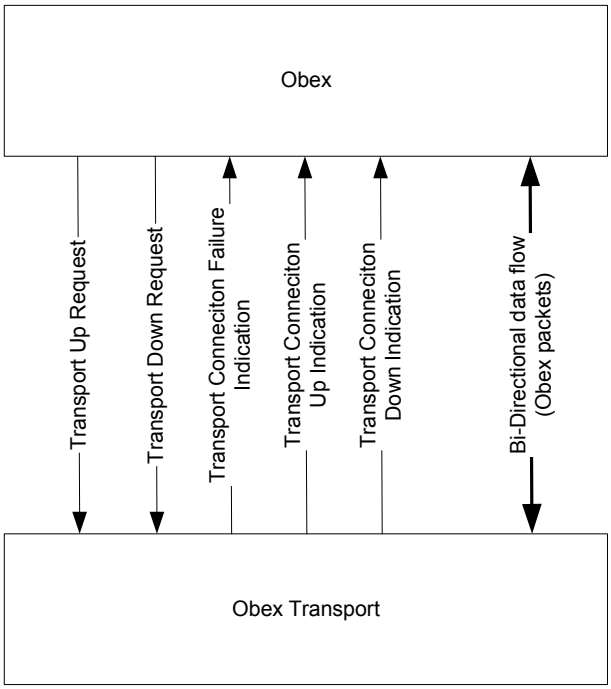


Figure B-2 OBEX transport control and indication primitives

The purposes of these primitives are as follows;

Transport Up Request
This request is issued by an OBEX client (never by an OBEX server) to prompt the OBEX transport to bring itself up. In response to such a request, the OBEX client expects to asynchronously receive either a **Transport Connection Failure Indication** or a **Transport Connection Up Indication** upon the respective failure or success of bringing up the transport connection.

Transport Down Request
This request is issued by an OBEX entity (either OBEX client or OBEX server) to prompt the OBEX transport to bring itself down. Since this request is not permitted to fail, the OBEX entity expects in response asynchronously to receive a **Transport Connection Down Indication** once the underlying transport link has been torn down.

Transport Connection Failure Indication
This indication is issued by an OBEX transport, and indicates that its attempt to bring up a transport connection in response to an OBEX client's **Transport Up Request** has been unsuccessful.

Transport Up Indication
This indication is issued by an OBEX transport, and indicates that the transport channel has been brought up (is available for data transfer). Where the associated OBEX entity is an OBEX client, this will be the result of a local **Transport Up Request** . Where the associated OBEX entity is an OBEX server, this will be the result of a remote OBEX client requesting a transport connection to a local OBEX server.

Transport Down Indication
This indication is issued by an OBEX transport, and indicates that the transport channel has been brought down (is not available for data transfer). This may be a result of either the local or remote OBEX entity requesting to bring down the transport connection, or simply a breakage of the underlying link (e.g. disconnection of a USB cable).