Universal Serial Bus
Class Definitions
for
Communication Devices

Version 1.1
January 19, 1999

Scope of this Revision

This version 1.1 of this class specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

Revision History

Revision	Issue date	Comments
1.1	January 19, 1999	Approved by DWG as 1.1
1.09d	January 19, 1999	RR91
1.09c	December 17, 1998	RR90
1.09b	December 14, 1998	RR69-89
1.09a	October 21, 1998	RR68
1.0.8a- 1.0.9rc	April 28, 1998 — October 13, 1998	1.08 releases
1.0	May 8, 1998	Approved by DWG as 1.0
0.9a- 0.9f	May 26, 1997 — October 10, 1997	0.9 releases.
0.8 – 0.8h	December 11, 1996 — May 20, 1997	0.8 releases.

Contributors

Andy Nicholson	Microsoft Corporation
Charlie Tai	Intel Corporation
Chuck Brabenac	Intel Corporation
Dan Moore	Diamond Multimedia Systems
Dave Perry	Mitel Corporation
Diego Friedel	AVM
Ed Endejan	3Com Corporation
Jim Wilson	U.S. Robotics
Joe Decuir	Microsoft Corporation
John Howard	Intel Corporation
Ken Lauffenburger	Efficient Networks, Inc.
Kenny Richards	Microsoft Corporation
Mats Webjörn	Universal Access
Nathan Peacock	Northern Telecom
Paul E. Berg	Moore Computer Consultants, Inc.
Randy Fehr	Northern Telecom
Ron Lewis	Rockwell Semiconductors
Paul Chehowski	Mitel Corporation
Shelagh Callahan	Intel Corporation
Stefan Eder	Siemens Semiconductors
Terry Moore	Moore Computer Consultants, Inc.

USB Class Definitions for Communication Devices
Copyright © 1996-1999 USB Implementers' Forum

All rights reserved.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

A LICENSE IS HEREBY GRANTED TO REPRODUCE AND DISTRIBUTE THIS SPECIFICATION FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY OTHER INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. AUTHORS OF THIS SPECIFICATION ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

Hayes is a registered trademark of Hayes Microcomputer Products, Inc.
All other product names are trademarks, registered trademarks, or service marks of their respective owners.

Please send comments via the discussion forum at:
http://www.usb.org/developers/dwg/communications_class_grp/

Contents

1. Introduction 1
1.1 Scope 1
1.2 Purpose 1
1.3 Related Documents 1
1.4 Terms and Abbreviations 4
2. Management Overview 7
3. Functional Characteristics 8
3.1 Device Organization 8
3.1.1 Communication Device Management 8
3.2 Device Operation 9
3.3 Interface Definitions 9
3.3.1 Communication Class Interface 9
3.3.2 Data Class Interface 10
3.3.2.1 Protocol Data Wrapper 11
3.4 Endpoint Requirements 12
3.4.1 Communication Class Endpoint Requirements 12
3.4.2 Data Class Endpoint Requirements 12
3.5 Device Models 12
3.6 USB POTS Models 12
3.6.1 Direct Line Control Model 13
3.6.1.1 DL Model 13
3.6.1.2 Datapump Model 14
3.6.2 Abstract Control Model 15
3.6.2.1 Abstract Control Model Serial Emulation 16
3.6.3 USB Telephone Model 17
3.6.3.1 Telephone Control Model 17
3.7 USB ISDN Models 18
3.7.1 Multi-Channel Model 18
3.7.1.1 Topology 20
3.7.2 USB CAPI Model 20
3.7.2.1 CAPI Control Model 21
3.8 USB Networking Models 22
3.8.1 Common Data Plane Characteristics 22
3.8.1.1 Segment Delineation 23
3.8.1.2 Segment Size 24
3.8.2 Ethernet Networking Control Model 24
3.8.3 ATM Networking Control Model 26
4. Class-Specific Codes for Communication Devices 28
4.1 Communication Device Class Code 28
4.2 Communication Interface Class Code 28
4.3 Communication Interface Class SubClass Codes 28
4.4 Communication Interface Class Control Protocol Codes 29
4.5 Data Interface Class Codes 29
4.6 Data Interface Class SubClass Codes 29
4.7 Data Interface Class Protocol Codes 29
5. Descriptors 31
5.1 Standard USB Descriptor Definitions 31
5.1.1 Device Descriptor 31

5.1.2 Configuration Descriptor 31
5.1.3 Interface Descriptors 31
5.1.4 Endpoint Descriptors 32
5.2 Class-Specific Descriptors 32
5.2.1 Class-Specific Device Descriptor 32
5.2.2 Class-Specific Configuration Descriptor 32
5.2.3 Functional Descriptors 32
5.2.3.1 Header Functional Descriptor 34
5.2.3.2 Call Management Functional Descriptor 34
5.2.3.3 Abstract Control Management Functional Descriptor 35
5.2.3.4 Direct Line Management Functional Descriptor 36
5.2.3.5 Telephone Ringer Functional Descriptor 37
5.2.3.6 Telephone Operational Modes Functional Descriptor 38
5.2.3.7 Telephone Call and Line State Reporting Capabilities Descriptor 39
5.2.3.8 Union Functional Descriptor 40
5.2.3.9 Country Selection Functional Descriptor 41
5.2.3.10 USB Terminal Functional Descriptor 41
5.2.3.11 Network Channel Terminal Functional Descriptor 42
5.2.3.12 Protocol Unit Functional Descriptor 43
5.2.3.13 Extension Unit Functional Descriptor 43
5.2.3.14 Multi-Channel Management Functional Descriptor 44
5.2.3.15 CAPI Control Management Functional Descriptor 44
5.2.3.16 Ethernet Networking Functional Descriptor 45
5.2.3.17 ATM Networking Functional Descriptor 47
5.3 Sample Class-Specific Functional Descriptors 49

6. Communication Interface Class Messages 51
6.1 Overview 51
6.2 Management Element Requests 51
6.2.1 SendEncapsulatedCommand 54
6.2.2 GetEncapsulatedResponse 54
6.2.3 SetCommFeature 54
6.2.4 GetCommFeature 54
6.2.5 ClearCommFeature 55
6.2.6 SetAuxLineState 56
6.2.7 SetHookState 56
6.2.8 PulseSetup 56
6.2.9 SendPulse 57
6.2.10 SetPulseTime 57
6.2.11 RingAuxJack 57
6.2.12 SetLineCoding 57
6.2.13 GetLineCoding 58
6.2.14 SetControlLineState 58
6.2.15 SendBreak 59
6.2.16 SetRingerParms 59
6.2.17 GetRingerParms 60
6.2.18 SetOperationParms 60
6.2.19 GetOperationParms 60
6.2.20 SetLineParms 61
6.2.21 GetLineParms 62
6.2.22 DialDigits 63
6.2.23 SetUnitParameter 64
6.2.24 GetUnitParameter 64
6.2.25 ClearUnitParameter 64

6.2.26 GetProfile 65
6.2.27 SetEthernetMulticastFilters 65
6.2.28 SetEthernetPowerManagementPatternFilter 65
6.2.29 GetEthernetPowerManagementPatternFilter 66
6.2.30 SetEthernetPacketFilter 66
6.2.31 GetEthernetStatistic 67
6.2.32 SetATMDataFormat 69
6.2.33 GetATMDeviceStatistics 71
6.2.34 SetATMDefaultVC 72
6.2.35 GetATMVCStatistics 72
6.3 Notification Element Notifications 73
6.3.1 NetworkConnection 74
6.3.2 ResponseAvailable 74
6.3.3 AuxJackHookState 74
6.3.4 RingDetect 74
6.3.5 SerialState 75
6.3.6 CallStateChange 75
6.3.7 LineStateChange 77
6.3.8 ConnectionSpeedChange 77
Appendix A: Communication Device Class Examples 79
A.1 Basic Telephone 79
A.2 Modem 79
A.3 CAPI Device 79
Appendix B: Sample Configurations 80
B.1 Basic Telephony Configurations 80
B.2 Modem Configurations 80
B.3 CAPI Device Configuration 81
Appendix C: Multi-channel ISDN B-Channel setup 83
C.1 General 83
Appendix D: Multi-Channel Implementation Examples 84
D.1 ISDN BRI T/A with two POTS interfaces 84
D.2 ISDN BRI T/A with vendor specific protocol (Bonding) 85
D.3 Passive ISDN Solutions 85
Appendix E: Data Class Protocol Definitions 88
Definitions 88
E.1 Physical Interface Protocols 88
E.1.2 I.430: BASIC USER-NETWORK INTERFACE – LAYER 1 88
E.2 Framing Protocols 92
E.2.1 HDLC Framing 92
E.2.2 Transparent framing 95
E.3 Data Link Protocols 97
E.3.1 Q.921 Management: ISDN USER-NETWORK INTERFACE DATA LINK LAYER SPECIFICATION FOR CIRCUIT MODE BEARER SERVICES 97
E.3.2 Q.921: ISDN USER-NETWORK INTERFACE DATA LINK LAYER SPECIFICATION FOR CIRCUIT MODE BEARER SERVICES 99
E.3.3 Q.921 TEI-multiplexor: TERMINAL ENDPOINT IDENTIFIER MULTIPLEXOR FOR ISDN USER-NETWORK INTERFACE DATA LINK LAYER 101
E.4 Network layer Protocols 102
E.4.1 Q.931/Euro-ISDN User Side 102

E.4.2 V.42*bis*: Data compression procedures for DCE using error correction procedures 109
E.4.3 V.120: V.24 rate adaptation to ISDN 110

List of Tables

Table 1: Data Class Protocol Wrapper Layout 11
Table 2: Requests — Direct Line Control Model* 13
Table 3: Notifications — Direct Line Control Model* 14
Table 4: Requests — Abstract Control Model* 16
Table 5: Notifications — Abstract Control Model* 16
Table 6: Requests — Telephone Control Model* 18
Table 7: Notifications — Telephone Control Model* 18
Table 8: Requests — Multi-Channel Model* 19
Table 9: Requests — CAPI Control Model* 21
Table 10: Requests — Ethernet Networking Control Model* 25
Table 11: Notifications — Ethernet and ATM Networking Control Models* 25
Table 12: Requests — ATM Networking Control Model* 26
Table 13: Notifications — Ethernet and ATM Networking Control Models* 27
Table 14: Communication Device Class Code 28
Table 15: Communication Interface Class Code 28
Table 16: Communication Interface Class SubClass Codes 28
Table 17: Communication Interface Class Control Protocol Codes 29
Table 18: Data Interface Class Code 29
Table 19: Data Interface Class Protocol Codes 29
Table 20: Communication Device Class Descriptor Requirements 31
Table 21: Communication Class Interface Descriptor Requirements 31
Table 22: Data Class Interface Descriptor Requirements 32
Table 23: Functional Descriptor General Format 32
Table 24: Type Values for the bDescriptorType Field 33
Table 25: bDescriptor SubType in Functional Descriptors 33
Table 26: Class-Specific Descriptor Header Format 34
Table 27: Call Management Functional Descriptor 34
Table 28: Abstract Control Management Functional Descriptor 35
Table 29: Direct Line Management Functional Descriptor 36
Table 30: Telephone Ringer Functional Descriptor 37
Table 31: Telephone Operational Modes Functional Descriptor 39
Table 32: Telephone Call State Reporting Capabilities Descriptor 40
Table 33: Union Interface Functional Descriptor 40
Table 34: Country Selection Functional Descriptor 41
Table 35: USB Terminal Functional Descriptor 42
Table 36: Network Channel Terminal Functional Descriptor 42
Table 37: Protocol Unit Functional Descriptor 43
Table 38: Extension Unit Functional Descriptor 43
Table 39: Multi-Channel Management Functional Descriptor 44
Table 40: CAPI Control Management Functional Descriptor 45
Table 41: Ethernet Networking Functional Descriptor 45
Table 42: Ethernet Statistics Capabilities 46
Table 43: ATM Networking Functional Descriptor 47
Table 44: Sample Communication Class Specific Interface Descriptor* 49
Table 45: Class-Specific Requests 51
Table 46: Class-Specific Request Codes 53
Table 47: Communication Feature Selector Codes 55

Table 48: Feature Status Returned for ABSTRACT_STATE Selector 55
Table 49: POTS Relay Configuration Values 56
Table 50: Line Coding Structure 58
Table 51: Control Signal Bitmap Values for SetControlLineState 58
Table 52: Ringer Configuration Bitmap Values 59
Table 53: Operation Mode Values 60
Table 54: Line State Change Value Definitions 61
Table 55: Line Status Information Structure 62
Table 56: Line State Bitmap 62
Table 57: Call State Bitmap 63
Table 58: Call State Value Definitions 63
Table 59: Characters in a Dialing Command 63
Table 60: Unit Parameter Structure 64
Table 61: Power Management Pattern Filter Structure 66
Table 62: Ethernet Packet Filter Bitmap 67
Table 63: Ethernet Statistics Feature Selector Codes 68
Table 64: ATM Data Format 69
Table 65: ATM Device Statistics Feature Selector Codes 71
Table 66: ATM VC Selector Codes 72
Table 67: Class-Specific Notifications 73
Table 68: Class-Specific Notification Codes 73
Table 69: UART State Bitmap Values 75
Table 70: Call State Change Value Definitions 76
Table 71: Line State Change Values 77
Table 72: ConnectionSpeedChange Data Structure 78
Table 73: Telephone Configurations 80
Table 74: Example Modem Configurations 80
Table 75: Example CAPI Device Configurations 82
Table 76: Command Type Encoding 88
Table 77: I.430 Configuration Parameter List 88
Table 78: I.430 Command Message Format 88
Table 79: I.430 Commands 89
Table 80: I.430 Activate, Deactivate Command Wrapper 89
Table 81: I.430 PhActivateBReq Command Wrapper 90
Table 82: I.430 PhDeactivateBReq Command Wrapper 90
Table 83: I.430 PhDataReq Command Wrapper 90
Table 84: I.430 MphErrorInd Command Wrapper 90
Table 85: I.430 MphInformationInd Command Wrapper 91
Table 86: HDLC Configuration Parameter List 92
Table 87: HDLC Commands 94
Table 88: HDLC ControlRes, StatusReq Command Wrapper 94
Table 89: HDLC ControlReq Command Wrapper 94
Table 90: HDLC StatusInd/Res Command Wrapper 94
Table 91: HDLC DataReq/Ind Command Wrapper 95
Table 92: TRANS Configuration Parameter List 95
Table 93: TRANS Commands 96
Table 94: TRANS ControlRes, StatusReq Command Wrapper 96
Table 95: TRANS ControlReq Command Wrapper 96
Table 96: TRANS StatusInd/Res Command Wrapper 96

Table 97: TRANS DataReq/Ind Command Wrapper 97
Table 98: Q.921M Configuration Parameter List 97
Table 99: Q.921M Command Message Format 98
Table 100: Q.921M Commands 98
Table 101: Q.921M DlAssignReq wrapper 98
Table 102: Q.921M DlAssignInd, DlRemoveInd Command Wrapper 99
Table 103: Q.921M DlErrorReq, DlErrorCon Command Wrapper 99
Table 104: Q.921 Configuration Parameter List 99
Table 105: Command Message Format 100
Table 106: Q.921 commands 100
Table 107: Q.921 General Message Structure 100
Table 108: Q.931/Euro-ISDN Configuration Parameter List 102
Table 109: Q.931/Euro-ISDN Command Message Format 102
Table 110: Q.931/Euro-ISDN Commands 104
Table 111: Q.931/Euro-ISDN System Management Commands 104
Table 112: Q.931/Euro-ISDN General Command Structure 104
Table 113: V.42*bis* Configuration Parameter List 109

1. Introduction

There are three classes that make up the definition for communication devices: the Communication Device Class, Communication Interface Class and the Data Interface Class. The Communication Device Class is a device level definition and is used by the host to properly identify a communication device that may present several different types of interfaces. The Communication Interface Class defines a general-purpose mechanism that can be used to enable all types of communication services on the Universal Serial Bus (USB). The Data Interface Class defines a general-purpose mechanism to enable bulk or isochronous transfer on the USB when the data does not meet the requirements for any other class.

1.1 Scope

Given the broad nature of communication equipment, this specification does not attempt to dictate how all communication equipment should use the USB. Rather, it defines an architecture that is capable of supporting any communication device. The current release of the specification focuses on supporting connectivity to telecommunication services (devices that have traditionally terminated an analog or digital telephone line), and medium speed networking services ("Always Connected" LAN/WAN media types). The specification currently outlines the following types of devices:

- *Telecommunications devices:* analog modems, ISDN terminal adapters, digital telephones, and analog telephones
- *Networking devices:* ADSL modems, cable modems, 10BASE-T Ethernet adapters/hubs, and "Ethernet" cross-over cables.

This specification does not attempt to redefine existing standards for connection and control of communication services. The Communication Class defines mechanisms for a device and host to identify which existing protocols to use. Where possible, existing data formats are used and the transport of these formats are merely enabled by the USB through the definition of the appropriate descriptors, interfaces, and requests. More specifically, this specification describes a framework of USB interfaces, data structures, and requests under which a wide variety of communication devices can be defined and implemented.

1.2 Purpose

This specification provides information to guide implementers in using the USB logical structures for communication devices. This information applies to manufacturers of communication devices and system software developers.

1.3 Related Documents

Universal Serial Bus Specification, version 1.0 and version 1.1 (also referred to as the *USB Specification*). This specification is available on the World Wide Web site **http://www.usb.org**.

Universal Serial Bus Common Class Specification, version 1.0. This specification is available on the World Wide Web site **http://www.usb.org**.

ANSI/TIA-602, *Serial Asynchronous Automatic Dialing and Control* - available at **http://www.eia.org**.

Bellcore NI-1 (National ISDN 1), *Support network terminating services for ISDN service* - available at **http://www.bellcore.com**.

ITU V.25ter, *Serial Asynchronous Automatic Dialing and Control* - available at **http://www.itu.ch**

ITU-T V.42 (03/93), *Error correction procedures for DCE using asynchronous to synchronous conversion*

ITU-T V.42*bis* (1990*), Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures*

ITU-T I.430 (11/95), *Basic user-network interface – layer 1 specification*

ITU-T I.431 (03/93), *Primary rate user-network interface – layer 1 specification*

ITU-T Q.921 (03/93), *ISDN user-network interface data link layer specification*

ITU-T Q.922 (1992), *ISDN data link layer specification for frame mode bearer services*

ITU-T Q.931 (03/93), *ISDN user-network interface—layer 3 specification for basic Call Control*

ITU-T Q.2931 (02/95), B-ISDN DSS2 User Network Interface (UNI) Layer 3 Specification for Basic Call/Connection Control.

ITU-T X.25 (03/93), Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit

ITU-T X.75 (10/96), Packet switched signaling system between public networks providing data transmission services – available at **http://www.itu.ch**.

ITU-T T.30 (07/96), Procedures for document facsimile transmission in the general switched telephone network – available at **http://www.itu.ch**.

ITU-T.200, *Programmable communication interface for terminal equipment connected to ISDN appendix II*

ITU-T V.110 (09/92), Support of data terminal equipment with V-series type interfaces by an integrated services digital network.

ITU-T V.120 (09/92), Support by an ISDN terminal adapter equipment with V-series type interface with provision for statistical multiplexing.

CAPi2.0, *COMMON-ISDN-API Version 2.0 – available at* **http://www.capi.org**.

ETSI prETS 300 838, integrated service digital network (ISDN); harmonized programmable communication interface (HPCI) for ISDN

ETSI prETS 300 917, digital cellular telecommunications system (Phase 2+);
GSM application programming Interface (GSM-API) (GSM 07.08 version 5.0.0)

Data Over Cable Service Interface Specifications (DOCSIS) Customer Premises to CPE Interface (CMCI) Interim Specification, SP-CMCI-I02-980317, March, 1998 – available at **http://www.cablemodem.com**.

Ethernet Version 2.0, Digital, Intel, Xerox (DIX), 1982.

ISO/IEC 8802-3 (ANSI/IEEE Std 802.3): 1993, Information technology — Local and metropolitan area networks — Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.

ADSL Forum TR-002, ATM over ADSL Recommendations, March 1997.

ANSI T1.413, Network and Customer Installation Interfaces - Asymmetric Digital Subscriber Line (ADSL) Metallic interface 1995.

ITU-T I.361 - B-ISDN ATM Layer specification, November 95.

ITU-T I.363 – B-ISDN ATM Adaptation Layer (AAL) Specification, March 93

ITU-T I.413 (3/93) - B-ISDN User-Network Interface

ITU-T I.610 - B-ISDN Operation and Maintenance Principles and Functions, March 93.

Detailed examples of typical communication device classes are provided in separate white papers that are not a part of this specification. The latest copies of the white papers can be found at **http://www.usb.org**.

1.4 Terms and Abbreviations

802.3	Second generation networking cabling and signaling, commonly known as Ethernet II.(IEEE 802.3)
AAL	ATM Adaptation Layer
ADSL	Asymmetric Digital Subscriber Line
ASVD	Analog Simultaneous Voice and Data, signaling method mixes data and voice.
AT COMMAND SET	A telecommunication device control protocol. For details, see TIA-602 or V.25ter.
ATM	Asynchronous Transfer Mode
BRI	ISDN Basic Rate Interface, consisting of one D channel and two B channels.
BYTE	For the purposes of this document, the definition of a byte is 8 bits.
CALL MANAGEMENT	Refers to a process that is responsible for the setting up and tearing down of calls. This same process also controls the operational parameters of the call. The term “call,” and therefore “call management,” describes processes which refer to a higher level of call control, rather than those processes responsible for the physical connection.
CAPI	COMMON-ISDN-API
COMMUNICATION INTERFACE	Refers to a USB interface that identifies itself as using the Communication Class definition.
DATA INTERFACE	Refers to a USB interface that identifies itself as using the Data Class definition.
DCE	Data Circuit Terminating Equipment; for example, a modem or ISDN TA.
DEVICE MANAGEMENT	Refers to requests and responses that control and configure the operational state of the device. Device management requires the use of a Communication Class interface.
DIX	A networking cabling and signaling specification jointly developed by DEC, Intel and Xerox.
DSVD	Digital Simultaneous Voice and Data, signaling method mixes data and digitized voice.
DTE	Data Terminal Equipment; for example, a PC.
ENTITY	Family name for protocol building blocks. See Terminal and Unit below.
ETHERNET FRAME	Generic term representing any data frames that may be exchanged over DIX or 802.3 networks.
HEATHERINGTON ESCAPE SEQUENCE	A reliable technique used in modems to switch between data mode and command mode. Developed by Dale Heatherington, an employee of Hayes during the early 1980s. This is covered under United States patent number 4,549,302.
HEC	Header Error Control
ISDN	Integrated Services Digital Network.
ITU	International Telecommunications Union (formerly CCITT).

I.430	ISDN BRI physical interface standard. See ITU-T I.430 above
I.431	ISDN PRI physical interface standard. See ITU-T I.431 above
MANAGEMENT ELEMENT	Refers to a type of USB pipe that manages the communication device and its interfaces. Currently, only the Default Pipe is used for this purpose.
MASTER INTERFACE	A Communication Class interface, which has been designated the master of zero or more interfaces that implement a complete function in a USB communication device. This interface will accept management requests for the union.
NI-1	National ISDN 1 is intended to be a set of standards, which every manufacture can conform to for building switch independent ISDN devices. Future standards, denoted as NI-2 and NI-3, are currently being developed.
NOTIFICATION ELEMENT	Refers to a type of USB pipe. Although a notification element is not required to be an interrupt pipe, a notification element is typically defined in this way.
PDU	Protocol Data Unit - A combination of the SDU and the current protocol layer's header and/or trailer.
POTS	Plain Old Telephone Service. *See* PSTN.
PRI	Primary Rate Interface, which consists of one or two D channels and up to 30 B channels.
PSTN	Public Switched Telephone Network.
Q.921	ISDN data link layer protocol. See ITU-T Q.921 above
Q.922	ISDN data link layer protocol. See ITU-T Q.922 above
Q.931	ISDN call control protocol. See ITU-T Q.931 above
Q.2931	B-ISDN User Network Interface Layer 3 Specification. See ITU-T Q.2931 above.
SDU	Service Data Unit – User data and control information created at the upper protocol layers that is transferred transparently through a primitive by layer (N+1) to layer (N) and subsequently to (N-1).
T.30	Protocol for sending faxes over PSTN. See ITU-T T.30 above.
TA	Terminal Adapter, which is the equivalent of a modem for ISDN.
TERMINAL	Entity that represents a starting/ending point for a protocol stack.
TIA	Telecommunications Industry Association.
TIES	Time Independent Escape Sequence, which is an alternative method to the Heatherington Escape Sequence for switching between command mode and data mode on an analog modem. This was developed by a group of modem manufacturers in 1991.
UNION	A relationship between a collection of one or more interfaces that can be considered to form a functional unit.
UNIT	Entity that provides the basic building blocks to describe a protocol stack.
VIDEO PHONE	A device which simultaneously sends voice and video with optional data.

V.25TER	This is the ITU-T standard for Serial Asynchronous Automatic Dialing and Control, which is commonly known as the “AT” command set.
V.4	This is the ITU-T standard for general structure of signals of international alphabet number 5 code for data transmission over the public telephone network.
V.42	V.42*bis* data link layer protocol. See ITU-T V.42 above
V.42BIS	Data compression procedures. See ITU-T V.42*bis* above
V.110	Rate adoption procedures. See ITU-T V.110 above
V.120	Rate adoption procedures. See ITU-T V.120 above
X.25	See ITU-T X.25 above.
X.75	Packet switched signaling layer for PSTN. See ITU-T X.75 above

2. Management Overview

Several types of communication devices can benefit from the USB. This specification provides models for telecommunication devices, such as telephones, analog modems, ISDN devices and networking devices. It describes:

- Specifications for:
  - Communication Device Class
  - Communication Interface Class
  - Data Interface Class
- Framework for building a communication device:
  - Assembling the relevant USB logical structures into configurations.
  - Communication Class interface and its usage.
  - Data Class interface and its usage.
  - Usage of additional class types or vendor specific interfaces.
- Implementation examples of communication devices, such as a basic telephone, and an analog modem.

3. Functional Characteristics

This section describes the functional characteristics of the Communication Device Class, Communication Interface Class and Data Interface Class, including:

- Device organization:
  - Endpoint requirements.
  - Constructing interfaces from endpoints.
  - Constructing configurations from a variety of interfaces, some of which are defined by other class specifications.
  - Identifying groups of interfaces within configurations that make functional units and assigning a master interface for each union.
- Device operation

Although this specification defines both the Communication Interface Class and Data Interface Class, they are two different classes. All communication devices shall have an interface using the Communication Class to manage the device and optionally specify themselves as communication devices by using the Communication Device Class code. Additionally, the device has some number of other interfaces used for actual data transmission. The Data Interface Class identifies data transmission interfaces when the data does not match the structure or usage model for any other type of class, such as Audio.

3.1 Device Organization

A communication device has three basic responsibilities:

- Device management
- Operational management
- Data transmission

The device shall use a Communication Class interface to perform device management and optionally for call management. The data streams are defined in terms of the USB class of data that is being transmitted. If there is no appropriate USB class, then the designer can use the Data Class defined in this specification to model the data streams.

Device management refers to the requests and notifications that control and configure the operational state of the device, as well as notify the host of events occurring on the device.

Call management refers to a process that is responsible for the setting up and tearing down of calls. This same process also controls the operational parameters of the call. The term “call,” and therefore “call management,” describes processes that refer to a higher level of call control than those processes responsible for the physical connection.

Data transmission is accomplished using interfaces in addition to the Communication Class interface. These interfaces can use any defined USB class or can be vendor-specific.

3.1.1 Communication Device Management

There are two levels of device management for communication devices. The most basic form of device management results from control transfers made on endpoint 0 as outlined in Chapter 9 of the *USB Specification*. Device management is also required at a higher level, which is specific to communication devices. An example would be configuration of country-specific details for proper configuration of the telephone services.

To allow device management at the communication device level, a Union shall be made between all the interfaces that make up the functional unit of the device. A functional descriptor is used to define the group of interfaces that make up a functional unit within a device and is outlined in Section 5.2.3.8, “Union Functional Descriptor,” of this specification.

With the increasing popularity of multi-channel devices, a new class of device may need to expose multiple device management interfaces for device management at the communication device level. This would allow individual control of the multiple channels, such as an ISDN device. In this case, the Union would be between the Communication Class interface providing call control and the various interfaces it was managing at the moment.

3.2 Device Operation

Communication devices present data to the host in a form defined by another class, such as Audio, Data, or HID Interface. To allow the appropriate class driver to manage that data, the host is presented with one or more interfaces, as specified for that class. The interfaces required may change according to events that are initiated by the user or the network during a communication session: for example, the transition from a data only call to a data and voice call.

To allow the host to properly deal with the situation where multiple interfaces are used to create a single function, the device can optionally identify itself at the device level with the Communication Device Class code. This allows the host, if needed, to load any special drivers to properly configure the multiple interfaces into a single function in the host.

Note: In the case where the device does not choose to identify itself at the device level with the Communication Device Class code, the device shall employ a USB Common Class Feature mechanism that associates multiple interfaces on the device with a single driver in the host. As of December 15, 1998, this feature was still under development within the Common Class Working Group.

Static characteristics of the device, such as the physical connections, are described in terms of the USB device, interface, and endpoint descriptors. The data that moves over the physical interfaces is dynamic in nature, causing the characteristics of the interfaces to change as the data requirements change. These dynamic changes are defined in terms of messages transmitted between the device and host over the Communication Class interface. The device can use a standard or proprietary mechanism to inform its host software when an interface is available and what the format of the data will be. The host software can also use this same mechanism to retrieve information about data formats for an interface and select a data format when more than one is available.

3.3 Interface Definitions

Two classes of interfaces are described in this specification: Communication Class interfaces and Data Class interfaces. The Communication Class interface is a management interface and is required of all communication devices. The Data Class interface can be used to transport data whose structure and usage is not defined by any other class, such as Audio. The format of the data moving over this interface can be identified using the associated Communication Class interface.

3.3.1 Communication Class Interface

This interface is used for device management and, optionally, call management. Device management includes the requests that manage the operational state of the device, the device responses, and event notifications. Call management includes the requests for setting up and tearing down calls, and the managing of their operational parameters.

The Communication Class defines a Communication Class interface consisting of a management element and optionally a notification element. The management element configures and controls the device, and consists of endpoint 0. The notification element transports events to the host, and in most cases, consists of a interrupt endpoint.

Notification elements pass messages via an interrupt or bulk endpoint, using a standardized format. Messages are formatted as a standardized 8-byte header, followed by a variable-length data field. The header identifies the kind of notification, and the interface associated with the notification; it also indicates the length of the variable length portion of the message.

The Communication Class interface shall provide device management by furnishing a management element (endpoint 0); the interface optionally can provide host notification by furnishing a notification element. Only the management element is required for a complete Communication Class interface. The management element also meets the requirements for devices as outlined in the *USB Specification*. Call management is provided in the communication

interface and optionally multiplexed on a data interface. The following configurations describe how the device might provide call management with and without the use of the Communication Class interface:

- The device does not provide any call management on the Communication Class interface and is made up of only a management element (endpoint 0). In this case, the Communication Class interface is minimally represented and only provides device management over a management element (endpoint 0). This corresponds to the Multi-Channel Control Model, as described in Section 3.7.1, and the CAPI Control Model as described in Section 3.7.2.

- The device does not provide an internal implementation of call management and only accepts minimum set of call management commands from the host. In this case, both a management element and a notification element represent the Communication Class interface. This corresponds to the Direct Line Control Model, as described in Section 3.6.1 “Direct Line Control Model.”

- The device provides an internal implementation of call management over the Data Class interface but not the Communication Class interface. In this case, the Communication Class interface is also minimally represented and only provides device management over a management element (endpoint 0). This configuration most closely corresponds to the Abstract Control Model in which commands and data are multiplexed over the Data Class interface. Activation of the command mode from data mode is accomplished using the Heatherington Escape Sequence or the TIES method. For more information about the Abstract Control Model, see Section 3.6.2, “Abstract Control Model.”

- The device provides an internal implementation of call management that is accessed by the host over the Communication Class interface. In this case, the Communication Class interface performs both call and device management, and consists of a management element (endpoint 0) and a notification element (normally a interrupt endpoint). The management element will transport both call management and device management commands. The notification element will transport asynchronous event information from the device to the host, such as notification of an available response, which then prompts the host to retrieve the response over the management element. This corresponds to the Abstract Control Model. For more information about the Abstract Control Model, see Section 3.6.2, “Abstract Control Model.”

3.3.2 Data Class Interface

The Data Class defines a data interface as an interface with a class type of Data Class. Data transmission on a communication device is not restricted to interfaces using the Data Class. Rather, a data interface is used to transmit and/or receive data that is not defined by any other class. This data could be:

- Some form of raw data from a communication line.

- Legacy modem data.

- Data using a proprietary format.

At this time, it is the responsibility of the host software and device to communicate with each other over some other interface (such as a Communication Class interface) to determine the appropriate format to use. As more complicated communication devices are defined, it may become necessary to define a method of describing the protocol used within the Data Class interface. The attributes of a Data Class interface are as follows:

- The Interface descriptor uses the Data Class code as its class type. This is the only place that the Data Class code is to be used.

- The data is always a byte stream. The Data Class does not define the format of the stream, unless a protocol data wrapper is used.

- If the interface contains isochronous endpoints, on these endpoints, the data is considered synchronous.

- If the interface contains bulk endpoints, on these endpoints, the data is considered asynchronous.

Isochronous pipes are used for data that meets the following criteria:

- Constant bit rate.
- Real-time communication that requires low latency.

In general, isochronous endpoints can be used where raw information (either sampled or direct) from the network is sent to the host for further processing and interpretation. For example, an inexpensive ISDN TA could use an isochronous pipe for transport of the raw-sampled bits off a network connection. In this case, the host system would be responsible for the different network protocol that makes up an ISDN connection. This type of interface shall only be used in situations in which an Audio Class interface would not provide the necessary definitions or control.

The type and formatting of the media to be used is specified via messaging over the management element of a Communication Class interface when the host activates an interface or the device requests that an interface be activated. The bandwidth of the pipe is defined by the Endpoint descriptors and can be changed by selecting an alternate interface of an appropriate bandwidth.

3.3.2.1 Protocol Data Wrapper

To support embedded high-level protocols in a device, the data and commands between host and device must retain their order. This ensures that a protocol stack that is designed to run in a real time operating system can be split into two parts running in separate devices. Therefore, commands and data for a protocol have to be multiplexed onto the same interface using a wrapper; this wrapper also has the facility to send data to any layer of the stack. Each protocol specifies how to define protocol-specific commands and data fields going across its upper interface edge.

The host and device agree upon the wrapper feature at the time the protocol of the data is established. It is the responsibility of the host software and the device to communicate with each other over some other interface (such as a Communication Class interface) to determine the protocol. The wrapper is not used if there is no protocol established. It is optional to use the wrapper if the established protocol could use it; it is mandatory to use the wrapper if the protocol requires it.

To enable the different types of protocol stacks found on communication devices, two general forms have been defined for the data wrapper header as defined in Table 1. The structure for both forms is the same, the only difference is the usage of the source protocol ID. If no source protocol is needed or known, then offset 3, *bSrcProtocol* is set to 00h. The second form of the data wrapper header allows for both a source and destination protocol for the more structured protocol stack where both a source and destination protocol are needed.

Both data wrapper forms impose no restrictions on the data format, beyond the general requirement that the data is byte data. In any case where the source protocol is unneeded or unknown the source protocol ID (*bSrcProtocol*) of 00h is used.

Note: Use of a Protocol Data Wrapper on an isochronous pipe is not recommended, because of the possible loss of data because of the unreliable nature of isochronous pipes.

Table 1: Data Class Protocol Wrapper Layout

Offset	Field	Size	Value	Description
0		*2*	*Number*	*Size of wrapper in bytes*
2	*bDstProtocol*	*1*	*Protocol*	*Destination protocol ID.*
3	*bSrcProtocol*	*1*	*Protocol*	*Source protocol ID.*
4	*BData0*	*1*	*Number*	*First data bytes*
…	*…*	*…*	*…*	
N+3	*BDataN-1*	*1*	*Number*	*Nth data byte*

3.4 Endpoint Requirements

The following sections describe the requirements for endpoints in Communication Class or Data Class interfaces.

3.4.1 Communication Class Endpoint Requirements

The Communication Class interface requires one endpoint, a management element. It optionally can have an additional endpoint, the notification element. The management element uses the default endpoint for all standard and Communication Class-specific requests. The notification element normally uses an interrupt endpoint.

3.4.2 Data Class Endpoint Requirements

The type of endpoints belonging to a Data Class interface are restricted to being either isochronous or bulk, and are expected to exist in pairs of the same type (one In and one Out).

3.5 Device Models

Particular USB communication device configurations are constructed from the interfaces described in previous sections and those described by other class specifications. All communication devices consist of a Communication Class interface plus zero or more other data transmission interfaces, adhering to some other USB class requirements or implemented as vendor-specific interfaces. For example, the following descriptors are appropriate for a communication device:

- Device descriptor contains the class code of the Communication Device Class, defined in Table 14. Optionally, the device descriptor contains a class code of 00h, which indicates that the host should look at the interfaces to determine how to use the device.

- An Interface descriptor with the Communication Class code, which contains a management element and optionally a notification element.

- Zero or more other interfaces with class codes of various types such as Audio, Data, etc.

The device models outlined in the following sections are divided into several categories. As this specification develops, other models will be added. The term *model* describes a type of device and the interfaces that make it up. The term *control model* describes the type of Communication Class interface being used and is assigned a SubClass code for that interface. A control model can be used in several device models in which the method of device control and call management are similar.

3.6 USB POTS Models

A USB telephony device used on a POTS line has several types of interfaces that could be presented to the host. The arrangement and use of those different interfaces depends upon the type of POTS telephony device and the basic model used to build the device.

The difference between the various models of telephony devices can be divided according to the amount of processing the device performs on the analog signal before presenting it to the host. To help illustrate how the different types of interfaces could be put together to build a USB POTS telephony device, three example models are presented in the following sections.

Note: In many cases, a Data Class interface might not be used to present data to the host. Where the USB device is constructed with minimal intelligence, some analog class-specific interface control codes are required.

3.6.1 Direct Line Control Model

The Direct Line Control Model contains two examples: the Direct Line Model (or DL Model) and the Datapump Model.

A Communication Class interface of type Direct Line Control Model will consist of a minimum of two pipes; one is used to implement the management element and the other to implement a notification element. In addition, the device can use two or more pipes to implement channels over which to carry vendor-specific data.

3.6.1.1 DL Model

The DL Model is the simplest type of connection to a POTS line. At this level, the USB device is only converting the analog POTS line signal to digital data and presenting it to the USB bus. The modem modulation protocol (e.g. V.34, V.32bis) is implemented in the host. Instead of using the Data Class, the Audio Class is used to present the digitally converted data to the host. This type of connection could also be useful for a voice-only device, such as an answering machine.

Because the DL Model is the simplest, it provides a perfect example of why a device requires the Direct Line Control Model control codes. The key feature of a DL Model device is low cost, so reducing the processing power requirements on the USB device is essential. The DL Model uses a Direct Line Control Model SubClass code in the descriptor definition of its Communication Class interface.

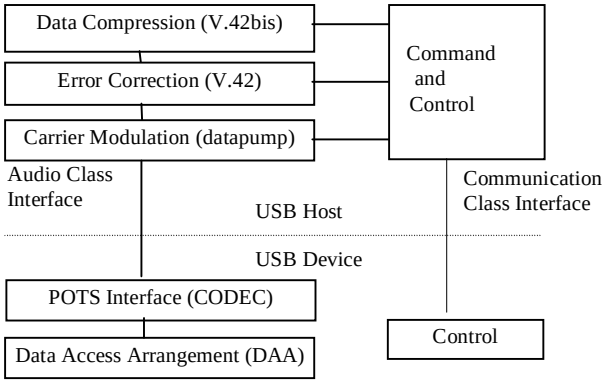


Figure 1: DL Model

These requests for controlling the interface between the USB device and the POTS line are presented in Table 2. There are also some additional signals that fall outside the analog phone signal which shall go back to the host as notifications, which are represented in Table 3. These requests and notifications are transported via the Communication Class interface for the device.

Table 2: Requests — Direct Line Control Model*

Request	Code	Description	Req'd/Opt
SET_AUX_LINE_STATE	10h	Request to connect or disconnect secondary jack from POTS circuit or CODEC, depending on hook state.	Optional
SET_HOOK_STATE	11h	Select relay setting for on-hook, off-hook, and caller ID.	Required
PULSE_SETUP	12h	Initiate pulse dialing preparation.	Optional
SEND_PULSE	13h	Request number of make/break cycles to generate.	Optional
SET_PULSE_TIME	14h	Setup value for time of make and break periods when pulse dialing.	Optional

Request	Code	Description	Req'd/Opt
RING_AUX_JACK	15h	Request for a ring signal to be generated on secondary phone jack.	Optional

*These requests are specific to the Communication Class.

The only class-specific request codes that are valid for a Communication Class interface with a Communication Class SubClass code of Direct Line Control Model are listed in the previous Table 2. The other class-specific requests not listed in the previous table, such as SEND_ENCAPSULATED_COMMAND, are inappropriate for a Direct Line Control Model and shall generate a STALL condition if sent to such an interface. For example, hanging up the line would be accomplished by using SET_HOOK_STATE, rather than by sending "ATH" via SEND_ENCAPSULATED_COMMAND.

Table 3: Notifications — Direct Line Control Model*

Notification	Code	Description	Req'd/Opt
AUX_JACK_HOOK_ STATE	08h	Indicates hook state of secondary device plugged into the auxiliary phone jack.	Optional
RING_DETECT	09h	Message to notify host that ring voltage was detected on POTS interface.	Required

* These notifications are specific to the Communication Class.

The only class-specific notification codes, which are valid for a Communication Class interface with a Communication Class SubClass code of Direct Line Control Model, are listed in the previous table. The other class-specific notifications not listed in the previous table, such as RESPONSE_AVAILABLE, are inappropriate for a Direct Line Control Model and shall not be sent by such a device.

3.6.1.2 Datapump Model

The Datapump view of the device is the next logical break and is similar to the DL Model. In the Datapump view, the USB device handles the carrier modulation instead of the host. Because there are no standard interfaces for Datapumps, and it would be difficult to generalize the I/O space and registers required, it is assumed a vendor-specific interface is employed based on the specifics of the Datapump being used.

The POTS line interface requests and notifications required for the Datapump USB device are the same as the DL Model as described in Table 2 and Table 3, so the Direct Line Control Model SubClass code shall be used.

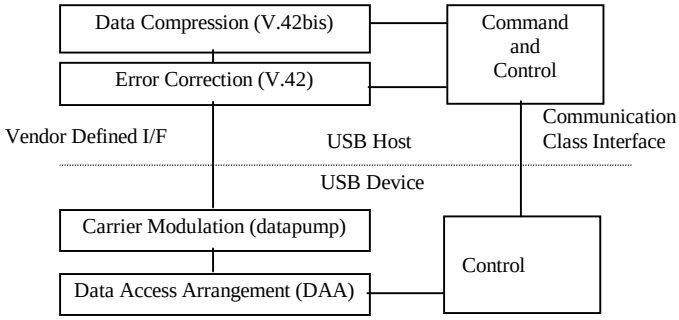


Figure 2: Datapump Model

3.6.2 Abstract Control Model

With an Abstract Control Model, the USB device understands standard V.25ter (AT) commands. The device contains a Datapump and micro-controller that handles the AT commands and relay controls. The device uses both a Data Class interface and a Communication Class interface. For an illustration of the use of both interfaces, see Figure 3. The device can also, at times, make use of other class interfaces; for example a device could use an Audio Class interface for the audio functions in a speakerphone.

A Communication Class interface of type Abstract Control Model will consist of a minimum of two pipes; one is used to implement the management element and the other to implement a notification element. In addition, the device can use two pipes to implement channels over which to carry unspecified data, typically over a Data Class interface.

For POTS line control, an Abstract Control Model shall either support V.25ter commands embedded in the data stream or V.25ter commands sent down the Communication Class interface. When V.25ter commands are multiplexed in the data stream, the Heatherington Escape Sequence or the TIES method would define the only supported escape sequences.

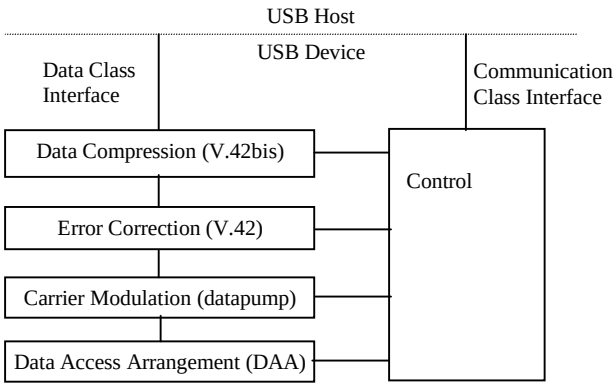


Figure 3: Abstract Control Model

Error correction and data compression could be implemented on the host, and not necessarily on the device. This type of device differs from the Direct Line Control Model, because the data from the USB device is presented to the host via a native class-defined interface rather than a vendor-specific Datapump interface. Also, V.25ter commands are used to control the POTS line interface. V.80 defines one way that the host can control the DCE data stream to accomplish this, but there are also proprietary methods.

3.6.2.1 Abstract Control Model Serial Emulation

The Abstract Control Model can bridge the gap between legacy modem devices and USB devices. To support certain types of legacy applications, two problems need to be addressed. The first is supporting specific legacy control signals and state variables which are addressed directly by the various carrier modulation standards. Because of these dependencies, they are important for developing an analog modem, which presents an Abstract Control Model type Communication Class interface to the host. To support these requirement additional requests (Table 4) and notifications (Table 5) have been created.

The second significant item which is needed to bridge the gap between legacy modem designs and the Abstract Control Model is a means to multiplex call control (AT commands) on the Data Class interface. Legacy modem designs are limited by only supporting one channel for both "AT" commands and the actual data. To allow this type of functionality, the device must have a means to specify this limitation to the host.

When describing this type of device, the Communication Class interface would still specify a Abstract Control Model, but call control would actually occur over the Data Class interface. To describe this particular characteristic, the Call Management Functional Descriptor (Section 5.2.3.2) would have bit D1 of *bmCapabilities* set.

For devices that support both modes, call control over the Communication Class interface and call control over a Data Class interface, and need to switch between them, then the ***GetCommFeature*** (Section 6.2.4) request is used to switch between modes.

Table 4: Requests — Abstract Control Model*

Request	Code	Description	Req'd/Opt
SEND_ENCAPSULATED_ COMMAND	00h	Issues a command in the format of the supported control protocol.	Required
GET_ENCAPSULATED_ RESPONSE	01h	Requests a response in the format of the supported control protocol.	Required
SET_COMM_FEATURE	02h	Controls the settings for a particular communication feature.	Optional
GET_COMM_FEATURE	03h	Returns the current settings for the communication feature.	Optional
CLEAR_COMM_FEATURE	04h	Clears the settings for a particular communication feature.	Optional
SET_LINE_CODING	20h	Configures DTE rate, stop-bits, parity, and number-of-character bits.	Optional +
GET_LINE_CODING	21h	Requests current DTE rate, stop-bits, parity, and number-of-character bits.	Optional +
SET_CONTROL_LINE_STATE	22h	RS-232 signal used to tell the DCE device the DTE device is now present.	Optional
SEND_BREAK	23h	Sends special carrier modulation used to specify RS-232 style break.	Optional

* These requests are specific to the Communication Class.
+ For an analog modem, it is strongly recommended to support these requests.

The only class-specific request codes that are valid for a Communication Class interface with a Communication Class SubClass code of Abstract Control Model are listed in the previous Table 4. The other class-specific requests not listed in the previous table, such as SET_HOOK_STATE, are inappropriate for an Abstract Control Model and would generate a STALL condition if sent to such an interface. For example, hanging up the line would be accomplished by sending "ATH" via SEND_ENCAPSULATED_COMMAND, rather than by using SET_HOOK_STATE.

Table 5: Notifications — Abstract Control Model*

Notification	Code	Description	Req'd/Opt
NETWORK_CONNECTION	00h	Notification to host of network connection status.	Optional +
RESPONSE_AVAILABLE	01h	Notification to host to issue a GET_ENCAPSULATED_RESPONSE request.	Required
SERIAL_STATE	20h	Returns the current state of the carrier detect, DSR, break, and ring signal.	Optional +

* These notifications are specific to the Communication Class.
+ For an analog modem, it is strongly recommended to support these requests.

The only class-specific notification codes, which are valid for a Communication Class interface with a Communication Class SubClass code of Abstract Control Model, are listed in the previous Table 5. The other class-specific notifications not listed in the previous table, such as RING_DETECT, are inappropriate for an Abstract Control Model and shall not be sent by such a device.

3.6.3 USB Telephone Model

A USB telephone device has a type of Communication Class interface that will be presented to the host, and it has the SubClass code of Telephone Control Model. A telephone device will not typically present a Data Class interface.

3.6.3.1 Telephone Control Model

Telephone devices with multiple lines will have a separate Communication Class interface for each physical line connected to the device. Each individual interface will correspond to a different physical line representing a network connection to the device.

Functional descriptors will be used to describe the various capabilities of a USB telephone device. These functional descriptors are defined in Section 5.2.3, “Functional Descriptors.”

A Communication Class interface of SubClass code Telephone Control Model will consist of a minimum of two pipes: one to implement the management element and the other to implement the notification element. This model describes the simplest version of a USB telephone device using only a Communication Class interface; other, more complicated implementations are possible.

To create more complicated implementations of a USB telephone device for example, use an Audio Class interface to provide the audio capabilities of a telephone and a Human Interface Device Class interface to provide the keypad capabilities of a telephone.

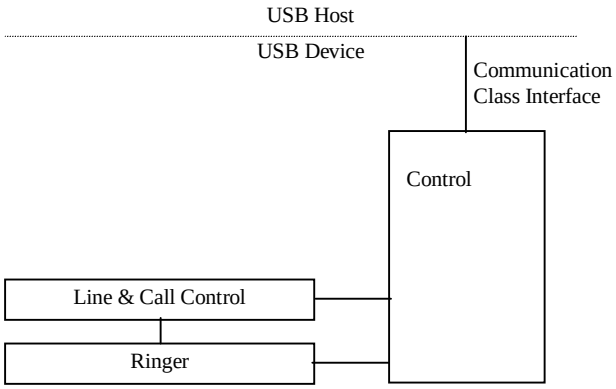


Figure 4: Telephone Control Model

The requests for controlling the USB telephone device via its Communication Class interface are presented in Table 6. Unsolicited messages from the USB telephone device to the host are sent using the notification element messages that are presented in Table 7. These requests and notifications are transported via the Communication Class interface for the device.

Table 6: Requests — Telephone Control Model*

Request	Code	Description	Req’d/Opt
SET_COMM_FEATURE	02h	Used to set a unique communication feature, which is normally specific to a particular device.	Optional
GET_COMM_FEATURE	03h	Returns the current settings for the communication feature.	Optional
CLEAR_COMM_FEATURE	04h	Clears the settings for a particular communication feature.	Optional
SET_RINGER_PARMS	30h	Configures the ringer for a telephone device.	Optional

Request	Code	Description	Req'd/Opt
GET_RINGER_PARMS	31h	Gets the current ringer configuration for a telephone device.	Required
SET_OPERATION_PARMS	32h	Configures the operational mode of the telephone.	Optional
GET_OPERATION_PARMS	33h	Gets the current operational mode of the telephone.	Optional
SET_LINE_PARMS	34h	Allows changing the current state of the line associated with the interface, providing basic call capabilities, such as dialing and answering calls.	Required
GET_LINE_PARMS	35h	Gets current status of the line.	Required
DIAL_DIGITS	36h	Dials digits on the network connection.	Required

* These requests are specific to the Communication Class.

Table 7: Notifications — Telephone Control Model*

Notification	Code	Description	Req'd/Opt
CALL_STATE_CHANGE	28h	Reports a state change on a call.	Required
LINE_STATE_CHANGE	29h	Reports a state change on a line.	Optional

* These notifications are specific to the Communication Class.

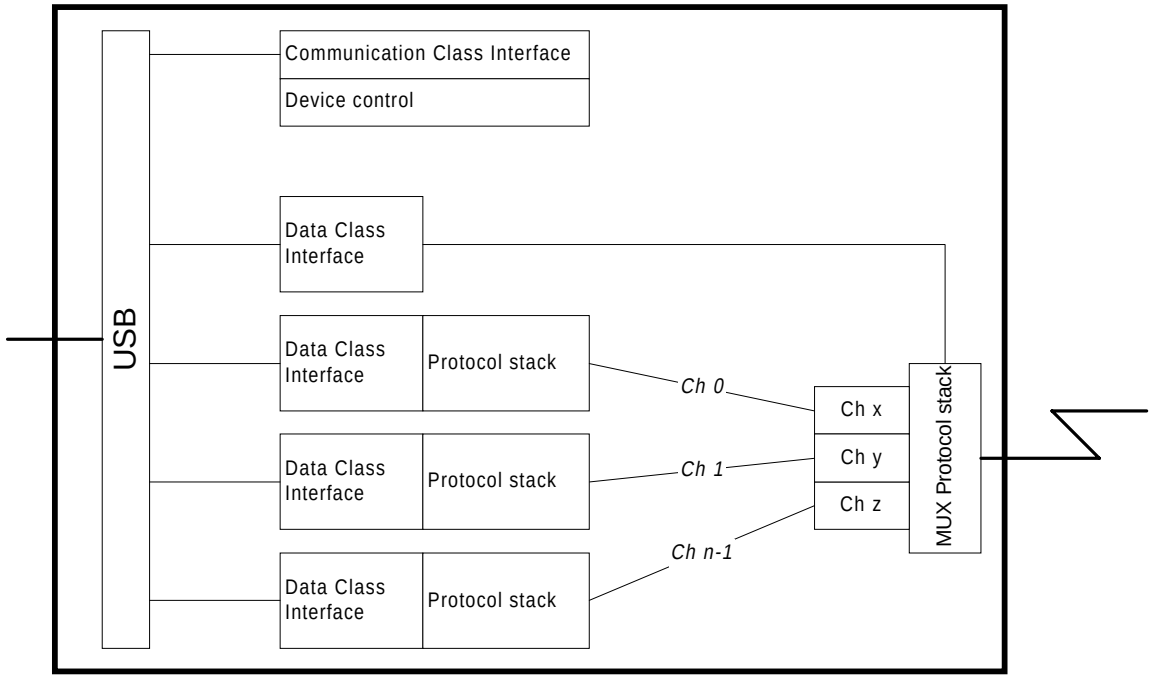
3.7 USB ISDN Models

An ISDN network provides several channels that an USB ISDN device may present to a host. They consist of a call control channel (D-Channel) and some data channels (B-Channels). Depending on functional requirements on the device, these channels may be presented to the host on separate Data Class interfaces using the Multi-Channel Model, or multiplexed onto one Data Class interface using the CAPI Model. Common for the two models are that the Communication Class interface is only used for device management.

3.7.1 Multi-Channel Model

A *Multi-Channel communication device* is defined as a communication device having a number of channels multiplexed on the physical network interface, where each channel has independent call control.

The prime characteristic of a Multi-Channel device is its ability to multiplex several channels on a physical network interface using a *MUX protocol stack*. Assuming there are *n* channels (*x .. z*) on the physical interface, where *n* is network and device specific, physical channels are mapped to a standard set of channels (*0 .. n-1*) by the protocol stack. The standard channels carrying data with unspecified format are then explicitly mapped to some USB interface (See Section 5.2.3.11 for more details). The protocol stack may also expose an USB interface for protocol management.



Before a channel is presented to USB it is assigned a Class interface. A Data Class interface has to run a protocol stack on the channel in order to define what data the channel carries. **Note:** A single protocol is considered a protocol stack and framing is considered a protocol. A *Protocol Data Wrapper* should be used if access to any protocols in the protocol stack is needed. The *Multi-Channel* model is based on the Open Systems Interconnection (OSI) model which is a layered architecture to structure data communication. The OSI construct is implemented for example in ISDN communication and TCP/IP protocols (for Internet access), as well as local area networks.

The basic idea of the OSI model is a hierarchical structure of functions necessary for communication. The OSI model defines 7 layers for handling communication procedures. These layers communicate on a peer-to-peer basis by using a fixed protocol. A communication layer n uses the services of layer n-1 to transfer data and information to a peer entity. Interlayer service access points and connection endpoints provide the means for the transfer of protocol primitives (data, commands and notifications) between the layers.

Though in theory, the OSI model strictly separates the different layers and assigned functions, in practice functional units may not be exactly assigned to a definite layer and partly span one or two layers.

A *Multi-Channel communication device* uses for device management a Communication Class interface with a Communication Class SubClass code of Multi-Channel. The only class-specific request codes that are valid for this SubClass code are listed in Table 8. All other class-specific requests not listed in the table are inappropriate for an Multi-Channel Model and would generate a STALL condition if sent to such an interface.

Table 8: Requests — Multi-Channel Model*

Request	Code	Description	Req'd/Opt
SET_UNIT_ PARAMETER	37h	Used to set a Unit specific parameter	Optional
GET_UNIT_ PARAMETER	38h	Used to retrieve a Unit specific parameter	Required

Request	Code	Description	Req'd/Opt
CLEAR_UNIT_ PARAMETER	39h	Used to set a Unit specific parameter to its default state.	Optional

* These requests are specific to the Communication Class.

3.7.1.1 Topology

To be able to manipulate the properties of a protocol stacks, its functionality must be divided into addressable Entities. Two types of such generic Entities are identified and are called Units and Terminals. Protocol stacks are built by connecting together several of these Entities to form the required topology. These Entities may be connected in a many to one or one to many fashion in order to "bond" channels or share a channel among many interfaces.

- **Units:**
  Units are Entities that provide the basic building blocks to describe different protocol stacks. There are two kinds of Units: Protocol and Extension. Protocol Units identify instances of protocols defined in this document. Extension Units are vendor specific extensions to this set. Each Unit has one or more Child pins used to connect to its immediate neighboring Unit or Terminal below it on the stack.

- **Terminals:**
  Terminals are Entities that represents a starting/ending point for a protocol stack. It is used to interface between the 'outside world' and Units in the protocol stack and serves as a receptacle for data flowing in and out. There are two kinds of Terminals: USB and Network Channel Terminals. USB Terminals are those on the top of the protocol stack. Network Channel Terminals are those on the bottom end of the stack. The USB Terminal has one or more Child pins used to connect to its immediate neighboring Unit or Terminal below it on the stack. The Network Channel Terminal, having no Unit or Terminal below it, has no Child pins.

Each Unit and Terminal within a Configuration is assigned a unique identification number, the EntityID, contained in the *bEntityID* field of the Unit and Terminal descriptor. The value 0x00 is reserved for undefined ID's, effectively restricting the total number of addressable Entities (both Units and Terminals) to 255.

Besides uniquely identifying all addressable Entities, the ID's are also used to describe the topology of the protocol stack(s); i.e. the *bChild* of a Unit or USB Terminal descriptor indicates to which other lower Unit or Terminal this one is connected.

A protocol stack can be thought of as some number of Units and Terminals connected together, with the upper most unit exposed as a USB interface and the lower most unit connected to the actual device hardware. By selecting an interface, either during configuration or by setting an alternate interface, you enable the protocol stack. Taking this concept further, if you defined an optional alternate interface with no endpoints (must be number zero), this can be used to relinquish bandwidth (for Isochronous endpoints) and at the same time move a stack into a deactivated state. By moving from a configured protocol stack interface to an alternate interface, with no endpoints, you deactivate the protocol stack.

If the default alternate interface zero is used, with no endpoints, the stack will start from a deactivated state and will need to be activated when needed. When the Unit is activated, its state is reset. When a unit is deactivated it will not respond to any message sent to it from Entities above or below.

3.7.2 USB CAPI Model

A USB CAPI device has a single type of Communication Class interface that will be presented to the host and it will have the SubClass code of a CAPI Control Model. A USB CAPI device will present a Data Class interface which is used to exchange CAPI messages. The CAPI Control Model do not use a notification element. CAPI provides an abstraction of services which is independent from the underlying network. Multiple lines, if provided by the underlying network, will be presented through one single interface and controlled and managed via CAPI messages.

The definition of CAPI covers all network relevant details such as call-management and protocol-relevant issues where appropriate. The management and data information are part of the CAPI messages which are by definition operating system independent. The USB CAPI Model supports both intelligent and simple CAPI device designs.

3.7.2.1 CAPI Control Model

With a CAPI Control Model, the USB device understands CAPI commands and CAPI messages. The device will make use of both a Data Class Interface and a Communication Class interface, see Figure 5.

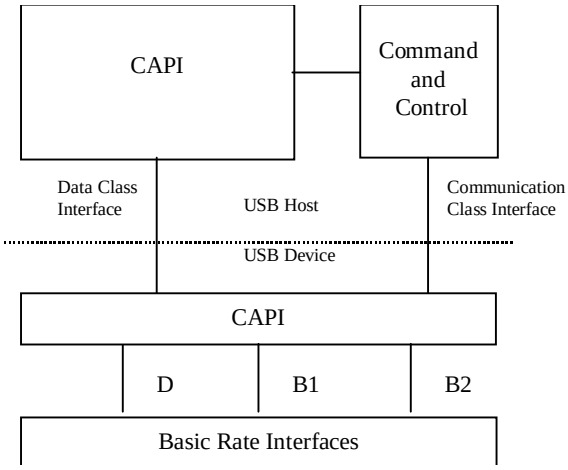


Figure 5: CAPI Control Model for a Basic-Rate Configuration

The CAPI functionality is divided into two parts as shown in the diagram above. Dividing CAPI into two parts allows for different CAPI device designs. Intelligent CAPI devices handle the call management according to the underlying network, for example Q.931 or NI-1 for ISDN, as well as a full set of protocols within the data channels, for example X.75, V.120, V.110, V.42bis, T.30 etc. on the USB device itself. For these devices the CAPI part within the USB device is powerful and is usually loaded as firmware on startup on the device. A firmware download to a device is done through manufacturer specific operations. Simple CAPI devices implement some low layer functionality, usually the direct network interface only. These devices enable the creation of low-cost solutions and require the host to do the bulk of the protocol processing. These simple USB devices are also managed and controlled by CAPI messages.

All messages exchanged between the host and CAPI consist of a fixed-length header and a parameter area of variable length. The message length is stored at the beginning of the fixed-length header thus enabling adaptive drivers to forward CAPI messages without further knowledge of the internal CAPI message format. The messages carry all management and data information for the CAPI device. These messages are exchanged via the Data Interface of the device. The CAPI message stream encapsulates all types of data a connection of the underlying network can carry. If conversions to other data formats are necessary these can be accomplished within the host and done in software. In such a way the support of an audio interface is included. This approach enables a integration into the framework of abstract host interfaces as well as the support of existing CAPI applications.

The CAPI device reports its implemented functionality using a Communication Class specific request, in order to allow the host to choose an appropriate upper part of the CAPI to run within the host. The request is transported via the Communication Class interface for the device and presented in Table 9.

Table 9: Requests — CAPI Control Model*

Request	Code	Description	Req'd/Opt
GET_PROFILE	3Ah	Returns the implemented capabilities of the device	Required

* These requests are specific to the Communication Class.

The only class specific request which is valid for a Communication Class interface with a Communication Class SubClass code of the CAPI Control Model is listed in Table 9 above. The other class specific requests not listed in the above table, such as SEND_ENCAPSULATED_COMMAND, are inappropriate for a CAPI Control Model and shall generate a STALL condition if sent to such an interface.

3.8 USB Networking Models

A USB Networking device has a type of Communication Class Interface that will be presented to the host for configuring and managing the networking device. Networking devices are typically "Always Connected", spending all of their time with the "link up". The Communication Class Interface is primarily used to configure and manage the networking device, not to place calls.

In contrast to a telecommunications device, a networking device will always have at least one associated Data Class interface to exchange network traffic. In a typical host software stack, the same driver that is responsible for configuring and managing the network device is also the client that is a source/sink of networking traffic. An example of such a host resident networking driver is either an ATM or Ethernet driver.

The usage of more than one pair of Communication/Data interfaces will be common for devices that expose more than one interface to the network. For example, an MCNS DOCSIS cable modem has two 48 bit MAC addresses. Its first MAC address is used to manage cable modem functions, while the second MAC address is used for subscriber access to the Internet. For a cable modem implementation where its management functions are migrated to the host, two different Communication Class and Data Class interfaces will be exposed to the host (one pair for each of the 2 MAC addresses).

Networking devices are differentiated by their SubClass code, which currently are comprised of the Ethernet Networking Control Model and ATM Networking Control Model.

3.8.1 Common Data Plane Characteristics

The core Data-In/Data-Out pipe mechanism is the same for all networking device models supported by this specification, independent of the media type (e.g., Cable, xDSL, Ethernet) or media data type (e.g., ATM cells, Ethernet frames).

Typical USB-based Networking devices will support bulk transfers as the default configuration to exchange data between a host and the USB device.

While each data packet of a bulk endpoint is limited to the maximum packet size defined in the associated endpoint descriptor, it should be noted that a host might request multiple bulk USB protocol packets within a single USB frame. For maximum throughput, a Networking device must be prepared to transfer multiple bulk packets within a single USB frame.

Some USB-based Networking device implementations may support isochronous data transfers in addition to (or instead of) bulk transfers. Isochronous transfers guarantee data throughput and bounded latency, consistent with the needs of real-time streams (audio, video). Isochronous data errors are reported to receiver, but no data integrity (i.e., retransmission) is provided by the USB link.

The Data Class Interface Descriptor protocol code for all Networking Control Models is 00h.

USB provides no inherent flow control mechanism for isochronous pipes, and this specification defines no higher level mechanism for doing so. Instead, it is assumed that the host software is responsible for doing traffic shaping as necessary to match any end-to-end negotiation. If the networking device is performing traffic shaping, then either a bulk endpoint should be used, or the flow control methods should be provided using vendor-specific methods.

The Data Class interface of a networking device shall have a minimum of two interface settings. The first setting (the default interface setting) includes no endpoints and therefore no networking traffic is exchanged whenever the default interface setting is selected. One or more additional interface settings are used for normal operation, and therefore each

includes a pair of endpoints (one IN, and one OUT) to exchange network traffic. Select an alternate interface setting to initialize the network aspects of the device and to enable the exchange of network traffic.

To recover the network aspects of a device to known states, select the default interface setting (with no endpoints) and then select the appropriate alternate interface setting. This action will flush device buffers, clear any filters or statistics counters and will cause NETWORK_CONNECTION and CONNECTION_SPEED_CHANGE notifications to be sent to the host. The effect of a "reset" on the device physical layer is media dependent and beyond the scope of this specification.

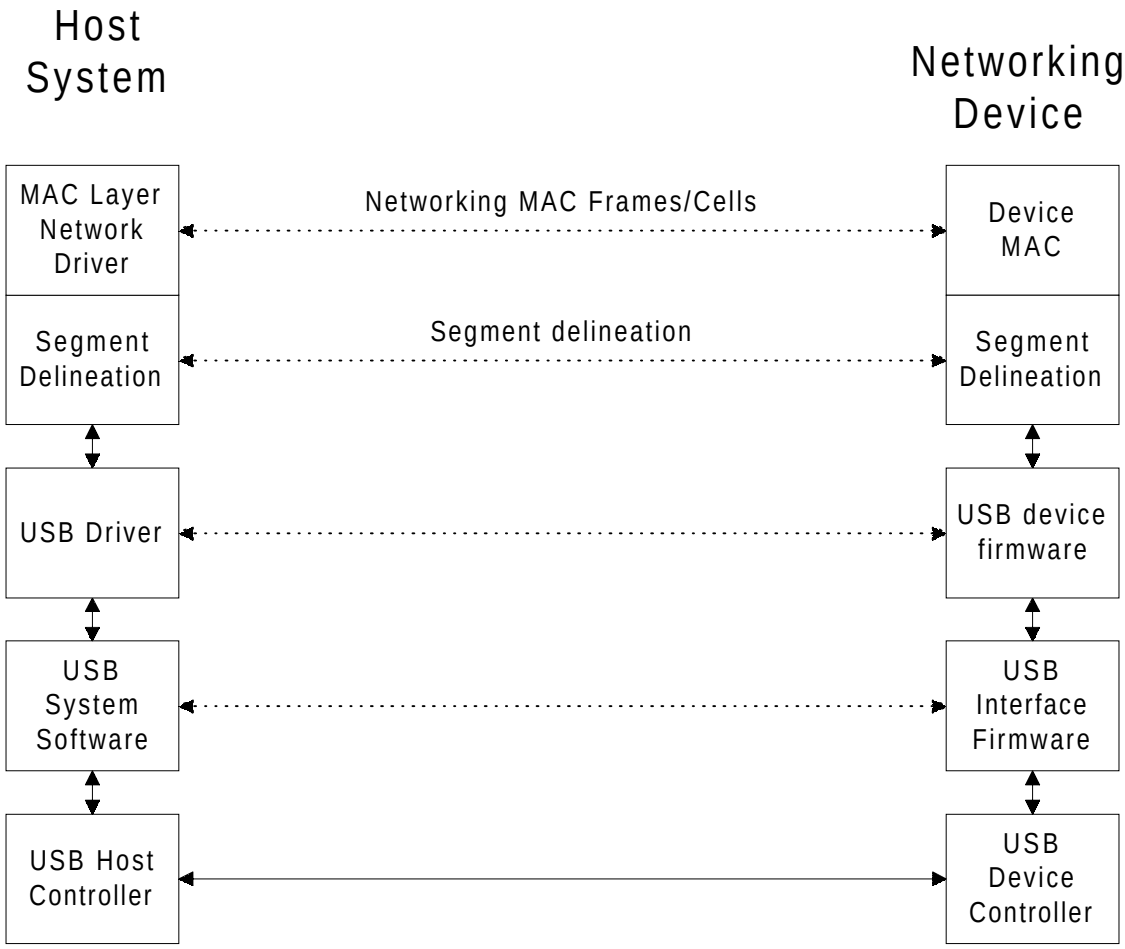


Figure 6 - USB Network Device Example

3.8.1.1 Segment Delineation

For almost any type of USB attached networking device, a mechanism is needed where both the networking device and the Host can delineate the beginning and ending of a *segment* within the data stream delivered by an endpoint. The meaning and cause of *segment* end is media dependent. Below are some examples:

- The end of an Ethernet frame
- The end of an AAL5 ATM SDU
- The end of an AAL5 ATM PDU

- The arrival of a high priority ATM cell

This positive delineation is done using a USB short packet mechanism. When a segment spans *N* USB packets, the first packet through packet *N-1* shall be the maximum packet size defined for the USB endpoint. If the *N*th packet is less than maximum packet size the USB transfer of this short packet will identify the end of the segment. If the *N*th packet is exactly maximum packet size, it shall be followed by a zero-length packet (which is a short packet) to assure the end of segment is properly identified.

When transmitting data to the networking device, it is assumed that the client of the host USB driver takes the appropriate actions to cause a short packet to be sent to the networking device. For segments with lengths that are an even multiple of the pipe's "max packet size", the ability to write a buffer of zero length is required to generate this short packet.

3.8.1.2 Segment Size

The host and the attached network device must negotiate to establish the maximum segment size. The upper limit for this is usually a function of the buffering capacity of the attached device, but there may be other factors involved as well.

For networking devices that exchange Ethernet frames, the size of a segment is also negotiable. Typical Ethernet frames are 1514 bytes or less in length (not including the CRC), but this could be longer (e.g., 802.1Q VLAN tagging).

An ATM oriented device moving cells is a different matter. In the case of AAL5 SDU exchanges, the segment size could be up to 64K bytes in length, and is a function of device buffering capacity and the results of end-to-end negotiation with Q.2931.

3.8.2 Ethernet Networking Control Model

The Ethernet Networking Control Model is used for exchanging Ethernet framed data between the device and host. A Communication Class interface is used to configure and manage various Ethernet functions, where an "Ethernet Networking Control Model" SubClass code is indicated in the descriptor definition of its Communication Class interface.

A Data Class interface is used to exchange Ethernet encapsulated frames sent over USB. These frames shall include everything from the Ethernet destination address (DA) up to the end of the data field. The CRC checksum must not be included for either send or receive data. It is the responsibility of the device hardware to generate and check CRC as required for the specific media. Receive frames that have a bad checksum must not be forwarded to the host. This implies that the device must be able to buffer at least one complete Ethernet frame.

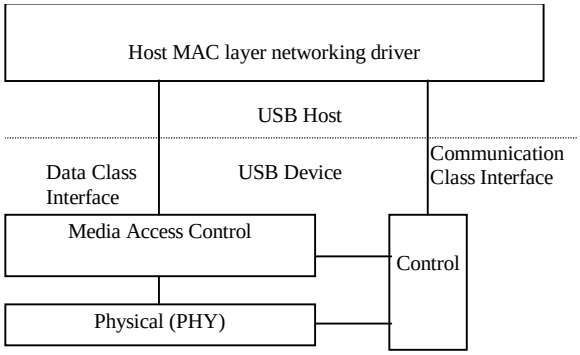


Figure 7: Ethernet Networking Model

Although a typical USB Networking device stays in an “always connected” state, some Networking device management requests are required to properly initialize both the device and the host networking stack. There also may be occasional changes of device configuration or state, e.g., adding multicast filters.

The only class-specific request codes that are valid for a Communication Class interface with a Communication Class SubClass code of Ethernet Networking Control Model are listed in Table 10.

Table 10: Requests — Ethernet Networking Control Model*

Request	Code	Description	Req’d/Opt
SEND_ENCAPSULATED_ COMMAND	00h	Issues a command in the format of the supported control protocol. The intent of this mechanism is to support networking devices (*e.g.,* host-based cable modems) that require an additional vendor-defined interface for media specific hardware configuration and management.	Optional
GET_ENCAPSULATED_ RESPONSE	01h	Requests a response in the format of the supported control protocol.	Optional
SET_ETHERNET_ MULTICAST_FILTERS	40h	As applications are loaded and unloaded on the host, the networking transport will instruct the device’s MAC driver to change settings of the Networking device’s multicast filters.	Optional
SET_ETHERNET_ POWER_MANAGEMENT_ PATTERN_FILTER	41h	Some hosts are able to conserve energy and stay quiet in a “sleeping” state while not being used. USB Networking devices may provide special pattern filtering hardware that enables it to wake up the attached host on demand when something is attempting to contact the host (*e.g.*, an incoming web browser connection). Primitives are needed in management plane to negotiate the setting of these special filters	Optional **
GET_ETHERNET_ POWER_MANAGEMENT_ PATTERN_FILTER	42h	Retrieves the status of the above power management pattern filter setting	Optional **
SET_ETHERNET_ PACKET_FILTER	43h	Sets device filter for running a network analyzer application on the host machine	Required
GET_ETHERNET_ STATISTIC	44h	Retrieves Ethernet device statistics such as frames transmitted, frames received, and bad frames received.	Optional

* These requests are specific to the Communication Class.
** If the SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER command is supported, then the GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER request must be supported as well.

The only class-specific notification codes, which are valid for a Communication Class interface with a Communication Class SubClass code of Ethernet Networking, are listed in the following table.

Table 11: Notifications — Ethernet and ATM Networking Control Models*

Notification	Code	Description	Req’d/Opt
NETWORK_CONNECTION	00h	Reports whether or not the physical layer (modem, Ethernet PHY, etc.) link is up.	Required
RESPONSE_AVAILABLE	01h	Notification to host to issue a GET_ENCAPSULATED_RESPONSE request.	Optional
CONNECTION_SPEED_ CHANGE	2Ah	Reports a change in upstream or downstream speed of the networking device connection.	Required

* These notifications are specific to the Communication Class.

3.8.3 ATM Networking Control Model

An ATM USB device is used to move ATM cells or AAL5 SDUs to and from the host. The segmentation and re-assembly (SAR) function in the ATM adaptation layer may be implemented on the host, and not necessarily on the device.

A Communication Class interface is used to configure and manage various ATM functions, where an "ATM Networking Control Model" SubClass code is indicated in the descriptor definition of its Communication Class interface. This Communication Class interface consists of a minimum of two pipes; one is used to implement the management element and the other to implement a notification element. A Data Class interface is used to exchange ATM cells or AAL5 SDUs sent over USB. The host may perform traffic shaping on an aggregated basis according to the upstream speed reported via the Communication Class interface.

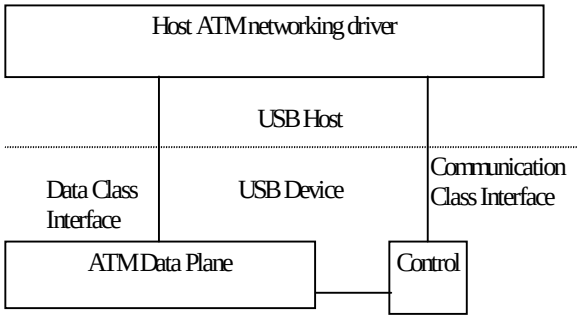


Figure 8: ATM Networking Model

These requests for controlling the interface between the USB ATM device are presented in the following table. There are also some additional signals that shall go back to the host as notifications, which are represented in Table 13. These requests and notifications are transported via the Communication Class interface for the device.

Table 12: Requests — ATM Networking Control Model*

Request	Code	Description	Req'd/Opt
SEND_ENCAPSULATED_ COMMAND	00h	Issues a command in the format of the supported control protocol. The intent of this mechanism is to support networking devices (*e.g.*, host-based cable modems) that require an additional vendor-defined interface for media specific hardware configuration and management.	Optional
GET_ENCAPSULATED_ RESPONSE	01h	Requests a response in the format of the supported control protocol.	Optional
SET_ATM_DATA_ FORMAT	50h	Chooses which ATM data format will be exchanged between the host and the ATM Networking device.	Required
GET_ATM_DEVICE_ STATISTICS	51h	Retrieves global statistics from the ATM Networking device.	Required
SET_ATM_DEFAULT_VC	52h	Pre-selects the VPI/VCI value for subsequent GetATMVCStatistics requests	Optional
GET_ATM_VC_ STATISTICS	53h	Retrieves statistics from the ATM Networking device for a particular VPI/VCI.	Optional

* These requests are specific to the Communication Class.

The only class specific request codes that are valid for a Communication Class interface with a SubClass code of ATM Networking Control Model are listed in the previous table. The other class specific requests not listed in the previous

table are inappropriate for an ATM Networking Control Model and must generate a STALL condition if sent to such an interface.

Table 13: Notifications — Ethernet and ATM Networking Control Models*

Notification	Code	Description	Req'd/Opt
NETWORK_CONNECTION	00h	Reports whether or not the physical layer (modem, Ethernet PHY, etc.) link is up.	Required
RESPONSE_AVAILABLE	01h	Notification to host to issue a GET_ENCAPSULATED_RESPONSE request.	Optional
CONNECTION_SPEED_ CHANGE	2Ah	Reports a change in upstream or downstream speed of the networking device connection.	Required

* These notifications are specific to the Communication Class.

The only class specific notification codes, which are valid for a Communication Class interface with a SubClass code of ATM Networking Control Model, are listed in the previous table. The other class specific notifications not listed in the previous table are inappropriate for a ATM Networking Control Model and shall not be sent by a device.

4. Class-Specific Codes for Communication Devices

This section lists the codes for the Communication Device Class, Communication Interface Class and Data Interface Class, including subclasses and protocols. These values are used in the *bDeviceClass*, *bInterfaceClass*, *bInterfaceSubClass*, and *bInterfaceProtocol* fields of the standard device descriptors as defined in chapter 9 of the *USB Specification.*

4.1 Communication Device Class Code

The following table defines the Communication Device Class code:

Table 14: Communication Device Class Code

Code	Class
02h	Communication Device Class

4.2 Communication Interface Class Code

The following table defines the Communication Class code:

Table 15: Communication Interface Class Code

Code	Class
02h	Communication Interface Class

4.3 Communication Interface Class SubClass Codes

The following table defines the SubClass codes for the Communication Interface Class:

Table 16: Communication Interface Class SubClass Codes

Code	SubClass
00h	RESERVED
01h	Direct Line Control Model
02h	Abstract Control Model
03h	Telephone Control Model
04h	Multi-Channel Control Model
05h	CAPI Control Model
06h	Ethernet Networking Control Model
07h	ATM Networking Control Model
08h-7Fh	RESERVED (future use)
80h-FEh	RESERVED (vendor specific)

The Datapump Model, as described in Section 3.6.1.2, “Datapump Model,” is not listed in Communication Class SubClass codes, because a device of that type will use a Direct Line Control Model for POTS line control and a vendor-specific interface.

4.4 Communication Interface Class Control Protocol Codes

A communication control protocol is used by the USB host to control communication functions in the device or on the network. This specification defines code values for certain standard control protocols. It also reserves codes for additional standard or vendor-specific control protocols. If the Communication Class control model does not require a specific protocol, the value of 00h should be used.

Table 17: Communication Interface Class Control Protocol Codes

Protocol code	Reference document	Description
00h	USB Specification	No class specific protocol required
01h	V.25ter	Common AT commands (also known as “Hayes™ compatible”)
02h-FEh		RESERVED (future use)
FFh	USB Specification	Vendor-specific

4.5 Data Interface Class Codes

The following table defines the Data Interface Class code:

Table 18: Data Interface Class Code

Code	Class
0Ah	Data Interface Class

4.6 Data Interface Class SubClass Codes

At this time this field is un-used for Data Class interfaces and should have a value of 00h.

4.7 Data Interface Class Protocol Codes

The following table defines the Protocol codes for the Data Interface Class:

Table 19: Data Interface Class Protocol Codes

Protocol Code	Reference Document	Description
00h	USB specification	No class specific protocol required
01h – 2Fh	None	RESERVED (future use)
30h	I.430	Physical interface protocol for ISDN BRI
31h	ISO/IEC 3309-1993	HDLC
32h	None	Transparent
33h – 4Fh	None	RESERVED (future use)
50h	Q.921M	Management protocol for Q.921 data link protocol
51h	Q.921	Data link protocol for Q.931
52h	Q921TM	TEI-multiplexor for Q.921 data link protocol

Protocol Code	Reference Document	Description
53h – 8Fh	None	RESERVED (future use)
90h	V.42bis	Data compression procedures
91h	Q.931/Euro-ISDN	Euro-ISDN protocol control
92h	V.120	V.24 rate adaptation to ISDN
93h	CAPI2.0	CAPI Commands
94h - FCh	None	RESERVED (future use)
FDh	None	Host based driver. Note: This protocol code should only be used in messages between host and device to identify the host driver portion of a protocol stack.
FEh	CDC specification	The protocol(s) are described using a Protocol Unit Functional Descriptors on Communication Class Interface.
FFh	USB specification	Vendor-specific

In certain types of USB communication devices, no protocol will need to be specified in the Data Class interface descriptor. In these cases the value of 00h should be used.

5. Descriptors

5.1 Standard USB Descriptor Definitions

This section defines requirements for the standard USB descriptors for the Communication Device Class, Communication Interface Class and Data Interface Class.

5.1.1 Device Descriptor

Communication device functionality resides at the interface level, with the exception being the definition of the Communication Device Class code. The device code is used solely to identify the device as a communication device and as such, multiple interfaces might be used to form USB functions. This is important to the host for configuration of the drivers to properly enumerate the device. All communication devices will have at least one Communication Class interface that will function as the device master interface. The following tables define the values to properly build a device descriptor and the accompanying interface descriptors.

Table 20: Communication Device Class Descriptor Requirements

Offset	Field	Size	Value	Description
4	*bDeviceClass*	1	02h	Communication Device Class code as defined in Table 14.
5	*bDeviceSubClass*	1	00h	Communication Device Subclass code, unused at this time.
6	*bDeviceProtocol*	1	00h	Communication Device Protocol code, unused at this time.

5.1.2 Configuration Descriptor

The Communication Device Class uses the standard configuration descriptor defined in chapter 9 of the *USB Specification.*

5.1.3 Interface Descriptors

The Communication Interface Class uses the standard Interface descriptor as defined in chapter 9 of the *USB Specification.* The fields defined in the following table shall be used as specified. The use of the remaining fields of the Communication Interface Class descriptor remains unchanged.

Table 21: Communication Class Interface Descriptor Requirements

Offset	Field	Size	Value	Description
5	*bInterfaceClass*	1	Class	Communication Interface Class code, as defined in Table 15.
6	*bInterfaceSubClass*	1	SubClass	Communication Interface Class SubClass code, as defined in Table 16.
7	*bInterfaceProtocol*	1	Protocol	Communication Interface Class Protocol code, which applies to the subclass, as specified in the previous field, is defined in Table 17.

The Data Interface Class also uses the standard Interface descriptor as defined in chapter 9 of the *USB Specification*. The fields defined in the following table shall be used as specified. The use of the remaining fields of the Data Interface Class descriptor remains unchanged.

Table 22: Data Class Interface Descriptor Requirements

Offset	Field	Size	Value	Description
5	*bInterfaceClass*	1	0Ah	Data Interface Class code, as defined in Table 18.
6	*bInterfaceSubClass*	1	00h	Data Class SubClass code.
7	*bInterfaceProtocol*	1	Protocol	Data Class Protocol code, which applies to the subclass, as specified in the previous field, is defined in Table 19.

5.1.4 Endpoint Descriptors

The Communication Interface Class and Data Interface Class use the standard Endpoint descriptor, as defined in chapter 9 of the *USB Specification*.

5.2 Class-Specific Descriptors

This section describes class-specific descriptors for the Communication Interface Class and Data Interface Class. A class-specific descriptor exists only at the Interface level. Each class-specific descriptor is defined as a concatenation of all of the functional descriptors for the Interface. The first functional descriptor returned by the device for the interface shall be a header functional descriptor.

5.2.1 Class-Specific Device Descriptor

This descriptor contains information applying to the entire communication device. The Communication Device Class does not currently use any class-specific descriptor information at the Device level.

5.2.2 Class-Specific Configuration Descriptor

The Communication Device Class currently does not use any class-specific descriptor information at the Configuration level.

5.2.3 Functional Descriptors

Functional descriptors describe the content of the class-specific information within an Interface descriptor. Functional descriptors all start with a common header descriptor, which allows host software to easily parse the contents of class-specific descriptors. Each class-specific descriptor consists of one or more functional descriptors. Although the Communication Class currently defines class specific descriptor information, the Data Class does not.

Table 23: Functional Descriptor General Format

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this descriptor.

Offset	Field	Size	Value	Description
1	*bDescriptorType*	1	Constant	CS_INTERFACE, as defined in Table 24.
2	*bDescriptorSubtype*	1	Constant	Identifier (ID) of functional descriptor. For a list of the supported values, see Table 25.
3	(function specific data0)	1	Misc.	First function specific data byte. These fields will vary depending on the functional descriptor being represented.
…	…	…	…	…
N+2	(functional specific data N-1)	1	Misc.	Nth function specific data byte. These fields will vary depending on the functional descriptor being represented.

The *bDescriptorType* values are the same ones defined in the *USB Device Class Definition for Audio Devices Specification*. They were derived by using the DEVICE, CONFIGURATION, STRING, INTERFACE, and ENDPOINT constants defined in chapter 9 of the *USB Specification* and by setting the class-specific bit defined within the Common Class Specification to generate corresponding class-specific constants.

Table 24: Type Values for the bDescriptorType Field

Descriptor type	Value
CS_INTERFACE	24h
CS_ENDPOINT	25h

Table 25: bDescriptor SubType in Functional Descriptors

Descriptor subtype	Comm IF descriptor	Data IF descriptor	Functional description
00h	Yes	Yes	Header Functional Descriptor, which marks the beginning of the concatenated set of functional descriptors for the interface.
01h	Yes	No	Call Management Functional Descriptor.
02h	Yes	No	Abstract Control Management Functional Descriptor.
03h	Yes	No	Direct Line Management Functional Descriptor.
04h	Yes	No	Telephone Ringer Functional Descriptor.
05h	Yes	No	Telephone Call and Line State Reporting Capabilities Functional Descriptor.
06h	Yes	No	Union Functional descriptor
07h	Yes	No	Country Selection Functional Descriptor
08h	Yes	No	Telephone Operational Modes Functional Descriptor
09h	Yes	No	USB Terminal Functional Descriptor

Descriptor subtype	Comm IF descriptor	Data IF descriptor	Functional description
0Ah	Yes	No	Network Channel Terminal Descriptor
0Bh	Yes	No	Protocol Unit Functional Descriptor
0Ch	Yes	No	Extension Unit Functional Descriptor
0Dh	Yes	No	Multi-Channel Management Functional Descriptor
0Eh	Yes	No	CAPI Control Management Functional Descriptor
0Fh	Yes	No	Ethernet Networking Functional Descriptor
10h	Yes	No	ATM Networking Functional Descriptor
11h-FFh	N/A	N/A	RESERVED (future use)

5.2.3.1 Header Functional Descriptor

The class-specific descriptor shall start with a header that is defined in Table 23. The *bcdCDC* field identifies the release of the *USB Class Definitions for Communication Devices Specification* (this specification) with which this interface and its descriptors comply.

Table 26: Class-Specific Descriptor Header Format

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this descriptor in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE descriptor type.
2	*bDescriptorSubtype*	1	Constant	Header functional descriptor subtype as defined in Table 25.
3	*bcdCDC*	2	Number	USB Class Definitions for Communication Devices Specification release number in binary-coded decimal.

5.2.3.2 Call Management Functional Descriptor

The Call Management functional descriptor describes the processing of calls for the Communication Class interface. It can only occur within the class-specific portion of an Interface descriptor.

Table 27: Call Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Call Management functional descriptor subtype, as defined in Table 25.

Offset	Field	Size	Value	Description
3	*bmCapabilities*	1	Bitmap	The capabilities that this configuration supports: D7..D2: RESERVED (Reset to zero) D1: 0 - Device sends/receives call management information only over the Communication Class interface. 1 - Device can send/receive call management information over a Data Class interface. D0: 0 - Device does not handle call management itself. 1 - Device handles call management itself. The previous bits, in combination, identify which call management scenario is used. If bit D0 is reset to 0, then the value of bit D1 is ignored. In this case, bit D1 is reset to zero for future compatibility.
4	*bDataInterface*	1	Number	Interface number of Data Class interface optionally used for call management. *

* Zero based index of the interface in this configuration.(*bInterfaceNum*)

5.2.3.3 Abstract Control Management Functional Descriptor

The Abstract Control Management functional descriptor describes the commands supported by the Communication Class interface, as defined in Section 3.6.2, with the SubClass code of Abstract Control Model. It can only occur within the class-specific portion of an Interface descriptor.

Table 28: Abstract Control Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Abstract Control Management functional descriptor subtype as defined in Table 25.

Offset	Field	Size	Value	Description
3	*bmCapabilities*	1	Bitmap	The capabilities that this configuration supports. (A bit value of zero means that the request is not supported.) D7..D4: RESERVED (Reset to zero) D3: 1 - Device supports the notification Network_Connection. D2: 1 - Device supports the request Send_Break D1: 1 - Device supports the request combination of Set_Line_Coding, Set_Control_Line_State, Get_Line_Coding, and the notification Serial_State. D0: 1 - Device supports the request combination of Set_Comm_Feature, Clear_Comm_Feature, and Get_Comm_Feature. The previous bits, in combination, identify which requests/notifications are supported by a Communication Class interface with the SubClass code of Abstract Control Model.

5.2.3.4 Direct Line Management Functional Descriptor

The Direct Line Management functional descriptor describes the commands supported by the Communication Class interface, as defined in Section 3.6.1, with the SubClass code of Direct Line Control Model. It can only occur within the class-specific portion of an Interface descriptor.

Table 29: Direct Line Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Direct Line Management functional descriptor subtype, as defined in Table 25.

Offset	Field	Size	Value	Description
3	*bmCapabilities*	1	Bitmap	The capabilities that this configuration supports. (A value of zero means that the request or notification is not supported.) D7..D3: RESERVED (Reset to zero) D2: 1 - Device requires extra Pulse_Setup request during pulse dialing sequence to disengage holding circuit. (see Section 6.2.8) D1: 1 - Device supports the request combination of Set_Aux_Line_State, Ring_Aux_Jack, and notification Aux_Jack_Hook_State. D0: 1 - Device supports the request combination of Pulse_Setup, Send_Pulse, and Set_Pulse_Time. The previous bits, in combination, identify which requests/notifications are supported by a Communication Class interface with the SubClass code of DL Control Modem.

5.2.3.5 Telephone Ringer Functional Descriptor

The Telephone Ringer functional descriptor describes the ringer capabilities supported by the Communication Class interface, as defined in Section 3.6.3.1, with the SubClass code of Telephone Control. It can only occur within the class-specific portion of an Interface descriptor.

For a multiple line phone device, where separate Communication Class interfaces would exist for each line supported by the phone, typically one interface would be designated via a Union functional descriptor to be the controlling interface for the device. If only one ringer existed for all the lines, the Telephone Ringer Functional descriptor would only be needed for the descriptor of this controlling interface.

Table 30: Telephone Ringer Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Telephone Ringer functional descriptor subtype as defined in Table 25.

Offset	Field	Size	Value	Description
3	*bRingerVolSteps*	1	Number	Number of discrete steps in volume supported by the ringer, values are: 0: 256 discrete volume steps. 1: 1 discrete volume step (i.e., fixed volume). Value 0 will be ringer off setting and Value 1 to 255 will result in the same ringer volume level. 2: 2 discrete volume steps Value 0 will be ringer off setting and Values 1 to 127 will result in the first volume level setting. Values 128 to 255 will result in the 2nd volume level setting 3: 3 discrete volume steps. Value 0 will be ringer off setting and Values 1 to 84 will result in the first volume level setting. Value 85 to 170 will result in the 2nd volume level setting. Value 171 to 255 will result in the 3rd volume level setting. As a general rule, the range of volume settings is broken up into a number of equal steps, the number of steps defined by the bRingerVolSteps value. A general formula for defining ranges, based on X=bRingerVolSteps and values [1 to Y] defining the first volume range is: Y = (256/X) - 1, where X<>0 and X<=128 Second volume range is: [(Y+1) to (Y+Y)] Note: that the maximum value in the last range must always be 255
4	*bNumRingerPatterns*	1	Number	Number of ringer patterns supported, values of 1 to 255 with a value of 0 being reserved for future use.

5.2.3.6 Telephone Operational Modes Functional Descriptor

The Telephone Operational Modes functional descriptor describes the operational modes supported by the Communication Class interface, as defined in Section 3.6.3.1, with the SubClass code of Telephone Control. It can only occur within the class-specific portion of an Interface descriptor. The modes supported are Simple, Standalone, and Computer Centric. See Section 6.2.18, "*SetOperationParms*" for a definition of the various operational modes and Table 53 for the definition of the operational mode values.

For a multiple line phone device, where separate Communication Class interfaces would exist for each line supported by the phone, typically one interface would be designated via a Union functional descriptor to be the controlling interface for the device. In this case, the Telephone Operational Modes descriptor would only be needed for the descriptor of this controlling interface.

Table 31: Telephone Operational Modes Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Telephone Operational Modes functional descriptor subtype as defined in Table 25.
3	*bmCapabilities*	1	Bitmap	This configuration supports the following operational modes: D7..D3: RESERVED (Reset to zero) D2: 0 - Does not support Computer Centric mode. 1 - Supports Computer Centric mode. D1: 0 - Does not support Standalone mode. 1 - Supports Standalone mode. D0: 0 - Does not support Simple mode. 1 - Supports Simple mode.

5.2.3.7 Telephone Call and Line State Reporting Capabilities Descriptor

The Telephone Call and Line State Reporting Capabilities functional descriptor describes the abilities of a telephone device to report optional call and line states. All telephone devices, as a minimum, shall be capable of reporting the following call states:

- Idle
- Dialtone
- Dialing
- Connected
- Ringing
- Answered

Call state reports that are optional and will be described by this descriptor are states such as:

- Interrupted dialtone
- Ringback
- Busy
- Fast busy (also known as equipment busy or reorder tone)
- Caller ID
- Distinctive ringing decoding

Line state reports are optional and will be described by this descriptor.

The Telephone Call State Reporting Capabilities functional descriptor can exist in the class-specific portion of a Communication Class interface, as defined in Section 3.6.3.1, with the SubClass code of Telephone Control. For a multiple line phone device, where separate Communication Class interfaces would exist for the each line supported by the phone, typically one interface would be designated via a Union functional descriptor, to be the controlling interface

for the device. In this case, the Telephone Call State Reporting Capabilities Functional descriptor would only be needed for the descriptor of this controlling interface, if each of the Communication Class interfaces supported the same call state reporting capabilities.

Table 32: Telephone Call State Reporting Capabilities Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Telephone Call State Reporting Capabilities descriptor subtype, as defined in Table 25.
3	*bmCapabilities*	4	Bitmap	Call and line state reporting capabilities of the device when the following bits are set: D31-D6: RESERVED (Reset to zero) D5: 0 – Does not support line state change notification. 1 – Does support line state change notification. D4: 0 – Cannot report dual tone multi-frequency (DTMF) digits input remotely over the telephone line. 1 – Can report DTMF digits input remotely over the telephone line. D3: 0 – Reports only incoming ringing. 1 – Reports incoming distinctive ringing patterns. D2: 0 – Does not report caller ID. 1 – Reports caller ID information. D1: 0 – Reports only dialing state. 1 – Reports ringback, busy, and fast busy states. D0: 0 – Reports only dialtone (does not differentiate between normal and interrupted dialtone). 1 – Reports interrupted dialtone in addition to normal dialtone.

5.2.3.8 Union Functional Descriptor

The Union functional descriptor describes the relationship between a group of interfaces that can be considered to form a functional unit. It can only occur within the class-specific portion of an Interface descriptor. One of the interfaces in the group is designated as a *master* or *controlling* interface for the group, and certain class-specific messages can be sent to this interface to act upon the group as a whole. Similarly, notifications for the entire group can be sent from this interface but apply to the entire group of interfaces. Interfaces in this group can include Communication, Data, or any other valid USB interface class (including, but not limited to, Audio, HID, and Monitor).

Table 33: Union Interface Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional 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 Table 25.
3	*bMasterInterface*	1	Constant	The interface number of the Communication or Data Class interface, designated as the master or controlling interface for the union.*
4	*bSlaveInterface0*	1	Number	Interface number of first slave or associated interface in the union. *
…	…	…	…	
N+3	*bSlaveInterfaceN-1*	1	Number	Interface number of N-1 slave or associated interface in the union. *

* Zero based index of the interface in this configuration (*bInterfaceNum*).

5.2.3.9 Country Selection Functional Descriptor

The Country Selection functional descriptor identifies the countries in which the communication device is qualified to operate. The parameters of the network connection often vary from one country to another, especially in Europe. Also legal requirements impose certain restrictions on devices because of different regulations by the governing body of the network to which the device must adhere. This descriptor can only occur within the class-specific portion of an Interface descriptor and should only be provided to a master Communication Class interface of a union. The country codes used in the Country Selection Functional Descriptor are not the same as the country codes used in dialing international telephone calls. Implementers should refer to the ISO 3166 specification for more information.

Table 34: Country Selection Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Country Selection functional descriptor Subtype as defined in Table 25.
3	*iCountryCodeRelDate*	1	Index	Index of a string giving the release date for the implemented ISO 3166 Country Codes. Date shall be presented as *ddmmyyyy* with *dd*=day, *mm*=month, and *yyyy*=year.
4	*wCountryCode0*	2	Number	Country code in hexadecimal format as defined in ISO 3166, release date as specified in offset 3 for the first supported country.
…	…	…	…	
2N+2	*wCountryCodeN-1*	2	Number	Country code in hexadecimal format as defined in ISO 3166, release date as specified in offset 3 for Nth country supported.

5.2.3.10 USB Terminal Functional Descriptor

The *USB Terminal Functional Descriptor* provides a means to indicate a relationship between a Unit and an USB Interface. It also defines parameters specific to the interface between the device and the host. It can only occur within the class-specific portion of an Interface descriptor.

Table 35: USB Terminal Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	USB Terminal Functional Descriptor Subtype as defined in Table 25.
3	*bEntityId*	1	Constant	Constant uniquely identifying the Terminal
4	*bInInterfaceNo*	1	Number	The input interface number of the associated USB interface.
5	*bOutInterfaceNo*	1	Number	The output interface number of the associated USB interface.
6	*bmOptions*	1	Bitmap	D7..D1: RESERVED (Reset to zero) D0: Protocol wrapper usage 0 - No wrapper used 1 - Wrapper used
7	*bChildId0*	1	Constant	First ID of lower Terminal or Unit to which this Terminal is connected.
…	…	…	…	
6+N	*bChildIdN-1*	1	Constant	Nth ID of lower Terminal or Unit to which this Terminal is connected.

5.2.3.11 Network Channel Terminal Functional Descriptor

The Network Channel Terminal Functional descriptor provides a means to indicate a relationship between a Unit and a Network Channel. It can only occur within the class-specific portion of an Interface descriptor.

Table 36: Network Channel Terminal Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Network Channel Terminal Functional Descriptor Subtype as defined in Table 25.
3	*bEntityId*	1	Constant	Constant uniquely identifying the Terminal
4	*iName*	1	Index	Index of string descriptor, describing the name of the Network Channel Terminal.
5	*bChannelIndex*	1	Number	The channel index of the associated network channel according to indexing rules below.
6	*bPhysicalInterface*	1	Constant	Type of physical interface: 0 – None 1 – ISDN 2 to 200 – RESERVED (future use) 201 to 255 - Vendor specific

Channel Indexing Rule

A zero-based value identifying the index in the array of concurrent channels multiplexed on the physical interface. For an ISDN physical interface the *bChannelIndex* starts with zero for the D-channel, one for B1 and so forth.

5.2.3.12 Protocol Unit Functional Descriptor

A communication protocol stack is a combination of communication functions (protocols) into a layered structure. Each layer in the stack presents some abstract function for the layer above according to some layer-interface-standard, making it possible to replace a function with another as long as it conforms to the standard. Each layer may have a set of protocol parameters, defined in Appendix E, to configure it for proper operation in the actual environment and the parameters may be retrieved and/or modified. The Unit state is initially reset. See Section 6.2.23 "***SetUnitParameter***", Section 6.2.24 "***GetUnitParameter***", and Section 6.2.25 "***ClearUnitParameter***" for details.

A *Protocol Unit Functional Descriptor* identifies with *bEntityId* a specific protocol instance of *bProtocol* in a stack. It can only occur within the class-specific portion of an Interface descriptor.

Table 37: Protocol Unit Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Protocol Unit Functional Descriptor Subtype as defined in Table 25.
3	*bEntityId*	1	Constant	Constant uniquely identifying the Unit
4	*bProtocol*	1	Protocol	Protocol code as defined in Table 19
5	*bChildId0*	1	Constant	First ID of lower Terminal or Unit to which this Terminal is connected.
…	…	…	…	
4+N	*bChildIdN-1*	1	Constant	Nth ID of lower Terminal or Unit to which this Terminal is connected.

5.2.3.13 Extension Unit Functional Descriptor

The *Extension Unit Functional Descriptor* provides minimal information about the Extension Unit for a generic driver at least to notice the presence of vendor-specific components within the protocol stack. The *bExtensionCode* field may contain a vendor-specific code that further identifies the Extension Unit. If it is not used, it should be set to zero. The Unit may have a set of vendor specific parameters to configure it for proper operation in the actual environment and the parameters may be retrieved and/or modified. The Unit state is initially reset. Set Section 6.2.23 "SetUnitParameter", Section 6.2.24 "*GetUnitParameter*", and Section 6.2.25 "*ClearUnitParameter*" for details.

The descriptor can only occur within the class-specific portion of an Interface descriptor.

Table 38: Extension Unit Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Extension Unit Functional Descriptor Subtype as defined in Table 25

Offset	Field	Size	Value	Description
3	*bEntityId*	1	Constant	Constant uniquely identifying the Unit
4	*bExtensionCode*	1	Number	Vendor specific code identifying the Extension Unit.
5	*iName*	1	Index	Index of string descriptor, describing the name of the Extension Unit.
6	*bChildId0*	1	Constant	First ID of lower Terminal or Unit to which this Terminal is connected.
…	…	…	…	
5+N	*bChildIdN-1*	1	Constant	Nth ID of lower Terminal or Unit to which this Terminal is connected.

5.2.3.14 Multi-Channel Management Functional Descriptor

The Multi-Channel Management functional descriptor describes the commands supported by the Communication Class interface, as defined in Section 3.6.1, with the SubClass code of Multi-Channel. It can only occur within the class-specific portion of an Interface descriptor.

Table 39: Multi-Channel Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Multi-Channel Management functional descriptor subtype, as defined in Table 25
3	*bmCapabilities*	1	Bitmap	The capabilities that this configuration supports. (A value of zero means that the request or notification is not supported.) D7..D3: RESERVED (Reset to zero) D2: 1 – Device supports the request Set_Unit_Parameter. D1: 1 – Device supports the request Clear_Unit_Parameter. D0: 1 – Device stores Unit parameters in non-volatile memory. The previous bits identify which requests are supported by a Communication Class interface with the SubClass code of Multi-Channel Control Model.

5.2.3.15 CAPI Control Management Functional Descriptor

The CAPI control management functional descriptor describes the commands supported by the CAPI Control Model over the Data Class interface with the protocol code of CAPI control. It can only occur within the class specific portion of Communication Class Interface descriptor.

Table 40: CAPI Control Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	CAPI Control Management Functional Descriptor Subtype as defined in Table 25
3	*bmCapabilities*	1	Bitmap	Which capabilities are supported by this configuration. D7..D1: RESERVED (reset to zero). D0: 1 – device is an Intelligent CAPI device 0 – device is a Simple CAPI device The above bits, in combination, identify which requests/notifications are supported by a Communication Class interface with the protocol code of CAPI Control.

5.2.3.16 Ethernet Networking Functional Descriptor

The Ethernet Networking functional descriptor describes the operational modes supported by the Communication Class interface, as defined in Section 3.8.2, with the SubClass code of Ethernet Networking Control. It can only occur within the class-specific portion of an Interface descriptor.

Table 41: Ethernet Networking Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Ethernet Networking functional descriptor subtype as defined in Table 25.
3	*iMACAddress*	1	Index	Index of string descriptor. The string descriptor holds the 48bit Ethernet MAC address. The Unicode representation of the MAC address is as follows: the first Unicode character represents the high order nibble of the first byte of the MAC address in network byte order. The next character represents the next 4 bits and so on. The Unicode character is chosen from the set of values 30h through 39h and 41h through 46h (0-9 and A-F). iMACAddress can not be zero and the Unicode representation must be 12 characters long. For example, the MAC Address 0123456789ABh is represented as the Unicode string "0123456789AB".

Offset	Field	Size	Value	Description
4	*bmEthernetStatistics*	4	Bitmap	Indicates which Ethernet statistics functions the device collects. If a bit is set to 0, the host network driver is expected to keep count for the corresponding statistic (if able). See Table 42 for a detailed listing of possible Ethernet statistics. Support for any of these statistics is optional. If none of these bits are set, the device does not support the GetEthernetStatistic request.
8	*wMaxSegmentSize*	2	Number	The maximum segment size that the Ethernet device is capable of supporting. This is typically 1514 bytes, but could be extended (e.g., 802.1d VLAN)
10	*wNumberMCFilters*	2	Bitmap	Contains the number of multicast filters that can be configured by the host. D15: 0 - The device performs perfect multicast address filtering (no hashing). 1- The device uses imperfect multicast address filtering (hashing). Here, the host software driver must perform further qualification itself to achieve perfect filtering. D14..0: Indicates the number of multicast address filters supported by the device (0 to 32767). If the host finds the number of filters supported by the device to be inadequate, it may choose to set the device's Ethernet Packet Filter to forward all multicast frames to the host, performing all multicast filtering in software instead. If this value is 0, the device does not support the SetEthernetMulticastFilters request.
12	*bNumberPowerFilters*	1	Number	Contains the number of pattern filters that are available for causing wake-up of the host.

Table 42: Ethernet Statistics Capabilities

Offset	Field	Description
D0	XMIT_OK	Frames transmitted without errors
D1	RVC_OK	Frames received without errors
D2	XMIT_ERROR	Frames not transmitted, or transmitted with errors
D3	RCV_ERROR	Frames received with errors that are not delivered to the USB host.
D4	RCV_NO_BUFFER	Frame missed, no buffers
D5	DIRECTED_BYTES_XMIT	Directed bytes transmitted without errors
D6	DIRECTED_FRAMES_XMIT	Directed frames transmitted without errors
D7	MULTICAST_BYTES_XMIT	Multicast bytes transmitted without errors
D8	MULTICAST_FRAMES_XMIT	Multicast frames transmitted without errors

Offset	Field	Description
D9	BROADCAST_BYTES_XMIT	Broadcast bytes transmitted without errors
D10	BROADCAST_FRAMES_XMIT	Broadcast frames transmitted without errors
D11	DIRECTED_BYTES_RCV	Directed bytes received without errors
D12	DIRECTED_FRAMES_RCV	Directed frames received without errors
D13	MULTICAST_BYTES_RCV	Multicast bytes received without errors
D14	MULTICAST_FRAMES_RCV	Multicast frames received without errors
D15	BROADCAST_BYTES_RCV	Broadcast bytes received without errors
D16	BROADCAST_FRAMES_RCV	Broadcast frames received without errors
D17	RCV_CRC_ERROR	Frames received with circular redundancy check (CRC) or frame check sequence (FCS) error
D18	TRANSMIT_QUEUE_LENGTH	Length of transmit queue
D19	RCV_ERROR_ALIGNMENT	Frames received with alignment error
D20	XMIT_ONE_COLLISION	Frames transmitted with one collision
D21	XMIT_MORE_COLLISIONS	Frames transmitted with more than one collision
D22	XMIT_DEFERRED	Frames transmitted after deferral
D23	XMIT_MAX_COLLISIONS	Frames not transmitted due to collisions
D24	RCV_OVERRUN	Frames not received due to overrun
D25	XMIT_UNDERRUN	Frames not transmitted due to underrun
D26	XMIT_HEARTBEAT_FAILURE	Frames transmitted with heartbeat failure
D27	XMIT_TIMES_CRS_LOST	Times carrier sense signal lost during transmission
D28	XMIT_LATE_COLLISIONS	Late collisions detected
D29-D31	RESERVED	Must be set to zero

5.2.3.17 ATM Networking Functional Descriptor

The ATM Networking functional descriptor describes the operational modes supported by the Communication Class interface, as defined in Section 3.8.3, with the SubClass code of ATM Networking Control. It can only occur within the class-specific portion of an Interface descriptor.

Table 43: ATM Networking Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	ATM Networking functional descriptor subtype as defined in Table 25.

Offset	Field	Size	Value	Description
3	*iEndSystemIdentifier*	1	Index	Index of string descriptor. The string descriptor holds the End System Identifier. The first 6 bytes are the unique hardware ID (like a MAC address), and the 7th byte is the end system selector byte. The Unicode representation of End System Identifier is as follows: the first Unicode character represents the high order nibble of the first byte of the Identifier in network byte order. The next character represents the next 4 bits, and so on. The Unicode character is chosen from the set of values 30h through 39h and 41h through 46h (0-9 and A-F). iEndSystemIdentifier can not be zero and the Unicode representation must be 14 characters long. For example, the End System Identifier 0123456789ABCDh is represented as the Unicode string "0123456789ABCD".
4	*bmDataCapabilities*	1	Bitmap	The ATM data types the device supports: D7..D4: RESERVED (Reset to zero) D3: Type 3 -- AAL5 SDU D2: Type 2 -- ATM header template + concatenated ATM cell payloads D1: Type 1 -- Concatenated ATM cells D0: RESERVED (Reset to zero) NOTE: Support for the Type 1 Data Format is mandatory.
5	*bmATMDeviceStatistics*	1	Bitmap	Indicates which optional statistics functions the device collects. If set to 0, the host network driver is expected to keep count. D3 and D4 are only applicable to type 3 devices. If neither D3 nor D4 are set, the type 3 device does not support the SetATMDefaultVC and GetATMVCStatistics requests. D7..D5: RESERVED (Reset to zero) D4: Device counts upstream cells sent on a per VC basis (VC_US_CELLS_ SENT) D3: Device counts downstream cells received on a per VC basis (VC_DS_CELLS_ RECEIVED) D2: Device counts cells with HEC error detected and corrected (DS_CELLS_HEC_ERROR_ CORRECTED) D1: Device counts upstream cells sent (US_CELLS_SENT) D0: Device counts downstream cells received (DS_CELLS_RECEIVED)
6	*wType2MaxSegmentSize*	2	Number	The maximum segment size that the Type 2 device is capable of supporting
8	*wType3MaxSegmentSize*	2	Number	The maximum segment size that the Type 3 device is capable of supporting

Offset	Field	Size	Value	Description
10	*wMaxVC*	2	Number	The maximum number of simultaneous virtual circuits the device is capable of supporting (Type 3 only)

The *wType2MaxSegmentSize* and *wType3MaxSegmentSize* indicate the maximum number of bytes of data in a network segment (i.e., between 2 USB short packets) a device would send to the host via USB for Type 2 and Type 3 devices, respectively. It is expected that a device supporting both Type 2 and Type 3 ATM data formats (as stipulated in the *bmDataCapabilities* field) may employ different buffer management strategies for different ATM data formats, thus may indicate a different maximum segment size for each type. The host driver should allocate buffers accordingly that are large enough to hold the incoming data to prevent any overflow or partial cell/SDU problems. These two fields also serve to inform the host the device buffer capability, and expect the host to transfer network segments no longer than that specified here. Note that this information does not preclude a device or a host to do cut-through forwarding (i.e., start forwarding the portion of the network segment received so far) before receiving a complete network segment.

This maximum segment size does not apply to Type 1 devices, which by definition forward a stream of cells in both directions, and have no concept of network segment size. However, as an implementation note, the host driver should allocate its buffers to contain an integral number of 53-byte cells to prevent partial cells.

5.3 Sample Class-Specific Functional Descriptors

Table 44 presents an example of the Communication Class Functional Descriptors for a simple Abstract Control Model device.

Table 44: Sample Communication Class Specific Interface Descriptor*

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	05h	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	24h	CS_INTERFACE
2	*bDescriptorSubtype*	1	00h	Header. This is defined in Table 25, which defines this as a header.
3	*bcdCDC*	2	0110h	USB Class Definitions for Communication Devices Specification release number in binary-coded decimal.
5	*bFunctionLength*	1	04h	Size of this functional descriptor, in bytes.
6	*bDescriptorType*	1	24h	CS_INTERFACE
7	*bDescriptorSubtype*	1	02h	Abstract Control Management functional descriptor subtype as defined in Table 25.
8	*bmCapabilities*	1	0Fh	This field contains the value 0Fh, because the device supports all the corresponding commands for the Abstract Control Model interface.
9	*bFunctionLength*	1	05h	Size of this functional descriptor, in bytes
10	*bDescriptorType*	1	24h	CS_INTERFACE
11	*bDescriptorSubtype*	1	06h	Union Descriptor Functional Descriptor subtype as defined in Table 25.
12	*bMasterInterface*	1	00h	Interface number of the control (Communication Class) interface
13	*bSlaveInterface0*	1	01h	Interface number of the slave (Data Class) interface

Offset	Field	Size	Value	Description
14	*bFunctionLength*	1	05h	Size of this functional descriptor, in bytes
15	*bDescriptorType*	1	24h	CS_INTERFACE
16	*bDescriptorSubtype*	1	01h	Call Management Functional Descriptor subtype as defined in Table 25.
17	*bmCapabilities*	1	03h	Indicate that the device handles call management itself (bit D0 is set), and will process commands multiplexed over the data interface in addition to commands sent using SEND_ENCAPSULATED_COMMAND (bit D1 is set).
18	*bDataInterface*	1	01h	Indicates that multiplexed commands are handled via data interface 01h (same value as used in the UNION Functional Descriptor)

* This descriptor is specific to the Communication Class.

6. Communication Interface Class Messages

6.1 Overview

The Communication Interface Class supports the standard requests defined in chapter 9 of the *USB Specification*. In addition, the Communication Interface Class has some class-specific requests and notifications. These are used for device and call management.

6.2 Management Element Requests

The Communication Interface Class supports the following class-specific requests. This section describes the requests that are specific to the Communication Interface Class. These requests are sent over the management element and can apply to different device views as defined by the Communication Class interface codes.

Table 45: Class-Specific Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_ENCAPSULATED_COMMAND	Zero	Interface	Amount of data, in bytes, associated with this recipient.	Control protocol-based command
10100001B	GET_ENCAPSULATED_RESPONSE	Zero	Interface	Amount of data, in bytes, associated with this recipient.	Protocol-dependent data response
00100001B	SET_COMM_FEATURE	Feature Selector	Interface	Length of State Data	State
10100001B	GET_COMM_FEATURE	Feature Selector	Interface	Length of Status Data	Status
00100001B	CLEAR_COMM_FEATURE	Feature Selector	Interface	Zero	None
00100001B	SET_AUX_LINE_STATE	0 –Disconnect 1 – Connect	Interface	Zero	None
00100001B	SET_HOOK_STATE	Relay Config.	Interface	Zero	None
00100001B	PULSE_SETUP	Enable/ Disable	Interface	Zero	None
00100001B	SEND_PULSE	Cycles	Interface	Zero	None
00100001B	SET_PULSE_TIME	Timing	Interface	Zero	None
00100001B	RING_AUX_JACK	Number of Rings	Interface	Zero	None
00100001B	SET_LINE_CODING	Zero	Interface	Size of properties	Line Coding Structure
10100001B	GET_LINE_CODING	Zero	Interface	Size of Structure	Line Coding Structure
00100001B	SET_CONTROL_LINE_STATE	Control Signal Bitmap	Interface	Zero	None
00100001B	SEND_BREAK	Duration of Break	Interface	Zero	None

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_RINGER_PARMS	Zero	Interface	4	Ringer Configuration bitmap
10100001B	GET_RINGER_PARMS	Zero	Interface	4	Ringer Configuration bitmap
00100001B	SET_OPERATION_ PARMS	Operation Mode	Interface	Zero	None
10100001B	GET_OPERATION_ PARMS	Zero	Interface	2	Operation mode
00100001B	SET_LINE_PARMS	Lines State Change	Interface	Length of Data	None/Data
10100001B	GET_LINE_PARMS	Zero	Interface	Size of Structure	Line Status Information structure
00100001B	DIAL_DIGITS	Zero	Interface	Length of Dial String	Dialing string
00100001B	SET_UNIT_ PARAMETER	Unit Parameter Structure	Interface	Length of Unit Parameter	Unit Parameter
10100001B	GET_UNIT_ PARAMETER	Unit Parameter Structure	Interface	Length of Unit Parameter	Unit Parameter
00100001B	CLEAR_UNIT_ PARAMETER	Unit Parameter Structure	Interface	Zero	None
10100001B	GET_PROFILE	Zero	Interface	64	Profile Information
00100001B	SET_ETHERNET_ MULTICAST_FILTERS	Number of filters (N)	Interface	N * 6	N 48 bit Multicast addresses
00100001B	SET_ETHERNET_ POWER_ MANAGEMENT_ PATTERN_FILTER	Filter number	Interface	Size of structure	Power management pattern filter structure
10100001B	GET_ETHERNET_ POWER_ MANAGEMENT_ PATTERN_FILTER	Filter number	Interface	2	Pattern active boolean
00100001B	SET_ETHERNET_ PACKET_FILTER	Packet Filter Bitmap	Interface	Zero	None
10100001B	GET_ETHERNET_ STATISTIC	Feature Selector	Interface	4	32 bit unsigned integer
00100001B	SET_ATM_DATA_ FORMAT	Data Format	Interface	Zero	None
10100001B	GET_ATM_DEVICE_ STATISTICS	Feature Selector	Interface	4	32 bit unsigned integer
00100001B	SET_ATM_DEFAULT_ VC	Zero	Interface	3	1-byte VPI followed by 2-byte VCI value
10100001B	GET_ATM_VC_ STATISTICS	Feature Selector	Interface	4	32 bit unsigned integer

Table 46: Class-Specific Request Codes

Request	Value
SEND_ENCAPSULATED_COMMAND	00h
GET_ENCAPSULATED_RESPONSE	01h
SET_COMM_FEATURE	02h
GET_COMM_FEATURE	03h
CLEAR_COMM_FEATURE	04h
RESERVED (future use)	05h-0Fh
SET_AUX_LINE_STATE	10h
SET_HOOK_STATE	11h
PULSE_SETUP	12h
SEND_PULSE	13h
SET_PULSE_TIME	14h
RING_AUX_JACK	15h
RESERVED (future use)	16h-1Fh
SET_LINE_CODING	20h
GET_LINE_CODING	21h
SET_CONTROL_LINE_STATE	22h
SEND_BREAK	23h
RESERVED (future use)	24h-2Fh
SET_RINGER_PARMS	30h
GET_RINGER_PARMS	31h
SET_OPERATION_PARMS	32h
GET_OPERATION_PARMS	33h
SET_LINE_PARMS	34h
GET_LINE_PARMS	35h
DIAL_DIGITS	36h
SET_UNIT_PARAMETER	37h
GET_UNIT_PARAMETER	38h
CLEAR_UNIT_PARAMETER	39h
GET_PROFILE	3Ah
RESERVED (future use)	3Bh-3Fh
SET_ETHERNET_MULTICAST_FILTERS	40h
SET_ETHERNET_POWER_MANAGEMENT_PATTERN FILTER	41h
GET_ETHERNET_POWER_MANAGEMENT_PATTERN FILTER	42h
SET_ETHERNET_PACKET_FILTER	43h
GET_ETHERNET_STATISTIC	44h
RESERVED (future use)	45h-4Fh

Request	Value
SET_ATM_DATA_FORMAT	50h
GET_ATM_DEVICE_STATISTICS	51h
SET_ATM_DEFAULT_VC	52h
GET_ATM_VC_STATISTICS	53h
RESERVED (future use)	54h-FFh

6.2.1 SendEncapsulatedCommand

This request is used to issue a command in the format of the supported control protocol of the Communication Class interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_ENCAPSULATED_COMMAND	Zero	Interface	Amount of data, in bytes, associated with this recipient.	Control protocol-based command

6.2.2 GetEncapsulatedResponse

This request is used to request a response in the format of the supported control protocol of the Communication Class interface.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ ENCAPSULATED_ RESPONSE	Zero	Interface	Amount of data, in bytes, associated with this recipient.	Protocol dependent data

6.2.3 SetCommFeature

This request controls the settings for a particular communication feature of a particular target

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_COMM_FEATURE	Feature Selector	Interface	Length of State Data	State

For more information about the defined list of feature selectors per target, see Section 6.2.4, "***GetCommFeature***."

6.2.4 GetCommFeature

This request returns the current settings for the communication feature as selected

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_COMM_FEATURE	Feature Selector	Interface	Length of Status Data	Status

Table 47: Communication Feature Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
ABSTRACT_STATE	01h	Interface	2	Two bytes of data describing multiplexed state and idle state for this Abstract Model communications device. This selector is only valid for Abstract Control Model.
COUNTRY_SETTING	02h	Interface	2	Country code in hexadecimal format as defined in ISO 3166, release date as specified in offset 3 of the Country Selection Functional Descriptor. This selector is only valid for devices that provide a Country Selection Functional Descriptor, and the value supplied shall appear as supported country in the Country Selection Functional Descriptor

For the ABSTRACT_STATE selector, the following two bytes of data are defined:

Table 48: Feature Status Returned for ABSTRACT_STATE Selector

Bit position	Description
D15..D2	RESERVED (Reset to zero)
D1	Data Multiplexed State 1: Enables the multiplexing of call management commands on a Data Class. 0: Disables multiplexing.
D0	Idle Setting 1: All of the endpoints in this interface will not accept data from the host or offer data to the host. This allows the host call management software to synchronize the call management element with other media stream interfaces and endpoints, particularly those associated with a different host entity (such as a voice stream configured as a USB Audio Class device). 0: The endpoints in this interface will continue to accept/offer data.

6.2.5 ClearCommFeature

This request controls the settings for a particular communication feature of a particular target, setting the selected feature to its default state. The validity of the feature selectors depends upon the target type of the request.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	CLEAR_ COMM_FEATURE	Feature Selector	Interface	Zero	None

For more information about for the defined list of feature selectors per target, see Section 6.2.4, "***GetCommFeature***."

6.2.6 SetAuxLineState

This request is used to connect or disconnect a secondary jack to POTS circuit or CODEC, depending on hook state.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_AUX_LINE_STATE	0 - Disconnect 1 - Connect	Interface	Zero	None

State selector values in the *wValue* field are used to instruct the device to connect or disconnect the secondary phone jack from the POTS circuit or CODEC, depending on hook state. Device will acknowledge the status change.

6.2.7 SetHookState

This request is used to set the necessary POTS line relay code for on-hook, off-hook, and caller ID states.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_HOOK_STATE	Relay Config.	Interface	Zero	None

The *wValue* will instruct the device to configure the necessary relays for going off-hook, on-hook, or into a snooping state for receiving caller ID data.

Table 49: POTS Relay Configuration Values

Code	Value
ON_HOOK	0000h
OFF_HOOK	0001h
SNOOPING	0002h

6.2.8 PulseSetup

This request is used to prepare for a pulse-dialing cycle.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	PULSE_SETUP	Enable/ Disable	Interface	Zero	None

If *wValue* field contains the value FFFFh, the request is being sent to disengage the holding circuit after the dialing sequence has been completed. Any other value in the *wValue* field is meant to prepare the device for a pulse-dialing cycle.

Not all devices require a **PulseSetup** request to disengage the holding circuit after a pulse dialing cycle. The extra request in the dialing cycle is generally required for devices designed to be usable in multiple countries. The device indicates whether the extra request is required or not by setting bit D2 of Direct Line Management Functional Descriptor, in Section 5.2.3.4.

6.2.9 SendPulse

This request is used to generate a specified number of make/break pulse cycles.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_PULSE	Cycles	Interface	Zero	None

The *wValue* field contains the number of make/break pulse cycles to generate.

6.2.10 SetPulseTime

This request sets the timing of the make and break periods for pulse dialing.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_PULSE_TIME	Timing	Interface	Zero	None

The *wValue* field specifies the break time period in the high byte and the make time period in the low byte. The time periods are specified in milliseconds.

6.2.11 RingAuxJack

This request is used to generate a ring signal on a secondary phone jack.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	RING_AUX_JACK	Number of Rings	Interface	Zero	None

The *wValue* field contains the number of ring signals to generate on a secondary phone jack of the device.

6.2.12 SetLineCoding

This request allows the host to specify typical asynchronous line-character formatting properties, which may be required by some applications. This request applies to asynchronous byte stream data class interfaces and endpoints; it also applies to data transfers both from the host to the device and from the device to the host.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_LINE_CODING	Zero	Interface	Size of Structure	Line Coding Structure

For the definition of valid properties, see Table 50, Section 6.2.13, "***GetLineCoding***."

6.2.13 GetLineCoding

This request allows the host to find out the currently configured line coding.

bmRequestType	bRequest	wValue	wIndex	Wlength	Data
10100001B	GET_LINE_CODING	Zero	Interface	Size of Structure	Line Coding Structure

The line coding properties are defined in the following table:

Table 50: Line Coding Structure

Offset	Field	Size	Value	Description
0	*dwDTERate*	4	Number	Data terminal rate, in bits per second.
4	*bCharFormat*	1	Number	Stop bits 0 - 1 Stop bit 1 - 1.5 Stop bits 2 - 2 Stop bits
5	*bParityType*	1	Number	Parity 0 - None 1 - Odd 2 - Even 3 - Mark 4 - Space
6	*bDataBits*	1	Number	Data bits (5, 6, 7, 8 or 16).

6.2.14 SetControlLineState

This request generates RS-232/V.24 style control signals.

bmRequestType	bRequest	wValue	wIndex	WLength	Data
00100001B	SET_CONTROL_LINE _STATE	Control Signal Bitmap	Interface	Zero	None

Table 51: Control Signal Bitmap Values for SetControlLineState

Bit position	Description
D15..D2	RESERVED (Reset to zero)
D1	Carrier control for half duplex modems. This signal corresponds to V.24 signal 105 and RS-232 signal RTS. 0 - Deactivate carrier 1 - Activate carrier The device ignores the value of this bit when operating in full duplex mode.

Bit position	Description
D0	Indicates to DCE if DTE is present or not. This signal corresponds to V.24 signal 108/2 and RS-232 signal DTR. 0 - Not Present 1 - Present

6.2.15 SendBreak

This request sends special carrier modulation that generates an RS-232 style break.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SEND_BREAK	Duration of Break	Interface	Zero	None

The *wValue* field contains the length of time, in milliseconds, of the break signal. If *wValue* contains a value of FFFFh, then the device will send a break until another **SendBreak** request is received with the *wValue* of 0000h.

6.2.16 SetRingerParms

This request configures the ringer for the communication device, either on a global basis (master interface of the union), or on a per-line basis for multiple line devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_RINGER_PARMS	Zero	Interface	4	Ringer Configuration bitmap

The command sets up the ringer characteristics for the communication device or for the line. The Ringer Configuration bitmap is defined in the following table:

Table 52: Ringer Configuration Bitmap Values

Bit position	Description
D31	0=A ringer does not exist. 1=A ringer exists. When using the GetRingerParms request to return the Ringer Configuration bitmap, a value of zero for this bit means a ringer does not exist for the addressed element (i.e. device or line).
D30..D16	RESERVED (Reset to zero)
D15..D8	Ringer Volume Setting 0 - Ringer Volume Off 255 - Maximum Ringer Volume
D7..D0	Ringer Pattern Type Selection This corresponds to an internal ringer pattern or sound supported within the device, which could be a distinctive ringing type pattern or a sound effect type ring like a chirping sound, siren sound, etc.

6.2.17 GetRingerParms

This request returns the ringer capabilities of the device and the current status of the device's ringer, including its enabled state and current selection.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_RINGER_PARMS	Zero	Interface	4	Ringer Configuration bitmap

This command is typically sent to the master interface of the union. If the ringer for each line can be configured independently, then sending the command to the interface representing a line gets the ringer information for that line. For a description of the returned Ringer Configuration bitmap values, see Table 52.

6.2.18 SetOperationParms

Sets the operational mode for the device, between a simple mode, standalone mode and a host centric mode. Standalone mode means no control from the host; host centric mode means all control is performed from the host.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_OPERATION_ PARMS	Operation Mode	Interface	Zero	None

The *wValue* field is used to specify the mode of operation to be used. Current supported modes of operation are defined in the following table:

Table 53: Operation Mode Values

Operation mode	Description
0	Simple Mode Communication device operates in standalone fashion, and sends no status information to the host and accepts only SetOperationMode commands from host. The device is capable of independent operation..
1	Standalone Mode Communication device operates in standalone fashion, but sends complete status information to the host and will accept any command from the host.
2	Host Centric Mode Communication device is completely controlled by computer but will not perform any communication functions without host control.

In the case of dialing on a phone device, mode 0 would correspond to operating as a typical phone, where the phone would dial out the digits over the phone line. Mode 1 would be the same, except each of the digits dialed by the phone would be reported to the host. In mode 2, the phone would simply report which digits were pushed on the phone keypad to the host, and the host would be responsible for dialing the digits over the phone line.

6.2.19 GetOperationParms

This request gets the current operational mode for the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_OPERATION_ PARMS	Zero	Interface	2	Operation mode

The returned operation mode value describes the current operational mode of the device, as specified in Table 53.

6.2.20 SetLineParms

This request is used to change the state of the line, corresponding to the interface or master interface of a union to which the command was sent.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_LINE_PARMS	Line State Change	Interface	Length of Data	Data

Some of the commands will require extra data, which will be provided in a packet transmitted during the Data phase. Current line state change values supported are defined in the following table:

Table 54: Line State Change Value Definitions

Line State change value	Description
0000h	Drop the active call on the line.
0001h	Start a new call on the line.
0002h	Apply ringing to the line.
0003h	Remove ringing from the line.
0004h	Switch to a specific call on the line. Data is used to pass a 1-byte call index that identifies the call.

6.2.21 GetLineParms

This request is used to report the state of the line that corresponds to the interface or master interface of a union to which the command was sent.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_LINE_PARMS	Zero	Interface	Size of Structure	Line Status Information Structure

This command is issued to the interface or master interface of a union representing a specific line. The returned Line Status Information structure is defined in the following table:

Table 55: Line Status Information Structure

Offset	Field	Size	Value	Description
0	*wLength*	2	Number	Size of this structure, in bytes.
2	*dwRingerBitmap*	4	Bitmap	Ringer Configuration bitmap for this line. For the format of this field, see Table 52.
6	*dwLineState*	4	Bitmap	Defines current state of the line.
10	*dwCallState0*	4	Bitmap	Defines current state of first call on the line.
…	…	…	…	
6 + N*4	*dwCallStateN-1*	4	Bitmap	Defines current state of call N on the line.

The Line State bitmap format provided within the line status information is defined in the following table:

Table 56: Line State Bitmap

Bit position	Description
D31	Active flag 0 - No activity on the line. 1 - Line is active (i.e. not idle).
D30..D8	RESERVED (Reset to zero)
D7..D0	Index of active call on this line. Equals 255 if no call exists on the line.

The Call State bitmap format provided within the line status information is defined in the following table:

Table 57: Call State Bitmap

Bit position	Description
D31	Active flag 0 - No active call. 1 - Call is active (i.e., not idle).
D30..D16	RESERVED (Reset to zero)
D15..D8	Call state change value. (For definitions of call state change values, see Table 70.)
D7..D0	Call state value. (For definitions of call state values, see Table 58.)

Table 58: Call State Value Definitions

Call state value	Description
00h	Call is idle.
01h	Typical dial tone.
02h	Interrupted dial tone.
03h	Dialing is in progress.
04h	Ringback. Call state additional data, D15..D8, contains extra information, as defined in Table 70.
05h	Connected. Call state additional data, D15..D8, contains extra information, as defined in Table 70.
06h	Incoming call. Call state additional data, D15..D8, contains extra information, as defined in Table 70.

6.2.22 DialDigits

This request dials the DTMF digits over the specified line.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	DIAL_ DIGITS	Zero	Interface	Length of Dialing String	Dialing string

The data packet consists of a dialing command, with only the following characters in V.4 supported as being part of the command:

Table 59: Characters in a Dialing Command

Characters	Action
0-9	Dial the specified digit.
* #	Dial the specified DTMF key.
P p	Use pulse dialing for dialing all subsequent digits.
T t	Use tone dialing for dialing all subsequent digits.
!	Insert a hook switch flash into the dialing string.

Characters	Action
,	(Comma) Pause the dialing for a fixed period of time defined by the device (usually 2 seconds).
;	(Semicolon) Indicates that more digits will be provided later.
W w	Wait for dial tone or interrupted dial tone before continuing to dial digits.
D d	Hold tone on. All subsequent dialing tones are left on until hold tone off is received.
U u	Hold tone off. All held dialing tones are turned off.

6.2.23 SetUnitParameter

This request sets the value of a parameter belonging to a Unit identified by Unit Parameter Structure, see Table 60. The timing of when the new parameter takes effect depends on the protocol or vendor specific function.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_UNIT _PARAMETER	Unit Parameter Structure	Interface	Length of Unit Parameter	Unit Parameter

Table 60: Unit Parameter Structure

Offset	Field	Size	Value	Description
0	*bEntityId*	1	Number	Unit Id
1	*bParameterIndex*	1	Number	A zero based value indicating Unit parameter index.

6.2.24 GetUnitParameter

This request returns the current value of a parameter belonging to a Unit pointed out by Unit Parameter Structure, see Table 60.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_UNIT _PARAMETER	Unit Parameter Structure	Interface	Length of Unit Parameter	Unit Parameter

6.2.25 ClearUnitParameter

This request restores the default value of a parameter belonging to a Unit identified by Unit Parameter Structure, see Table 60. The timing of when the new parameter takes effect depends on the protocol or vendor specific function.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	CLEAR_UNIT _PARAMETER	Unit Parameter Structure	Interface	Zero	None

6.2.26 GetProfile

This request returns the profile information as defined by CAPI 2.0. The profile describes the implemented capabilities of the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_PROFILE	Zero	Interface	64	Profile Information according to CAPI 2.0 chapter 8

6.2.27 SetEthernetMulticastFilters

This request sets the Ethernet device multicast filters as specified in the sequential list of 48 bit Ethernet multicast addresses.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_ MULTICAST_FILTER	Number of filters (N)	Interface	N * 6	A list of N 48 bit Multicast addresses, in network byte order

If the host wishes to change a single multicast filter in the device, it must reprogram the entire list of filters using this request. This sequential programming method for the entire multicast list is well-suited for devices that use hashing techniques.

Although the Data field for this request might be quite large, devices with limited buffering capacity may use NAKs as necessary to process (e.g., hash) a small number of multicast addresses at a time.

6.2.28 SetEthernetPowerManagementPatternFilter

This request sets up the specified Ethernet power management pattern filter as described in the data structure.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_ POWER_ MANAGEMENT_ PATTERN_FILTER	Filter number	Interface	Size of structure	Power management pattern filter structure

Some hosts are able to conserve energy and stay quiet in a “sleeping” state while not being used. USB Networking devices may provide special pattern filtering hardware that enables it to wake up the attached host on demand when something is attempting to contact the host (e.g., an incoming web browser connection).

NOTE: To enable remote wake up, additional steps must be completed that are described in the *USB Specification*.

If the host simply wishes to clear (remove) any previous setting for the specified pattern filter, the value of wLength is set to Zero and no Data field (pattern filter structure) follows.

If the specified pattern is not able to fit into the device, any pattern previously loaded is considered destroyed, and the device must set a value of FALSE (0x0000) into the associated status that will be read by the *GetEthernetPowerManagementPatternFilter* request.

Table 61: Power Management Pattern Filter Structure

Field	Size	Value	Description
MaskSize	2	Number	Contains the size (in bytes) of the Mask.
Mask	MaskSize	Bitmask	Each byte of Mask contains 8 masking bits, where each one of them represents whether or not the associated byte of the Pattern should be compared with what is seen on the media by the networking device. The least significant bit (D0) of the first Mask byte is associated with the first byte of the pattern, which starts at the Destination Address (DA) of the Ethernet frame. If the last byte of Mask contains trailing zeros in its highest order bits, the associated bytes in the Pattern field are not sent in this request.
Pattern	Specified by Mask	Number	This is a string of bytes to perform pattern matching on, starting from offset 0 of the Ethernet frame (the Destination Address).

6.2.29 GetEthernetPowerManagementPatternFilter

This request retrieves the status of the specified Ethernet power management pattern filter from the device. If the device has an active pattern set for the specified filter, a TRUE (0x0001) will be returned. If a FALSE (0x0000) is returned, either no pattern has yet been set for the specified filter, or the prior attempt by the host software to set this filter was not successful (i.e., was not able to fit).

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ETHERNET_ POWER_ MANAGEMENT_ PATTERN_FILTER	Filter number	Interface	2	Pattern active boolean

6.2.30 SetEthernetPacketFilter

This request is used to configure device Ethernet packet filter settings. The Packet Filter is the inclusive OR of the bitmap shown in Table 62. Though network adapters for faster buses (*e.g.*, PCI) may offer other hardware filters, the medium speed networking devices (< 10Mbit/s) attached via USB are only required to support promiscuous and all multicast modes. The host networking software driver is responsible for performing additional filtering as required.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ETHERNET_ PACKET_FILTER	Packet Filter Bitmap	Interface	Zero	None

Note that for some device types, the ability to run in promiscuous mode may be severely restricted or prohibited. For example, DOCSIS cable modems are only permitted to forward certain frames to its attached host. Even if forwarding of all frames were allowed, the raw cable modem downstream rate available on the RF interface can be many times the maximum USB throughput.

Table 62: Ethernet Packet Filter Bitmap

Bit position	Description
D15..D5	RESERVED (Reset to zero)
D4	PACKET_TYPE_MULTICAST 1: All multicast packets enumerated in the device's multicast address list are forwarded up to the host. (required) 0: Disabled. The ability to disable forwarding of these multicast packets is optional. ***
D3	PACKET_TYPE_BROADCAST 1: All broadcast packets received by the networking device are forwarded up to the host. (required) 0: Disabled. The ability to disable forwarding of broadcast packets is optional. ***
D2	PACKET_TYPE_DIRECTED 1: Directed packets received containing a destination address equal to the MAC address of the networking device are forwarded up to the host (required) 0: Disabled. The ability to disable forwarding of directed packets is optional. ***
D1	PACKET_TYPE_ALL_MULTICAST 1: ALL multicast frames received by the networking device are forwarded up to the host, not just the ones enumerated in the device's multicast address list (required) 0: Disabled.
D0	PACKET_TYPE_PROMISCUOUS: 1: ALL frames received by the networking device are forwarded up to the host (required) 0: Disabled.

*** Support for inhibiting (Dx = 0) the forwarding of "*ordinary*" directed, multicast and broadcast packets to the host is optional. Since there are no associated descriptors for the device to designate which filters are supported by a particular device, the host must blindly set these bits as desired, filtering out these undesired packets in host software should they appear.

6.2.31 GetEthernetStatistic

This request is used to retrieve a statistic based on the feature selector. The value returned indicates the number of matching frames with the specified statistic that have occurred since the device has been powered on or reset. This number is a 32 bit unsigned integer, which is incremented at each occurrence, and will be wrapped to 0 if reaching the maximum value.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ETHERNET_ STATISTIC	Feature Selector	Interface	4	32 bit unsigned integer

Table 63: Ethernet Statistics Feature Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
XMIT_OK	01h	Interface	4	Frames transmitted without errors
RCV_OK	02h	Interface	4	Frames received without errors
XMIT_ERROR	03h	Interface	4	Frames not transmitted, or transmitted with errors
RCV_ERROR	04h	Interface	4	Frames received with errors
RCV_NO_BUFFER	05h	Interface	4	Frames missed, no buffers
DIRECTED_BYTES_XMIT	06h	Interface	4	Directed bytes transmitted without errors
DIRECTED_FRAMES_XMIT	07h	Interface	4	Directed frames transmitted without errors
MULTICAST_BYTES_XMIT	08h	Interface	4	Multicast bytes transmitted without errors
MULTICAST_FRAMES_ XMIT	09h	Interface	4	Multicast frames transmitted without errors
BROADCAST_BYTES_XMIT	0Ah	Interface	4	Broadcast bytes transmitted without errors
BROADCAST_FRAMES_ XMIT	0Bh	Interface	4	Broadcast frames transmitted without errors
DIRECTED_BYTES_RCV	0Ch	Interface	4	Directed bytes received without errors
DIRECTED_FRAMES_RCV	0Dh	Interface	4	Directed frames received without errors
MULTICAST_BYTES_RCV	0Eh	Interface	4	Multicast bytes received without errors
MULTICAST_FRAMES_ RCV	0Fh	Interface	4	Multicast frames received without errors
BROADCAST_BYTES_RCV	10h	Interface	4	Broadcast bytes received without errors
BROADCAST_FRAMES_ RCV	11h	Interface	4	Broadcast frames received without errors
RCV_CRC_ERROR	12h	Interface	4	Frames received with circular redundancy check (CRC) or frame check sequence (FCS) error
TRANSMIT_QUEUE_ LENGTH	13h	Interface	4	Length of transmit queue
RCV_ERROR_ALIGNMENT	14h	Interface	4	Frames received with alignment error
XMIT_ONE_COLLISION	15h	Interface	4	Frames transmitted with one collision
XMIT_MORE_COLLISIONS	16h	Interface	4	Frames transmitted with more than one collision
XMIT_DEFERRED	17h	Interface	4	Frames transmitted after deferral
XMIT_MAX_COLLISIONS	18h	Interface	4	Frames not transmitted due to collisions
RCV_OVERRUN	19h	Interface	4	Frames not received due to overrun
XMIT_UNDERRUN	1Ah	Interface	4	Frames not transmitted due to underrun
XMIT_HEARTBEAT_ FAILURE	1Bh	Interface	4	Frames transmitted with heartbeat failure

Feature selector	Code	Targets	Length of Data	Description
XMIT_TIMES_CRS_LOST	1Ch	Interface	4	Times carrier sense signal lost during transmission
XMIT_LATE_COLLISIONS	1Dh	Interface	4	Late collisions detected

Refer to the ISO/IEC 8802-3 (ANSI/IEEE Std 802.3) specification for additional information on the meaning of each of these statistics.

6.2.32 SetATMDataFormat

This request is used to set the data format selected by the host in the *wValue* field.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ATM_DATA_ FORMAT	Data Format	Interface	Zero	None

Table 64: ATM Data Format

wValue	Description
1	Type 1 format: concatenated ATM cells
2	Type 2 format: ATM header template + concatenated ATM cell payloads
3	Type 3 format: AAL 5 SDU

The ATM Networking Control Model data format selected specifies how much processing is done in the USB device, versus how much is done in the host. Specifically, the Type 3 data format requires that ATM segmentation and re-assembly (SAR) functions be implemented in the USB device, while Type 1 and Type 2 data formats enable different functional partitioning, and migrate the SAR function to the host. As you move from Type 1 to Type 3, the amount of processing performed in the USB device increases. Support of Type 1 Data Format is mandatory to ensure minimal interoperability. If supported by the device, it is recommended that Type 3 Data Format be chosen by the host whenever possible.

Type 1 Data Format

Type 1 ATM oriented USB devices (e.g., ADSL modems or IEEE 802.14 cable modems) do the least amount of processing on the incoming data and are therefore the simplest types of such devices. The main function of the USB device is to pass the ATM cells from the WAN link to the host, and vice versa. The data format consists of a number of concatenated 53 byte ATM cells. The HEC field in the ATM cell header exists only as a placeholder when transferred over USB, and will be generated and verified by the ATM-oriented device for upstream and downstream traffic, respectively.

For Type 1 devices, all ATM and AAL functions are performed by the host, e.g. AAL layer encapsulation, ATM SAR, and traffic shaping (for upstream direction only). Various AAL types, Operation Administration and Maintenance (OAM) cells, and Resource Management (RM) cells are enabled, if supported by the host.

Type 2 Data Format

The Type 2 ATM-oriented USB device improves USB bus bandwidth efficiency by removing duplicate ATM cell headers prior to transfer over USB. This data format consists of a 4-byte ATM cell header template (excluding the

HEC field) followed by a number of 48-byte ATM cell payloads. As with Type 1 devices, the AAL encapsulation, SAR and traffic shaping are all performed by the host.

The Type 2 device needs to assemble cells based on the header template before transmitting them over the WAN link. All the AAL types are supported. OAM cells for flow management are enabled, as are RM cells for the ABR service. A unique requirement for this type of device is that all cell payloads must share the same cell header template. To accomplish this, extra processing rules need to be applied in the host and USB device as follows:

- The 48-octet cell payloads shall be contiguous without additional intervening cell headers, all destined to the same VC, and have the same payload type.

- For AAL5 VCs, the ATM-user-to-ATM-user indication bit in the payload type is used to delimit the AAL5 CPCS PDU. This bit shall be set if the last cell in this segment (i.e., data between 2 consecutive USB short packets) completes an AAL5 CPCS PDU. Note that in this case no other cells can be appended in the same segment after an AAL 5 CPCS PDU has been completed. The device at the other end of the USB bus has to set the ATM-user-to-ATM-user indication bit in the payload type only for the last cell in this segment, but not for any preceding cells, before forwarding them.

- The ATM cell header template transferred between the USB device and host can not include the 8-bit HEC field.

Also, the USB device will have to perform the following functions:

- Generate and insert a HEC field for upstream traffic.

- Verify and remove the HEC field from the downstream traffic.

- Discard cells with HEC errors from the downstream traffic.

For Type 1 and Type 2 ATM oriented USB devices, any number of ATM cells or payloads could be concatenated in an USB buffer as long as it adheres to the rules stipulated above. The general guideline to flush the buffer and send it via USB is when:

- The maximum segment size (*wType2MaxSegmentSize*) is reached (for Type 2 only), or

- Encounter the end of an AAL5 PDU, or

- The timer expires (so you don't hold the cells too long if there are no immediate incoming cells), or

- One or more cells from a low-latency VC are received. NOTE: If a grouping of low latency VC cells are received (e.g., a 20ms audio frame), it is recommended that devices not "flush" (generate a USB short packet) after every ATM cell in this instance to improve link efficiency and reduce host overhead. The methods for accomplishing this are beyond the scope of this document

Type 3 Data Format

Type 3 ATM oriented devices will process the AAL5 SDUs (e.g., Q.2931, ILMI and other SDU's like PPP packets in data VCs) as they arrive from the host, before sending them to the WAN link. This is accomplished by adding AAL5 encapsulation (including the appropriate padding and the CRC generation) to form the AAL5 PDU. The AAL5 PDU formed then goes through the SAR function before being transmitted over the WAN link.

For the ATM cells arriving from the WAN link, the device will reassemble them into AAL5 PDUs, validate the AAL5 CRC (discard the AAL5 PDU if incorrect), strip off the AAL5 encapsulation and transfer AAL5 SDU to the host.

This data format consists of the first 4 bytes of an ATM cell header (excluding the HEC field) followed by a single AAL5 SDU. The VPI/VCI value in the cell header indicates which VC this AAL5 SDU belongs to and the first bit of the PTI field shall be set to 0 for AAL5 SDUs. OAM and RM cells can be supported by sending those cells with the corresponding PTI bits set in the cell header.

6.2.33 GetATMDeviceStatistics

This request is used to retrieve the device statistics based on the feature selector.

BmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ATM_DEVICE_ STATISTICS	Feature Selector	Interface	4	32 bit unsigned integer

In Table 65, the terms UPSTREAM(US) and DOWNSTREAM(DS) refer to the direction of the data flow across the connection between the device and the network.

The value returned indicates the number of cells matching the specified statistic that have occurred since the device has been powered on or reset, or since the SET_INTERFACE request has been received (see Section 3.8.1 for more details). This number is a 32 bit unsigned integer, which is incremented at each occurrence and will be wrapped to zero upon reaching the maximum value.

Table 65: ATM Device Statistics Feature Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
US_CELLS_SENT	01h	Interface	4	The number of cells that have been sent upstream to the WAN link by the ATM layer. Support for this statistic by device hardware is optional.
DS_ CELLS_RECEIVED	02h	Interface	4	The number of cells that have been received downstream from the WAN link by the ATM layer. Support for this statistic by device hardware is optional.
DS_CELLS_USB_ CONGESTION	03h	Interface	4	The number of cells that have been received downstream from the WAN link by the ATM layer and discarded due to congestion on the USB link. Support for the feature code by the device is MANDATORY.
DS_CELLS_AAL5_ CRC_ERROR	04h	Interface	4	The number of cells that have been received downstream from the WAN link by the ATM layer and discarded due to AAL5 CRC errors. Support for this feature code by Type 3 devices is MANDATORY.
DS_CELLS_HEC_ ERROR	05h	Interface	4	The number of cells that have been received downstream from the WAN link and discarded due to HEC errors in the cell header. Support for this statistic by device hardware is MANDATORY.
DS_CELLS_HEC_ ERROR_CORRECTED	06h	Interface	4	The number of cells that have been received downstream from the WAN link and have been detected with HEC errors in the cell header and successfully corrected. Support for this statistic by device hardware is optional.

6.2.34 SetATMDefaultVC

This request is used to pre-select the VPI/VCI value for subsequent GetATMVCStatistics requests. This request only applies to type 3 devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ATM_DEFAULT_VC	Zero	Interface	3	1-byte VPI followed by 2-byte VCI value

6.2.35 GetATMVCStatistics

This request is used to retrieve the ATM device statistics based on the feature selector for a pre-selected VPI/VCI as stipulated in latest preceding SetATMDefaultVC request. This request only applies to type 3 devices.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ATM_VC_ STATISTICS	Feature Selector	Interface	4	32 bit unsigned integer

Table 66: ATM VC Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
VC_US_CELLS_SENT	01h	Interface	4	The number of cells that have been sent upstream to the WAN link for the specified VPI/VCI since the device has been powered on or reset. This number is a 32 bit unsigned integer, which is incremented each time a cell is sent, and will be wrapped to 0 if reaching the maximum value. Support for this statistic by the device is optional.
VC_DS_ CELLS_RECEIVED	02h	Interface	4	The number of cells that have been received downstream from the WAN link for the specified VPI/VCI since the device has been powered on or reset. This number is a 32 bit unsigned integer, which is incremented each time a cell is received, and will be wrapped to 0 if reaching the maximum value. Support for this statistic by the device is optional.

6.3 Notification Element Notifications

This section defines the Communication Interface Class notifications that the device uses to notify the host of interface, or endpoint events.

Table 67: Class-Specific Notifications

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	NETWORK_ CONNECTION	0 – Disconnect 1 – Connected	Interface	Zero	None
10100001B	RESPONSE_AVAILABLE	Zero	Interface	Zero	None
10100001B	AUX_JACK_HOOK_ STATE	0 – On hook 1 – Off hook	Interface	Zero	None
10100001B	RING_DETECT	Zero	Interface	Zero	None
10100001B	SERIAL_STATE	Zero	Interface	2	UART State bitmap
10100001B	CALL_STATE_CHANGE	Call Index and Call State Change Value	Interface	Length of Data	Variable-length structure containing additional information for call state change.
10100001B	LINE_STATE_CHANGE	Value	Interface	Length of Data	Variable length Line State structure.
10100001B	CONNECTION_SPEED_ CHANGE	Zero	Interface	8	Connection Speed Change Data Structure

Table 68: Class-Specific Notification Codes

Notification	Value
NETWORK_CONNECTION	00h
RESPONSE_AVAILABLE	01h
RESERVED (future use)	02h-07h
AUX_JACK_HOOK_STATE	08h
RING_DETECT	09h
RESERVED (future use)	0Ah-1Fh
SERIAL_STATE	20h
RESERVED (future use)	21h-27h
CALL_STATE_CHANGE	28h
LINE_STATE_CHANGE	29h
CONNECTION_SPEED_CHANGE	2Ah
RESERVED (future use)	2Bh-FFh

6.3.1 NetworkConnection

This notification allows the device to notify the host about network connection status.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	NETWORK_ CONNECTION	0 - Disconnect 1 - Connected	Interface	Zero	None

6.3.2 ResponseAvailable

This notification allows the device to notify the host that a response is available. This response can be retrieved with a subsequent **GetEncapsulatedResponse** request.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	RESPONSE_AVAILABLE	Zero	Interface	Zero	None

6.3.3 AuxJackHookState

This notification indicates the loop has changed on the auxiliary phone interface of the USB device. The secondary or downstream device, which is connected to the auxiliary phone interface, has changed hook states.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	AUX_JACK_ HOOK_STATE	0 – On hook 1 – Off hook	Interface	Zero	None

On devices that provide separate control of the auxiliary or downstream phone interface, this notification provides a means of announcing hook state changes of devices plugged into that interface. When the USB device has separate control of this phone interface, it is helpful to know when the secondary device, which is plugged into the auxiliary phone interface, switches between the on-hook/off-hook states.

The *wValue* field returns whether loop current was detected or not detected. Notification is only sent when the state changes.

6.3.4 RingDetect

This notification indicates ring voltage on the POTS line interface of the USB device.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	RING_DETECT	Zero	Interface	Zero	None

6.3.5 SerialState

This notification sends asynchronous notification of UART status.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	SERIAL_STATE	Zero	Interface	2	UART State bitmap

The *Data* field is a bitmapped value that contains the current state of carrier detect, transmission carrier, break, ring signal, and device overrun error. These signals are typically found on a UART and are used for communication status reporting. A state is considered enabled if its respective bit is set to 1.

SerialState is used like a real interrupt status register. Once a notification has been sent, the device will reset and re-evaluate the different signals. For the consistent signals like carrier detect or transmission carrier, this will mean another notification will not be generated until there is a state change. For the irregular signals like break, the incoming ring signal, or the overrun error state, this will reset their values to zero and again will not send another notification until their state changes.

Table 69: UART State Bitmap Values

Bits	Field	Description
D15..D7		RESERVED (future use)
D6	*bOverRun*	Received data has been discarded due to overrun in the device.
D5	*bParity*	A parity error has occurred.
D4	*bFraming*	A framing error has occurred.
D3	*bRingSignal*	State of ring signal detection of the device.
D2	*bBreak*	State of break detection mechanism of the device.
D1	*bTxCarrier*	State of transmission carrier. This signal corresponds to V.24 signal 106 and RS-232 signal DSR.
D0	*bRxCarrier*	State of receiver carrier detection mechanism of device. This signal corresponds to V.24 signal 109 and RS-232 signal DCD.

6.3.6 CallStateChange

This notification identifies that a change has occurred to the state of a call on the line corresponding to the interface or union for the line.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	CALL_STATE_CHANGE	Call Index and Call State Change Value.	Interface	Length of Data	Variable length structure containing additional information for call state change.

The high-order byte D15-D8 of the *wValue* field will contain the call index, and the low-order byte D7-D0 will contain the call state change value. Not all devices may be capable of reporting all changes of the call state, which should not cause any problems to the higher-level software. All extra data associated with a call state change (*i.e.*, Caller ID data) is returned within the data field. Currently, defined call state values are listed in the following table:

Table 70: Call State Change Value Definitions

Call state change	Description
00h	RESERVED
01h	Call has become idle.
02h	Dialing.
03h	Ringback, with an extra byte of data provided to describe the type of ringback signaling 0 = normal 1 = busy 2 = fast busy 3-254 = reserved for future use 255=unknown ringback type
04h	Connected, with an extra byte of data provided to describe the type of connection 0 = voice connection 1 = answering machine connection 2 = fax machine connection 3 = data modem connection 4-254 = reserved for future use 255 = unknown connection type
05h	Incoming Call, with the following extra bytes of data (minimum of 4 extra bytes): Extra data byte 0 - Indicates the ringing pattern present as: 0 = ringing pattern 1 (default or normal pattern) 1 = ringing pattern 2 2 = ringing pattern 3 3 – ringing pattern 4 4-255 = reserved for future use Extra data byte 1 - Size of the string (next n bytes) which contains the time (in displayable format) of the incoming call as delivered via Caller ID. The string is not null terminated and is encoded using one character per byte. It is not a UNICODE string. If time is not available then a size of 0 is required as a place setter. Next data byte following number - Size of string (next n bytes) which contains the phone number of calling party as delivered via Caller ID. The string is not null terminated and is encoded using one character per byte. It is not a UNICODE string. If no number is available then a size of 0 is required as a place-setter. Next data byte following name - Size of string (next n bytes) which contains the name of the calling party as delivered via Caller ID. The string is not null terminated and is encoded using one character per byte. It is not a UNICODE string. If no name is available then a size of 0 is required as a place-setter.

6.3.7 LineStateChange

This notification identifies that a change has occurred to the state of the line corresponding to the interface or master interface of a union sending the notification message.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	LINE_STATE_CHANGE	Value	Interface	Length of Data	Variable Length Line State structure.

Some line state changes may provide extra information, and this information would be provided in the attached extra Line State data structure. Current line state change information are defined in the following table:

Table 71: Line State Change Values

Line State change	Description
0000h	Line has become idle.
0001h	Line connected to hold position.
0002h	Hook-switch has gone off hook.
0003h	Hook-switch has gone on hook.

6.3.8 ConnectionSpeedChange

This notification allows the device to inform the host-networking driver that a change in either the upstream or the downstream bit rate of the connection has occurred.

bmRequestType	bNotification	wValue	wIndex	wLength	Data
10100001B	CONNECTION_SPEED_CHANGE	Zero	Interface	8	Connection Speed Change Data Structure

The data phase for this notification contains a data structure with two 32 bit unsigned integers. The two values are the upstream bit rate, followed immediately by the downstream bit rate. (Table 72)

To assure that the host networking driver can always report the correct link speed, the device must send this notification immediately after every NETWORK_CONNECTION notification is sent. This normally occurs when the physical layer makes or loses a connection, but additionally appears implicitly after the device is reset (see discussion in Section 3.8.1 Common Data Plane Characteristics).

Table 72: ConnectionSpeedChange Data Structure

Offset	Field	Size	Value	Description
0	USBitRate	4	Number	Contains the upstream bit rate, in bits per second
4	DSBitRate	4	Number	Contains the downstream bit rate, in bits per second.

Appendix A: Communication Device Class Examples

This appendix highlights some examples of typical communication device classes. Detailed examples are provided in separate white papers that are not a part of this specification. The latest copies of the white papers can be found at **http://www.usb.org**.

A.1 Basic Telephone

A basic telephone is defined as the household/desktop type phone common to most users. This phone has a handset, keypad, and a 2-wire connection to a local telephone company. In this example, a USB port is added for connecting the phone to the host.

By connecting the phone to a host via the USB, the following functions can be supported:

1. Host monitoring of incoming and outgoing calls.
2. Host-originated dialing of a call.
3. Host recording and playback of voice over the phone line.

This example is not intended to define the computer telephony application features or user interface. The example demonstrates how the USB Communication Interface Class protocol can be used to identify, control, and monitor a telephony device.

A.2 Modem

For compatibility with legacy computer software and to facilitate the development of generic drivers, a USB modem should conform to the ANSI/TIA-602 standard. For common extended functions, the following standards are recommended:

- Modem identification: ITU V.25ter +G commands
- Data modems: ITU V.25ter (modulation, error control, data compression)
- Data modems: ITU V.80 In-band DCE control and synchronous data modes for asynchronous DTE
- Fax modems: ITU T.31 or T.32 +F commands (or TIA equivalents)
- Voice modems: TIA IS-101 +V commands
- General wireless modems: PCCA STD-101 +W commands (TIA 678)
- Analog cellular modems: PCCA STD-101 Annex I (TIA 678 Annex C)
- Digital cellular modems: TIA IS-707, TIA IS-135 or GSM 7.07 +C commands.
- Text phone modems: V.25ter, +MV18 commands.

For a complete list of standard modem command sets, see the ITU Supplement to V.25ter.

Note: A USB modem may provide means to accommodate common functions performed on a 16550 UART. For more information, see Section 3.6.2.1, “Abstract Control Model Serial Emulation.”

A.3 CAPI Device

For compatibility with existing CAPI software, and to facilitate the development of generic adaptive drivers, a USB CAPI Device has to conform to the CAPI 2.0 specification.

Appendix B: Sample Configurations

B.1 Basic Telephony Configurations

This section defines three examples of telephony configurations: a basic telephone, a telephone with keypad, and a combination telephone with analog modem. The minimum requirement for this type of device is a configuration with a single Communication Class interface. If you wish to support a standard telephone keypad, you would require an additional Human Interface Device Class interface to support the keypad. The most basic audio-capable telephone is constructed by adding an Audio interface for audio transmission and reception. A more advanced configuration could optionally have local Audio interfaces connected to the handset and microphone/speaker, and one Data interface. In this case, the Data interface could be the raw linear data as sampled from the network. The responsibility for demodulation and interpretation of this data would lie within the host at the application level (i.e., processor-based modem).

Table 73: Telephone Configurations

Example configuration	Interface (class code)	Reference section	Description
Basic telephone	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
Telephone with keypad	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
	HID Class	HID 1.0	I/O for a keypad interface.
Audio/data telephone	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
	Audio Class	Audio 1.0	I/O for uncompressed audio.
	Data Class	3.3.2	Demodulated modem data.

A communication device that supports audio type media streams over its interfaces can use the selected Audio interface to indicate which voice or audio coding formats it supports (for example, IS-101 for voice modems).

B.2 Modem Configurations

This section defines three examples of modem configurations: legacy modem, DSVD modem, and multimedia modem. The first configuration covers legacy modems for data, fax, and voice. The second configuration covers SVD modems, such as ASVD (ITU V.61) and DSVD (ITU V.70). The third configuration covers multimedia modems that would be used in ITU H.324 situation.

Table 74: Example Modem Configurations

Example configuration	Interface (class code)	Reference section	Description
Legacy modem	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
	Data Class	3.3.2	Demodulated modem data.

Table 74: Example Modem Configurations

Example configuration	Interface (class code)	Reference section	Description
SVD modem	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
	Data Class	3.3.2	Demodulated modem data.
	Audio Class	Audio 1.0	I/O for uncompressed audio.
Multimedia modem	Communication Class	3.3.1	Device management and call management. Consisting of a management element and a notification element.
	Data Class	3.3.2	Demodulated modem data.
	Audio Class	Audio 1.0	I/O for uncompressed audio.
	Image Class	TBD	I/O for video (for example, H.263).

Most of today's modem type devices — single-media or multimedia — contain various types of media processing resources such as compression engines (for example, V.42bis) or audio/video CODECs. Given the projections of increased processing power for future host systems and the availability of appropriate media transport to and from the host (i.e., the USB), it is likely that various models of media processing will emerge that do not rely solely on the device for these resources. In this case, where media processing resources are located arbitrarily within the system (for example, V.42bis on the host and V.34 on the device), interface choices for media types could vary. For example, if a device developer chose to include an MPEG2 CODEC in a device, a bi-directional isochronous interface may be more appropriate for transport of the video stream. Conversely, if the CODEC is not in the USB device, a bi-directional bulk interface would be more appropriate.

The processing required for some types of media streams is asymmetrical in nature. For example, MPEG2 decompression is trivial by today's standards, although compression requires substantial processing resources. In light of this fact, it may be appropriate to configure an interface with the appropriate asymmetry. Continuing the MPEG example, a device that relies on host-based decompression and device-based compression would choose an interface that consists of an isochronous endpoint for video in the host-to-device direction and a bulk endpoint for the device-to-host direction.

An example of the bandwidth implications of device in contrast with host-based media stream processing is outlined in the following list. USB bandwidth is expressed in bytes per millisecond (B/ms). For example, typical performance of V.42bis is 3-4:1 on data streams, 33.6 kb/s V.34 data could unpack to 16.8 B/ms.

Similar bandwidth issues are relevant to audio and video, but they will be handled by the video or audio interfaces rather than Communication Class interfaces.

- G.723 voice CODEC (5.5 - 6.5 Kb/s) could be unpacked to 11 kHz audio by the modem (22 B/ms).
- H.263 compressed video is in the 2.5 B/ms range; but if H.263 is decompressed, a typical bandwidth is approximately 96 B/ms.

B.3 CAPI Device Configuration

This section describes two examples of configurations of CAPI Devices. The first configuration covers the intelligent CAPI Device which implements the whole CAPI functionality on the device including call management (Q.931, NI-1, 5ESS, GSM, etc.) and the variety of the supported B channel protocols such as X.75, X.25, T.30, V.42bis, X.31, V.110, V.120 etc. The second configuration covers the simple CAPI Device which implements CAPI conform access to the Layer 1.

Table 75: Example CAPI Device Configurations

Example Configuration	Interface (Class Code)	Reference Section	Description
Intelligent CAPI Device	Communication Class	3.3.1	Device Management consisting of a Management Element
	Data Class	3.3.2	CAPI Messages
Simple CAPI Device	Communication Class	3.3.1	Device Management consisting of a Management Element
	Data Class	3.3.2	CAPI Messages for simple CAPI Devices

Appendix C: Multi-channel ISDN B-Channel setup

C.1 General

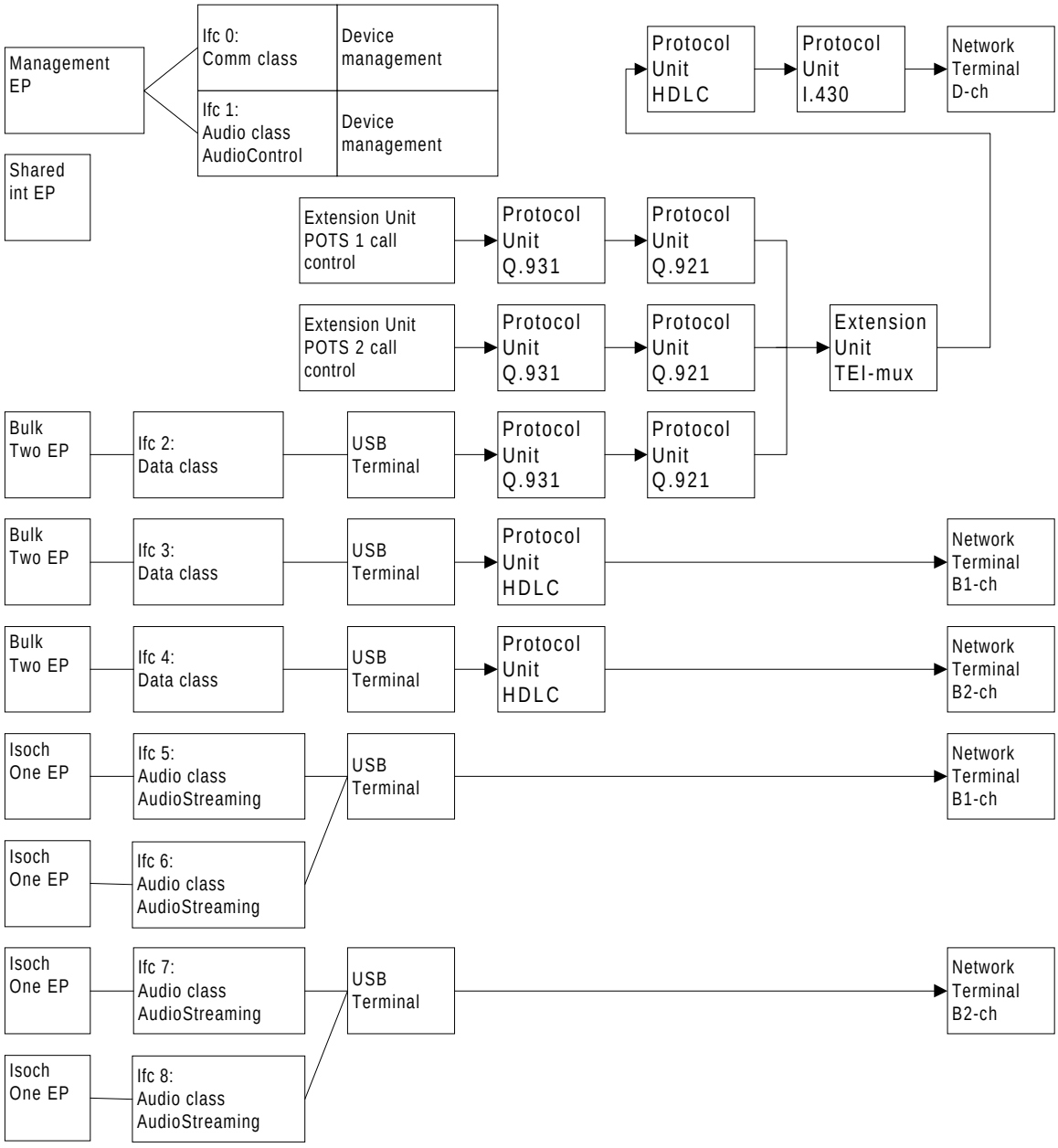
The basic idea is that call control to/from the host is performed through some sort of protocol state machine. This state machine will be specific for each protocol due to the problems in finding one generic that will suit all possible protocol derivatives that are in use, i.e. for ISDN layer 3 there are for example Q.931 (ITU), DSS1 (Europe), 1TR6 (Germany), VN4 (France), DMS100 Custom (USA), 5ESS Custom (USA), NI-1 (USA), NTT (Japan) and NS2 (USA). The call control protocol is accessed through a Data Class Interface. This is to enable a flexible interface with as few restrictions as possible on the protocol.

Each channel (2B+D) on the physical interface is represented by an interface with appropriate Interface Descriptors and Terminal/Unit Functional Descriptors if needed. The interface connected to the D-channel may then run a call control protocol stack starting with I.430 and some Framing, Data link and Network protocols (HDLC,Q.921 and Q.931).

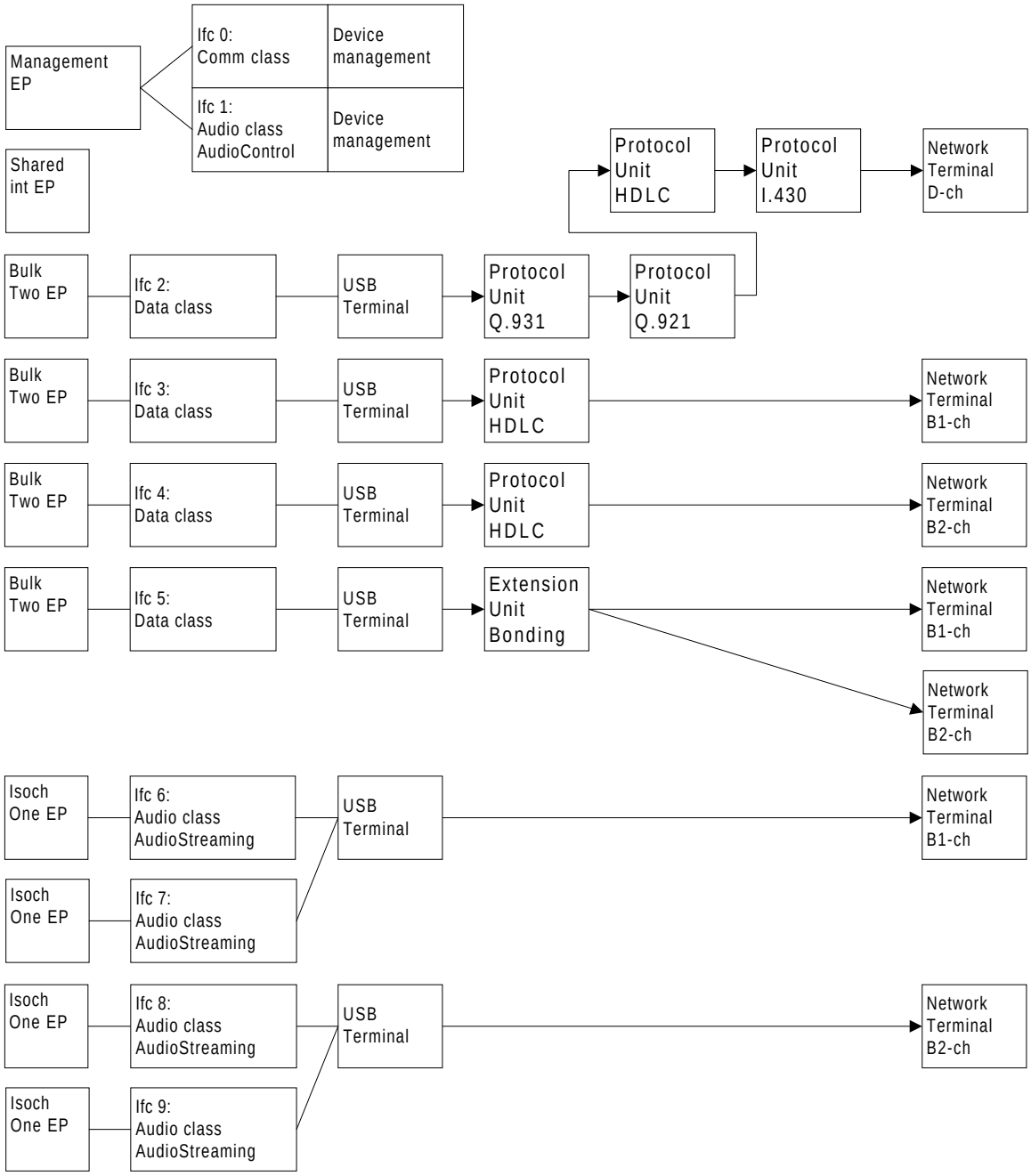
For each interface there exists a number of alternate settings. By incorporating an alternate setting with *bNumEndpoints* = 00h for each interface involved in data transfer, a device offers to the host the option to temporarily relinquish USB bandwidth. If such setting is implemented, it must be as a default alternate setting (alternate setting zero).

Appendix D: Multi-Channel Implementation Examples

D.1 ISDN BRI T/A with two POTS interfaces



D.2 ISDN BRI T/A with vendor specific protocol (Bonding)



D.3 Passive ISDN Solutions

The multi-channel model allows for different device implementations depending on the system requirements. Many of the ISDN devices described in this section utilize capabilities such as synchronized isochronous data transfers and shared endpoints as defined in the Universal Serial Bus Common Class Specification. Devices may also support a subset of the interfaces and alternate settings shown sections D.1 and D.2. This allows the realization of device designs

that are characterized by fewer requirements on the device's ISDN and USB portions, thus reducing the number and complexity of required components (e.g. number of interfaces and endpoints). This is necessary as an ISDN terminal may support only pure data applications (data class), pure voice applications (audio class) or both. Such designs use isochronous transfer instead of bulk in order to minimize buffer size, and locate the D-channel protocol stack in the host.

For example a USB ISDN device that follows the ISDN model provides multiplexing and framing functions for the D- and B-channels on the ISDN BRI physical interface. The channel data is transferred frame-interpreted (HDLC frames) for the D-channel and unmodified (transparent data) or optionally frame-interpreted for the B-channels (B1 and B2).

In the examples provided (Figure 9 through Figure 11), the D-channel control and data frame transfers are realized by using a data class interface. In order to fulfill the strict ISDN timing requirements for PTT approval, data is transferred isochronously via shared endpoints (as in Figure 9 and Figure 10) together with B-channel data. Alternatively, it is handled via a separate endpoint pair (as in Figure 11). Due to the timing requirements for D-channel collision resolution on the S-bus it is required to have the HDLC formatting done at the USB device, i.e. transparent framing cannot be used.

The examples further illustrate the different potential methods for handling the B-channels. Each B-channel utilizes a pair of Audio Class interfaces (isochronous pipes) when transferring voice data (as in Figure 11), and a Data Class interface when transferring raw un-interpreted frame data (Figure 10). The optional implementation of frame interpretation is not needed for the B-channel if the host provides the means for HDLC data encoding/decoding and error handling. Depending on the system requirements, another implementation may use an Audio Class and Data Class interface for the B-channels (as in Figure 9). Shared or separate endpoints can also be used for the B-channels.

Sync Information for channel B1 or channel B2 can be handled via an isochronous endpoint, or for B1 and B2 together via a shared isochronous endpoint. This endpoint can be omitted if no explicit sync info is required, e.g. when implicit feed forward information is contained in the IN datastream.

The Multi-Channel model allows different device implementations depending on the system requirements.

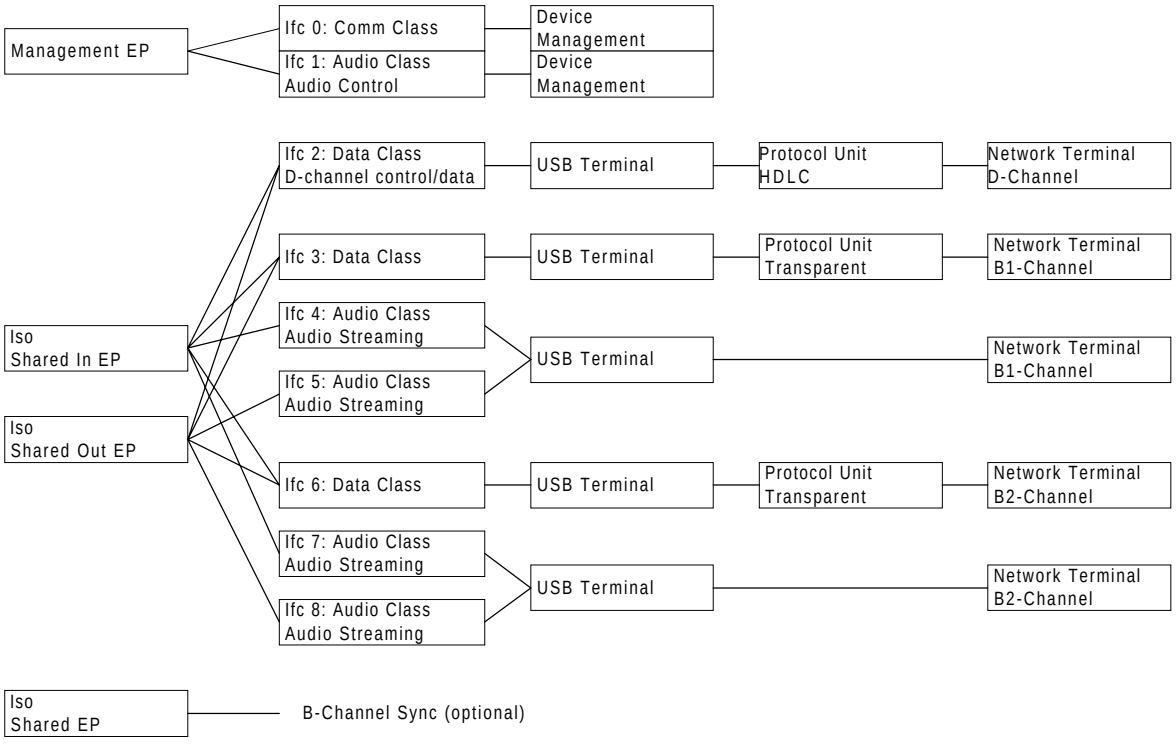


Figure 9: Passive ISDN Example 1

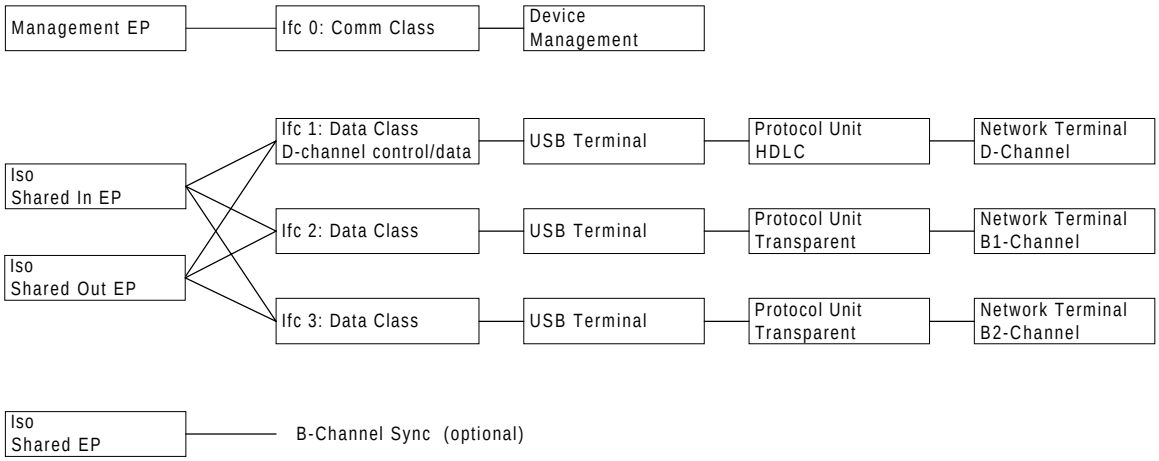


Figure 10: Passive ISDN Example 2

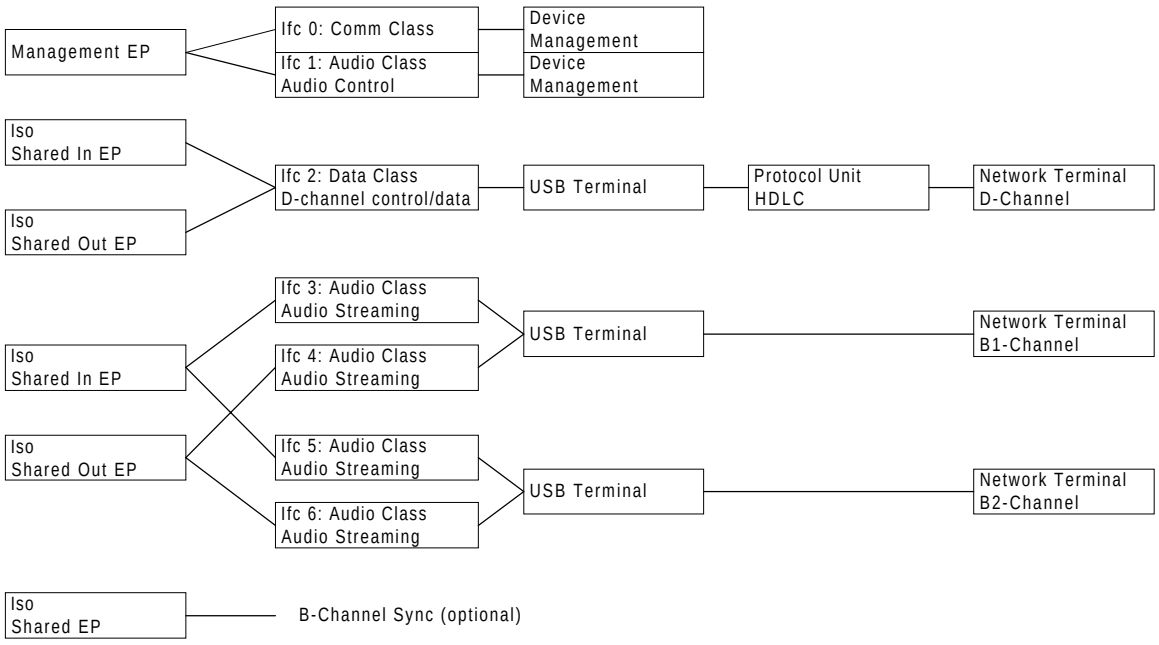


Figure 11: Passive ISDN Example 3

Appendix E: Data Class Protocol Definitions

Definitions

- A *REQuest* is a command from a higher protocol level to a lower.
- A *CONfirm* is an answer from a lower protocol level to a higher on a Request.
- An *INDication* is a command from a lower protocol level to a higher.
- A *RESponse* is an answer from a higher protocol level to a lower on an Indication.

Table 76: Command Type Encoding

Command type	Value
REQ	XXXXXX00b
CON	XXXXXX11b
IND	XXXXXX01b
RES	XXXXXX10b

E.1 Physical Interface Protocols

E.1.2 I.430: BASIC USER-NETWORK INTERFACE – LAYER 1

Protocol code: According to Table 19.

Description: This is a protocol running on an ISDN BRI device with an S0-interface. It provides de-multiplexing of two B-channels and a D-channel. The protocol covers both user and network side of a connection.

Table 77: I.430 Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmOptions*	1	Bitmap	D7..D2: RESERVED (Reset to zero) D1: D-channel transmit priority class 0 - Class 1 1 - Class 2 D0: 0 – User side 1 – Network side

Note : The parameter list is read by the protocol on activation of the Protocol Unit.

Table 78: I.430 Command Message Format

Command	Corresponding ITU I.430 Primitive	ITU I.430 Message Reference
I430_PH_DATA_REQ	PH-DATA request	2.3 Primitives between layer 1 and the other entities, Note 1
I430_PH_ACTIVATE_REQ	PH-ACTIVATE request	6.2.1.3 Activate primitives

Command	Corresponding ITU I.430 Primitive	ITU I.430 Message Reference
I430_PH_ACTIVATE_IND	PH-ACTIVATE indication	6.2.1.3 Activate primitives
I430_PH_ACTIVATE_B_REQ	N.A.	Note 2
I430_PH_DEACTIVATE_IND	PH-DEACTIVATE indication	6.2.1.4 Deactivate primitives, Note 4
I430_PH_DEACTIVATE_B_REQ	N.A.	Note 3
I430_MPH_ERROR_IND	MPH-ERROR indication	6.2.1.5 Management primitives
I430_MPH_ACTIVATE_IND	MPH-ACTIVATE indication	6.2.1.3 Activate primitives
I430_MPH_DEACTIVATE_REQ	MPH-DEACTIVATE request	6.2.1.4 Deactivate primitives
I430_MPH_DEACTIVATE_IND	MPH-DEACTIVATE indication	6.2.1.4 Deactivate primitives
I430_MPH_INFORMATION_IND	MPH-INFORMATION indication	6.2.1.5 Management primitives

Commands according to I.430 ”Table E.1 I.430 Primitives associated with layer 1”.

Note 1: PH-DATA request does not have a data field since this function is performed by other protocols such as HDLC. Therefore PH-DATA is excluded from the command list since it doesn’t have any function.

Note 2: This primitive is an USB extension to enable a specific B-channel.

Note 3: This primitive is an USB extension to disable a specific B-channel.

Note 4: This primitive will deactivate the physical layer connection (including all B-channels).

Table 79: I.430 Commands

bCommand	Value	Request	Indication	Confirm	Response
I430_PH_DATA_xxx	000000NNb	X	-	-	-
I430_PH_ACTIVATE_xxx	000001NNb	X	X	-	-
I430_PH_DEACTIVATE_xxx	000010NNb	-	X	-	-
I430_PH_ACTIVATE_B_xxx	000011NNb	X	-	-	-
I430_PH_DEACTIVATE_B_xxx	000100NNb	X	-	-	-
I430_MPH_ERROR_xxx	000101NNb	-	X	-	-
I430_MPH_ACTIVATE_xxx	000110NNb	-	X	-	-
I430_MPH_DEACTIVATE_xxx	000111NNb	X	X	-	-
I430_MPH_INFORMATION_xxx	001000NNb	-	X	-	-

NOTE 1: ‘NN’ in **Value** encoded according to Table 76

NOTE 2: X : Exists

\- : Does not exist

Table 80: I.430 Activate, Deactivate Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ACTIVATE_REQ, I430_PH_ACTIVATE_IND, I430_PH_DEACTIVATE_IND, I430_MPH_ACTIVATE_IND, I430_MPH_DEACTIVATE_REQ, I430_MPH_DEACTIVATE_IND command as defined in Table 79

Table 81: I.430 PhActivateBReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ACTIVATE_B_REQ command as defined in Table 79
1	*bChannel*	1	Number	Index of B-channel to activate 0 – RESERVED 1 – B1-channel 2 – B2-channel 3 to FFh – RESERVED

Table 82: I.430 PhDeactivateBReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_DEACTIVATE_B_REQ command as defined in Table 79
1	*bChannel*	1	Number	Index of B-channel to deactivate 0 – RESERVED 1 – B1-channel 2 – B2-channel 3 to FFh - RESERVED

Table 83: I.430 PhDataReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_DATA_REQ command as defined in Table 79
1	*bPriority*	1	Number	D-channel transmit data priority class 0 – Priority class 1 1 – Priority class 2 2 to FFh – RESERVED

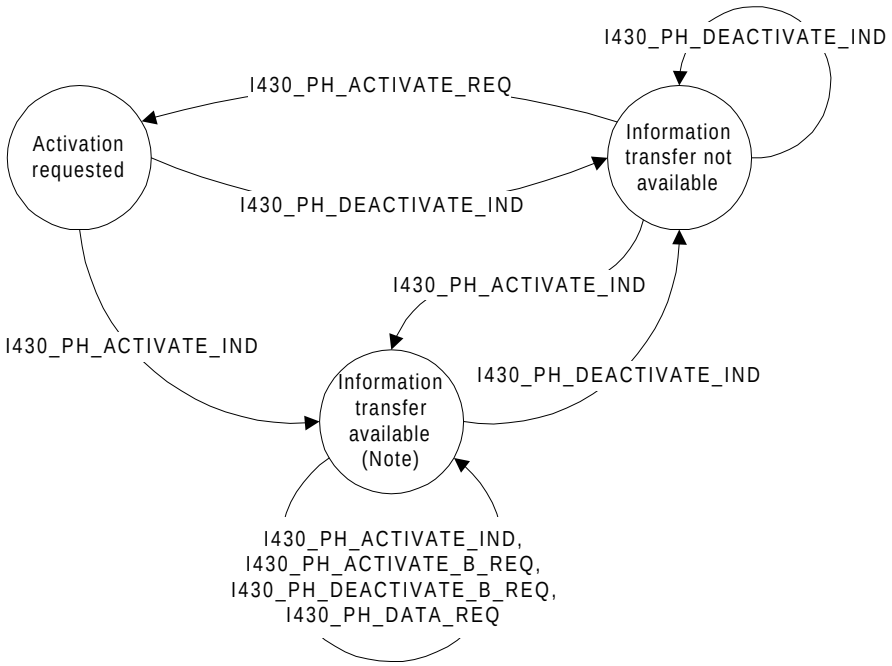
Table 84: I.430 MphErrorInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ERROR_IND command as defined in Table 79
1	*bInfo*	1	Number	Error codes 0 = Error, RX info 0 1 = Error, RX info 2 2 = Error, lost framing 3 = Recover from error, RX info 0 4 = Recover from error, RX info 2 5 = Recover from error, RX info 4 6 = FFh RESERVED

Note: See I.430 "TABLE 5/I.430", "TABLE 6/I.430", "TABLE C.1/I.430", "TABLE C.2/I.430" for more details

Table 85: I.430 MphInformationInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_INFORMATION_IND command as defined in Table 79
1	*bInfo*	1	Number	0 : Physical layer disconnected 1 : Physical layer connected 2 .. FFh: RESERVED (future use)



Note: Layer 2 is not aware if the information transfer capability is temporariliy interrupted

Figure 12: I.430 Layer 1 and Layer 2

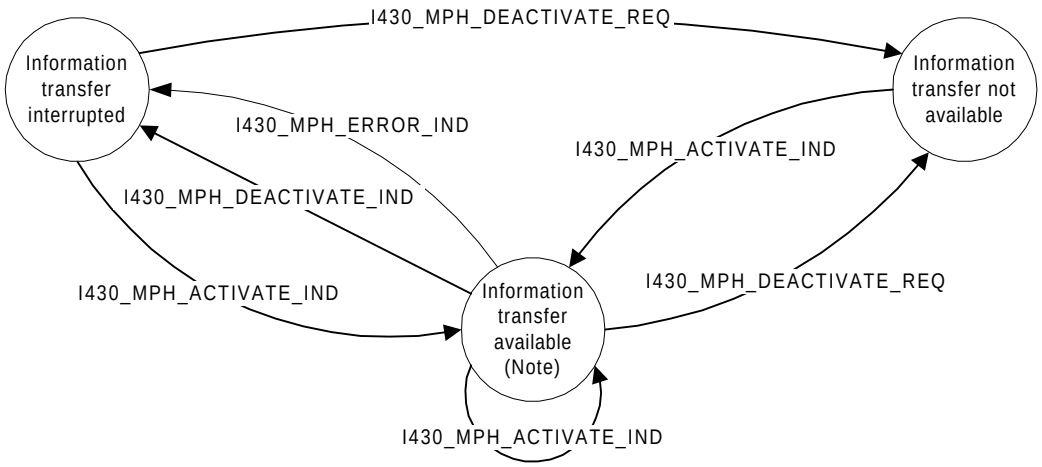


Figure 13: Layer 1 - Management Network Side

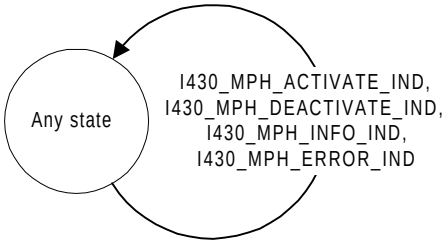


Figure 14: Layer 1 - Management User Side

E.2 Framing Protocols

E.2.1 HDLC Framing

Protocol code: According to Table 19.

Description: The HDLC framing protocol provides functions for creating and extracting HDLC frames on a serial synchronous data stream. See ISO/IEC 3309-1993 for further details.

Table 86: HDLC Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*wBufferLength*	2	Number	Max number of bytes in a buffer excluding FCS

bParameterIndex	Field	Size	Value	Description
1	*bmConfig*	2	Bitmap	D15: RESERVED (Reset to zero) D14: TX and RX data handling 0 – TX data is transmitted to the line. RX data is sent to the host. 1 - TX data is transmitted to the line and looped back to RX. RX data from the line is ignored. D13: TX frame handling after transmission 0 – Discard frame 1 – Return frame to host (for the sending protocol) D12..D11: Address filtering mode 00 – None 01 – 8 bit address 10 – 16 bit address 11 – RESERVED D10..D7: Min number of flags between frames (0 – 15) in TX direction D6: Idle 0 – Flags 1 – Mark D5: Generate FCS on TX data 0 – Generate FCS 1 – Do not generate FCS D4..D3: Check FCS on RX data 00 – Verify FCS 01 – Verify FCS and remove invalid frame 10 – Ignore FCS 11 – RESERVED D2..D1: Frame check sequence (FCS) on TX and RX data 00 – None 01 – CRC16 10 – CRC32 11 – RESERVED D0: Data encoding 0 – NRZ 1 – NRZI
2	*wAddr Comparator0*	2	Number	First address comparator. 8 bit address D7..D0: Address D15..D8: RESERVED (Reset to zero) 16 bit address D15..D0: Address
…	…	…	…	
1+N	*wAddr ComparatorN-1*	2	Number	Nth address comparator. 8 bit address D7..D0 Address D15..D8: RESERVED (Reset to zero) 16 bit address D15..D0: Address

Note: The parameter list is read by the protocol on activation of Protocol Unit.

Table 87: HDLC Commands

bCommand	Value	Request	Indication	Confirm	Response
HDLC_CONTROL_xxx	000000NNb	X	-	-	X
HDLC_STATUS_xxx	000001NNb	X	X	-	X
HDLC_DATA_xxx	000010NNb	X	X	-	-

NOTE 1: 'NN' in **Value** encoded according to Table 76
NOTE 2: X : Exists
- : Does not exist

Table 88: HDLC ControlRes, StatusReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_CONTROL_RES or HDLC_STATUS_REQ command as defined in Table 87

Note: The device should return a HDLC_STATUS_RES on reception of HDLC_STATUS_REQ

Table 89: HDLC ControlReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_CONTROL_REQ command as defined in Table 87
1	*bmControl*	1	Bitmap	D7..D3: RESERVED (Reset to zero) D2: Receiver abort 0 – No action 1 – Abort ongoing RX and abort pending RX buffers D1: FCS generation 0 – Generate correct FCS 1 – Generate bad FCS D0: Transmitter abort 0 – No action 1 – Abort ongoing TX and discard pending buffers ahead of this command

Note: The device should return a HDLC_CONTROL_RES on reception of HDLC_CONTROL_REQ

Table 90: HDLC StatusInd/Res Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_STATUS_IND or HDLC_STATUS_RES command as defined in Table 87

Offset	Field	Size	Value	Description
1	*bmStatus*	1	Bitmap	D7..D5: RESERVED (Reset to zero) D4: Source of receive data 0 – Normal RX data 1 – Return of TX data D3..D2: Frame length status 00 – OK 01 – Too short frame 10 – Too long frame 11 – Frame length is not an integer multiple of 8 bits D1: FCS status 0 – OK 1 – Error D0: Received frames discarded due to overrun 0 – No 1 – Yes

Note: The HDLC_STATUS_IND should immediately precede the HDLC_DATA_IND to which it applies. It is optional to send a HDLC_STATUS_IND if received data is without errors.

Table 91: HDLC DataReq/Ind Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_DATA_REQ or HDLC_DATA_IND command as defined in Table 87
1	*bProtocolData0*	1	Number	First byte with protocol data. FCS excluded.
…	…	…	…	
0+N	*bProtocolDataN-1*	1	Number	Nth byte with protocol data. FCS excluded.

E.2.2 Transparent framing

Protocol code: According to Table 19.

Description: This protocol provides no framing on a synchronous bitstream.

Table 92: TRANS Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmConfig*	1	Bitmap	D7..D2: RESERVED (Reset to zero) D1: TX and RX data handling 0 – TX data is transmitted to the line. RX data is sent to the host. 1 - TX data is transmitted to the line and looped back to RX. RX data from the line is ignored. D0: Data encoding 0 – NRZ 1 – NRZI
1	*bmConfigCapabilities*	1	Bitmap (read only)	D7..D1: RESERVED (Reset to zero) D0: NRZI encoding option 0 – NRZI not available 1 – NRZI available

Note : The parameter list is read by the protocol on activation of Protocol Unit.

Table 93: TRANS Commands

bCommand	Value	Request	Indication	Confirm	Response
TRANS_CONTROL_xxx	000000NNb	X	-	-	X
TRANS_STATUS_xxx	000001NNb	X	X	-	X
TRANS_DATA_xxx	000010NNb	X	X	-	-

NOTE 1: 'NN' in **Value** encoded according to Table 76
NOTE 2: X : Exists
- : Does not exist

Table 94: TRANS ControlRes, StatusReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_CONTROL_RES or TRANS_STATUS_REQ command as defined in Table 93

Note: The device should return a TRANS_STATUS_RES on reception of TRANS_STATUS_REQ

Table 95: TRANS ControlReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_CONTROL_REQ command as defined in Table 93
1	*bmControl*	1	Bitmap	D7..D3 : RESERVED (Reset to zero) D2: Transmitter data underrun option 0 – Transmit continuous idle mark whenever the transmit buffer is underrun. 1 – Repeatedly transmit the last buffer received from the host whenever the transmit buffer is underrun (for tones, etc) D1: Receiver abort 0 – No action 1 – Abort ongoing RX and abort pending RX buffers. D0: Transmitter abort 0 – No action 1 – Abort ongoing TX and discard pending buffers ahead of this one

Note: The device should return a TRANS_CONTROL_RES on reception of TRANS_CONTROL_REQ

Table 96: TRANS StatusInd/Res Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_STATUS_IND or TRANS_STATUS_RES command as defined in Table 93

Offset	Field	Size	Value	Description
1	*bmStatus*	1	Bitmap	D7 .. D2 : RESERVED (Reset to zero) D1: Transmitter underrun 0 – No transmitter underrun has occurred 1 – Transmitter underrun has occurred D0: Receive overrun 0 – No received data discarded due to overrun 1 – Received data discarded due to overrun

Note: The TRANS_STATUS_IND should immediately precede the TRANS_DATA_IND to which it applies. It is optional to send a TRANS_STATUS_IND if received data is without errors.

Table 97: TRANS DataReq/Ind Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_DATA_REQ or TRANS_DATA_IND command as defined in Table 93
1	*bProtocolData0*	1	Number	First byte with protocol data
0+N	*bProtocolDataN-1*	1	Number	Nth byte with protocol data

E.3 Data Link Protocols

E.3.1 Q.921 Management: ISDN USER-NETWORK INTERFACE DATA LINK LAYER SPECIFICATION FOR CIRCUIT MODE BEARER SERVICES

Protocol code: According to Table 19.

Description: Management procedure defined by Q.921 handles TEI negotiation and distribution of received messages. This protocol will reside below all Q.921 data link procedures, as opposed to the definition by ITU-T where the management entity resides above all data link procedures (ref FIGURE 9/Q.921). All command-names (Q921M_DL_xxx) are therefore reversed in order to maintain the definition of Request, Indication, Response and Confirm, but are in all other aspects identical to ITU-T specification. The protocol covers both user and network side of a connection.

Table 98: Q.921M Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmOptions*	1	Bitmap	D7..D1: RESERVED (Reset to zero) D0 : 0 – User side 1 – Network side
1	*bT201*	1	Number	Maximum time between retransmission of the TEI identity check message
2	*bN202*	1	Number	Maximum number of transmissions of the TEI identity request message
3	*bT202*	1	Number	Minimum time between the transmission of TEI identity request messages

bParameterIndex	Field	Size	Value	Description
4	*bmTEI*	1	Bitmap	D7 .. D1: Non-automatic TEI value D0: 0 – Use non-automatic TEI value 1 – Use automatic TEI value

Note 1: Parameters 1 – 3 according to Q.921 ”5.9 List of system parameters ”
Note 2: The parameter list is read by the protocol on activation of Protocol Unit.

Table 99: Q.921M Command Message Format

Command	Corresponding ITU Q.921 data link layer primitive	ITU Q.921 message reference
Q921M_DL_ASSIGN_REQ	MDL Assign indication	4.1.1.5 MDL-ASSIGN
Q921M_DL_ASSIGN_IND	MDL Assign request	4.1.1.5 MDL-ASSIGN
Q921M_DL_REMOVE_IND	MDL Remove request	4.1.1.6 MDL-REMOVE
Q921M_DL_ERROR_REQ	MDL Error indication	4.1.1.7 MDL-ERROR
Q921M_DL_ERROR_CON	MDL Error response	4.1.1.7 MDL-ERROR
Q921M_DL_DATA_REQ	Data request	4.1.1.3 DL-DATA
Q921M_DL_DATA_IND	Data request	4.1.1.3 DL-DATA
Q921M_DL_UNIT_DATA_REQ	UData request	4.1.1.4 DL-UNIT-DATA
Q921M_DL_UNIT_DATA_IND	UData request	4.1.1.4 DL-UNIT-DATA

Note: Commands according to Q.921 ”TABLE 6/Q.921 Primitives associated with this recommendation”

Table 100: Q.921M Commands

Command	Value	Request	Indication	Response	Confirm
Q921M_DL_ASSIGN_xxx	000000NNb	X	X	-	-
Q921M_DL_REMOVE_xxx	000001NNb	-	X	-	-
Q921M_DL_ERROR_xxx	000010NNb	X	-	-	X
Q921M_DL_DATA_xxx	000011NNb	X	X	-	-
Q921M_DL_UNIT_DATA_xxx	000100NNb	X	X	-	-

NOTE 1: ‘NN’ in Value encoded according to Table 76
NOTE 2: X : Exists
- : Does not exist

The Q921M_DL_DATA_xxx and Q921M_DL_UNIT_DATA_xxx follow the message structure according to Table 107.

Table 101: Q.921M DlAssignReq wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ASSIGN _REQ command as defined Table 100

Table 102: Q.921M DlAssignInd, DlRemoveInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ASSIGN _IND, Q921M_DL_REMOVE _IND command as defined Table 100
1	*bTei*	1	Number	TEI value

Table 103: Q.921M DlErrorReq, DlErrorCon Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ERROR_IND, Q921M_DL_ERROR_CON command as defined in Table 100
1	*bError*	1	Number	Error code according to Q.921 "TABLE II.1/Q.921 Management Entity Actions for MDL-Error-Indications" where A=1, B=2, …

E.3.2 Q.921: ISDN USER-NETWORK INTERFACE DATA LINK LAYER SPECIFICATION FOR CIRCUIT MODE BEARER SERVICES

Protocol code: According to Table 19.

Description: Q.921 is the link access procedure used by Q.931.. The protocol covers both user and network side of a connection.

Table 104: Q.921 Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmOptions*	1	Bitmap	D7..D1 RESERVED (Reset to zero) D0 0 – User side 1 – Network side
1	*bT200*	1	Number	Maximum time until an acknowledgment. must be received after the transmission of an I-frame
2	*bN200*	1	Number	Maximum number of retransmissions of a frame
3	*bN201*	1	Number	Maximum number of bytes in an information field
4	*bK*	1	Number	Maximum number of outstanding I-frames
5	*bT203*	1	Number	Maximum time allowed without frames being exchanged
6	*bSAPI*	1	Number	SAPI value according Table 2 of Q.921

Note 1: Parameters at offset 1 – 5 according to Q.921 "5.9 List of system parameters ”
Note 2 : The parameter list is read by the protocol on activation of Protocol Unit

Table 105: Command Message Format

Command	Corresponding ITU Q.921 data link layer primitive	ITU Q.921 message reference
Q921_DL_ESTABLISH_REQ	Establish request	4.1.1.1 DL-ESTABLISH
Q921_DL_ESTABLISH_IND	Establish indication	4.1.1.1 DL-ESTABLISH
Q921_DL_ESTABLISH_CON	Establish confirm	4.1.1.1 DL-ESTABLISH
Q921_DL_RELEASE_REQ	Release request	4.1.1.2 DL-RELEASE
Q921_DL_RELEASE_IND	Release indication	4.1.1.2 DL-RELEASE
Q921_DL_RELEASE_CON	Release confirm	4.1.1.2 DL-RELEASE
Q921_DL_DATA_REQ	Data request	4.1.1.3 DL-DATA
Q921_DL_DATA_IND	Data indication	4.1.1.3 DL-DATA
Q921_DL_UNIT DATA_REQ	Udata request	4.1.1.4 DL-UNIT DATA
Q921_DL_UNIT DATA_IND	Udata indication	4.1.1.4 DL-UNIT DATA

Note: Commands according to Q.921 "TABLE 6of Q.921 Primitives associated with this recommendation"

Table 106: Q.921 commands

Command	Value	Request	Indication	Response	Confirm
Q921_DL_ESTABLISH_xx	000000NNb	X	X	-	X
Q921_DL_RELEASE_xxx	000001NNb	X	X	-	X
Q921_DL_DATA_xxx	000010NNb	X	X	-	-
Q921_DL_UNIT DATA_xxx	000011NNb	X	X	-	-

NOTE 1: 'NN' in Value encoded according to Table 76
NOTE 2: X : Exists
- : Does not exist

Table 107: Q.921 General Message Structure

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Command according to Table 100 and Table 106
1	*bMessageData0*	1	Parameter	First byte with optional parameter data associated with the message
0+N	*bMessageDataN-1*	1	Parameter	Nth byte with optional parameter data associated with the message

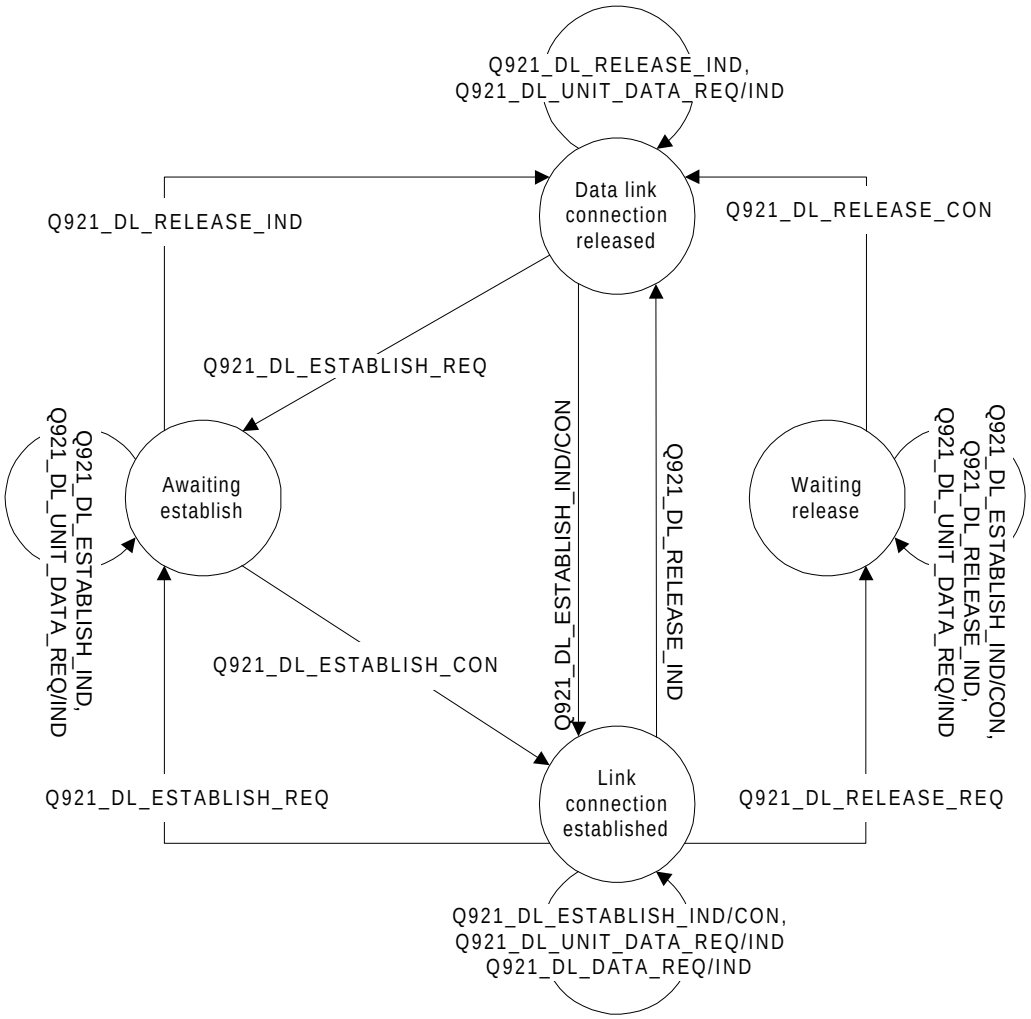


Figure 15: State Transition Diagram for Q.921

E.3.3 Q.921 TEI-multiplexor: TERMINAL ENDPOINT IDENTIFIER MULTIPLEXOR FOR ISDN USER-NETWORK INTERFACE DATA LINK LAYER

Protocol code: According to Table 19.

Description: TEI-multiplexor protocol is used to connect multiple instances of Q.921 to a D-channel in a device. The TEI-mux protocol distributes incoming messages to all connected Q.921 protocols for sending outgoing messages down the stack to the D-channel. This protocol has no configurable parameters, and it covers both user and network side of a connection.

E.4 Network layer Protocols

E.4.1 Q.931/Euro-ISDN User Side

Protocol code: According to Table 19.

Description: Call control protocol of the Q.931/Euro-ISDN userside is the ISDN connection. The protocol implements the interface described by *Primitives to/from call control* in Q.931 Annex A “User side and network side SDL diagrams”.

NOTE: Extensions for symmetric call operation are not supported.

Most commands use a general command structure as defined in Table 112 that corresponds to the message format defined in Q.931 chapter 4.1. The ones that don’t are explicitly defined within this document.

Table 108: Q.931/Euro-ISDN Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bT301*	1	Number	See Note 1
1	*bT302*	1	Number	See Note 1
2	*bT303*	1	Number	See Note 1
3	*bT304*	1	Number	See Note 1
4	*bT305*	1	Number	See Note 1
5	*bT308*	1	Number	See Note 1
6	*bT309*	1	Number	See Note 1
7	*bT310*	1	Number	See Note 1
8	*bT313*	1	Number	See Note 1
9	*bT314*	1	Number	See Note 1
10	*bT316*	1	Number	See Note 1
11	*bT317*	1	Number	See Note 1
12	*bT318*	1	Number	See Note 1
13	*bT319*	1	Number	See Note 1
14	*bT321*	1	Number	See Note 1
15	*bT322*	1	Number	See Note 1

Note 1 : Parameters according to Q.931 TABLE 9-2 "Timers in the user side”.
Note 2 : The parameter list is read by the protocol on activation of Protocol Unit.

Table 109: Q.931/Euro-ISDN Command Message Format

Command	Corresponding ITU Q.931 call control primitive	ITU Q.931 message format reference
Q931_ALERT_REQ	Alerting request	3.1.1 Alerting
Q931_ALERT_IND	Alerting indication	3.1.1 Alerting
Q931_DISC_REQ	Disc request	3.1.5 Disconnect
Q931_DISC_IND	Disc indication	3.1.5 Disconnect
Q931_ERROR_IND	Error indication	3.1.5 Disconnect /Note 1

Command	Corresponding ITU Q.931 call control primitive	ITU Q.931 message format reference
Q931_GET_STATISTICS_REQ	N.A.	N.A.
Q931_GET_STATISTICS_CON	N.A.	N.A.
Q931_INFO_REQ	Info request	3.1.6 Information
Q931_INFO_IND	Info indication	3.1.6 Information
Q931_LINK_FAIL_IND	Link fail indication	3.1.5 Disconnect /Note 1
Q931_MORE_REQ	More info request	3.1.15 Setup acknowledge
Q931_MORE_IND	More info indication	3.1.15 Setup acknowledge
Q931_NOTIFY_REQ	Notify request	3.1.7 Notify
Q931_NOTIFY_IND	Notify indication	3.1.7 Notify
Q931_PROCEED_REQ	Proceeding request	3.1.2 Call proceeding
Q931_PROCEED_IND	Proceeding indication	3.1.2 Call proceeding
Q931_PROGRESS_REQ	Progress request	3.1.8 Progress
Q931_PROGRESS_IND	Progress indication	3.1.8 Progress
Q931_REJECT_REQ	Reject request	3.1.10 Release complete
Q931_REJECT_IND	Reject indication	3.1.10 Release complete
Q931_RELEASE_REQ	Release request	3.1.9 Release
Q931_RELEASE_IND	Release indication	3.1.9 Release
Q931_RELEASE_CON	Release confirm	3.1.9 Release
Q.931_RESTART_REQ	Management restart request	3.4.1 Restart
Q.931_RESTART_CON	Management restart acknowledge	3.4.2 Restart acknowledge
Q931_RESUME_REQ	Resume request	3.1.11 Resume
Q931_RESUME_CON	Resume confirm (ok)	3.1.12 Resume acknowledge
Q931_RESUME_CON	Resume confirm (error)	3.1.12 Resume reject
Q931_SETUP_REQ	Setup request	3.1.14 Setup
Q931_SETUP_IND	Setup indication	3.1.14 Setup
Q931_SETUP_RES	Setup response	3.1.3 Connect
Q931_SETUP_CON	Setup confirm (ok)	3.1.3 Connect
Q931_SETUP_CON	Setup confirm (error)	3.1.5 Disconnect /Note 1
Q931_COMPLETE_IND	Setup complete indication (ok)	3.1.4 Connect acknowledge
Q931_COMPLETE_IND	Setup complete indication (error)	3.1.5 Disconnect /Note 1
Q931_STATUS_IND	Status indication	3.1.16 Status
Q931_SUSPEND_REQ	Suspend request	3.1.18 Suspend
Q931_SUSPEND_CON	Suspend confirm (ok)	3.1.19 Suspend acknowledge
Q931_SUSPEND_CON	Suspend confirm (error)	3.1.20 Suspend reject
Q931_TIMEOUT_IND	Timeout indication	3.1.5 Disconnect /Note 1
Q931_USERINFO_REQ	N.A.	3.3.13 User information
Q931_USERINFO_IND	N.A.	3.3.13 User information

Note 1: Only mandatory fields are used

Table 110: Q.931/Euro-ISDN Commands

bMessageType	Value	Request	Indication	Confirm	Response
Q931_ALERT_xxx	000000NNb	X	X	-	-
Q931_COMPLETE_xxx	000001NNb	-	X	-	-
Q931_DISC_xxx	000010NNb	X	X	-	-
Q931_ERROR_xxx	000011NNb	-	X	-	-
Q931_INFO_xxx	000100NNb	X	X	-	-
Q931_LINK_FAIL_xxx	000101NNb	-	X	-	-
Q931_MORE_xxx	000110NNb	X	X	-	-
Q931_NOTIFY_xxx	000111NNb	X	X	-	-
Q931_PROCEED_xxx	001000NNb	X	X	-	-
Q931_PROGRESS_xxx	001001NNb	X	X	-	-
Q931_REJECT_xxx	001010NNb	X	X	-	-
Q931_RELEASE_xxx	001011NNb	X	X	X	-
Q931_RESUME_xxx	001100NNb	X	-	X	-
Q931_SETUP_xxx	001101NNb	X	X	X	X
Q931_SUSPEND_xxx	001110NNb	X	-	X	-
Q931_STATUS_xxx	001111NNb	-	X	-	-
Q931_TIMEOUT_xxx	010000NNb	-	X	-	-
Q931_USERINFO_xxx	010001NNb	X	X	-	-

Note 1: 'NN' in Value encoded according to Table 76
Note 2: X : Exists
- : Does not exist

Table 111: Q.931/Euro-ISDN System Management Commands

bCommand	Value	Request	Indication	Confirm	Response
Q931_RESTART_xxx	100000NNb	X	-	X	-

Note 1: 'NN' in **Value** encoded according to Table 76
Note 2: X : Exists
- : Does not exist

Table 112: Q.931/Euro-ISDN General Command Structure

Offset	Field	Size	Value	Reference
0	*bCommand*	1	Number	Command according to Table 110
1	*iCallReference*	2 to N	Info element	Q.931 Chapter 4.3
1+N	*ilnformationElement*	Size of info element	Info element	Optional information element according to command.

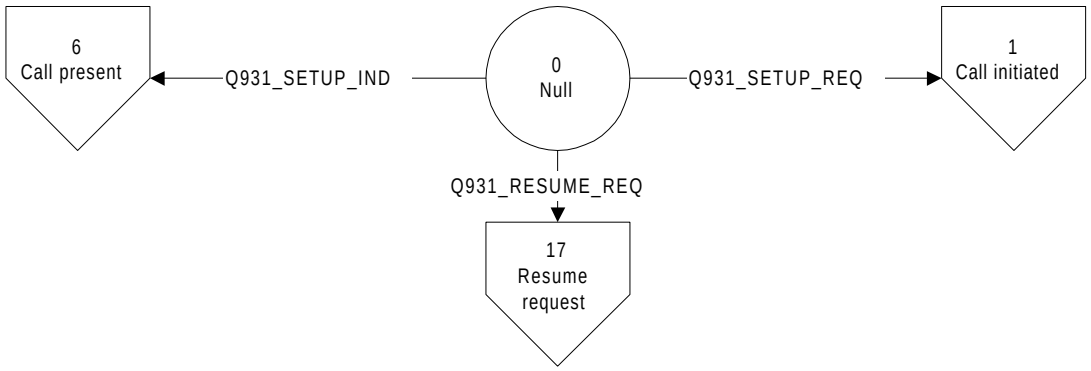


Figure 16: Q.931 Handling of Null State

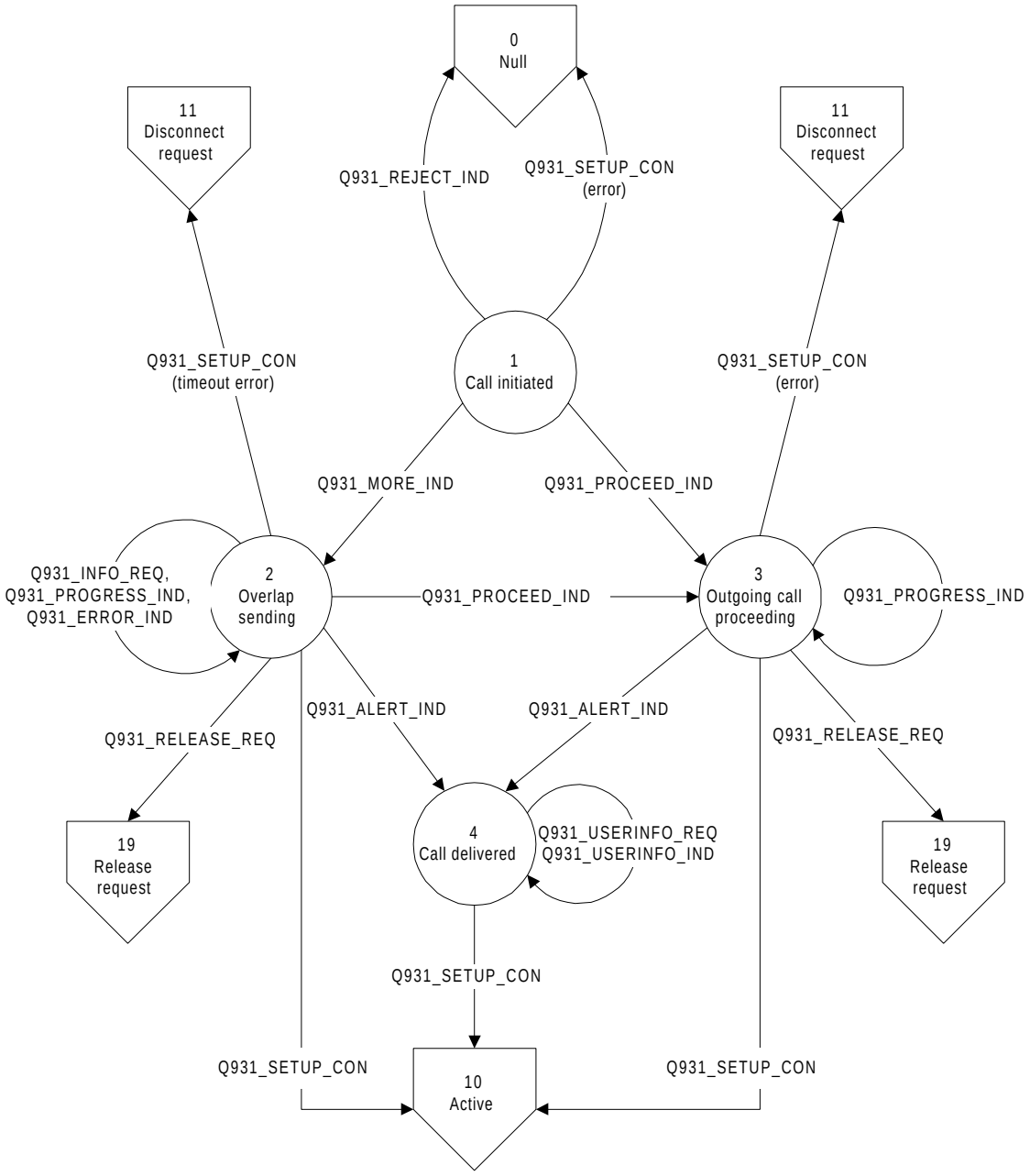


Figure 17: Q.931 Handling of Outgoing-Call States

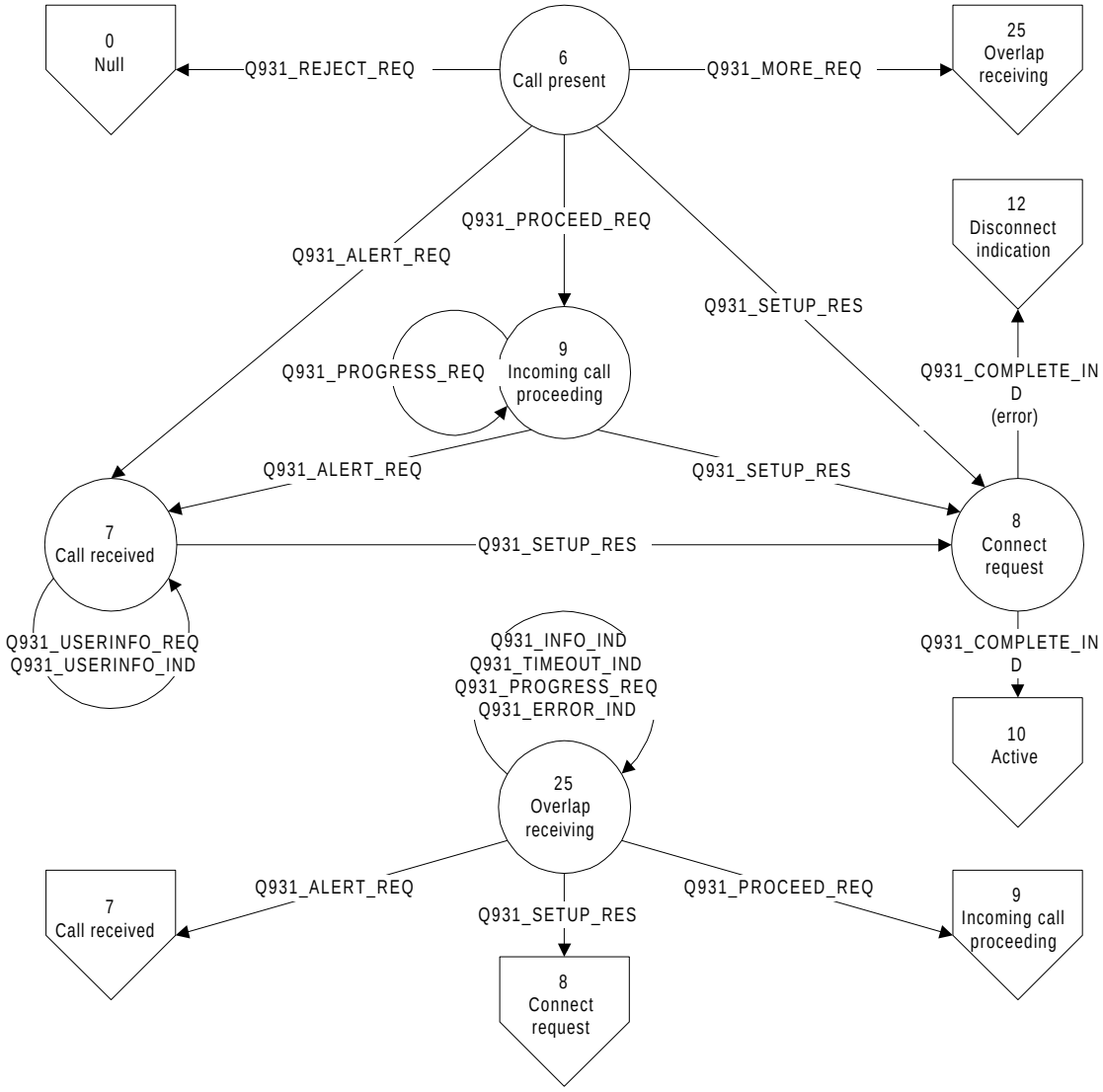


Figure 18: Q.931 Handling of Incoming Call-Setup States

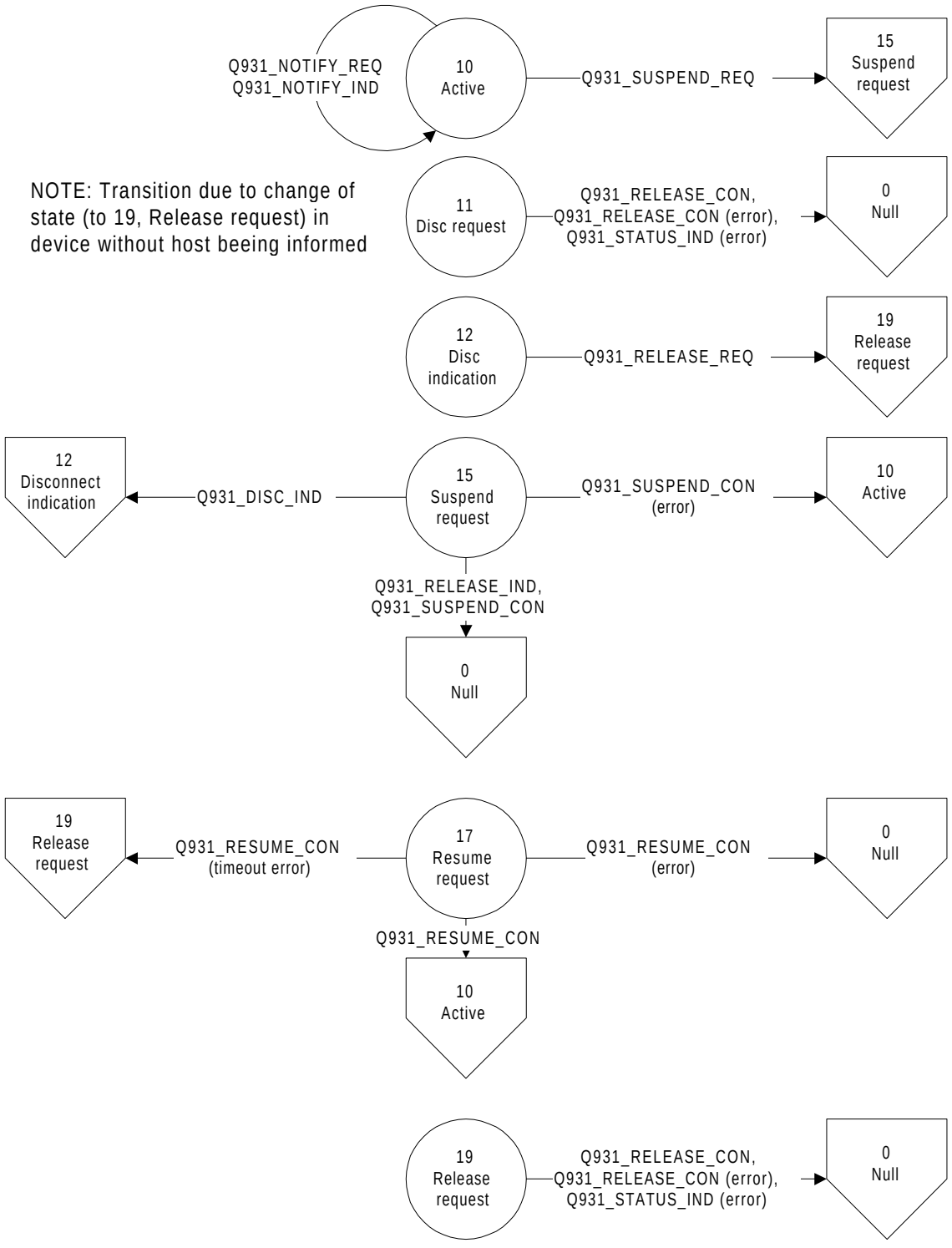


Figure 19: Q.931 Handling of Specific States

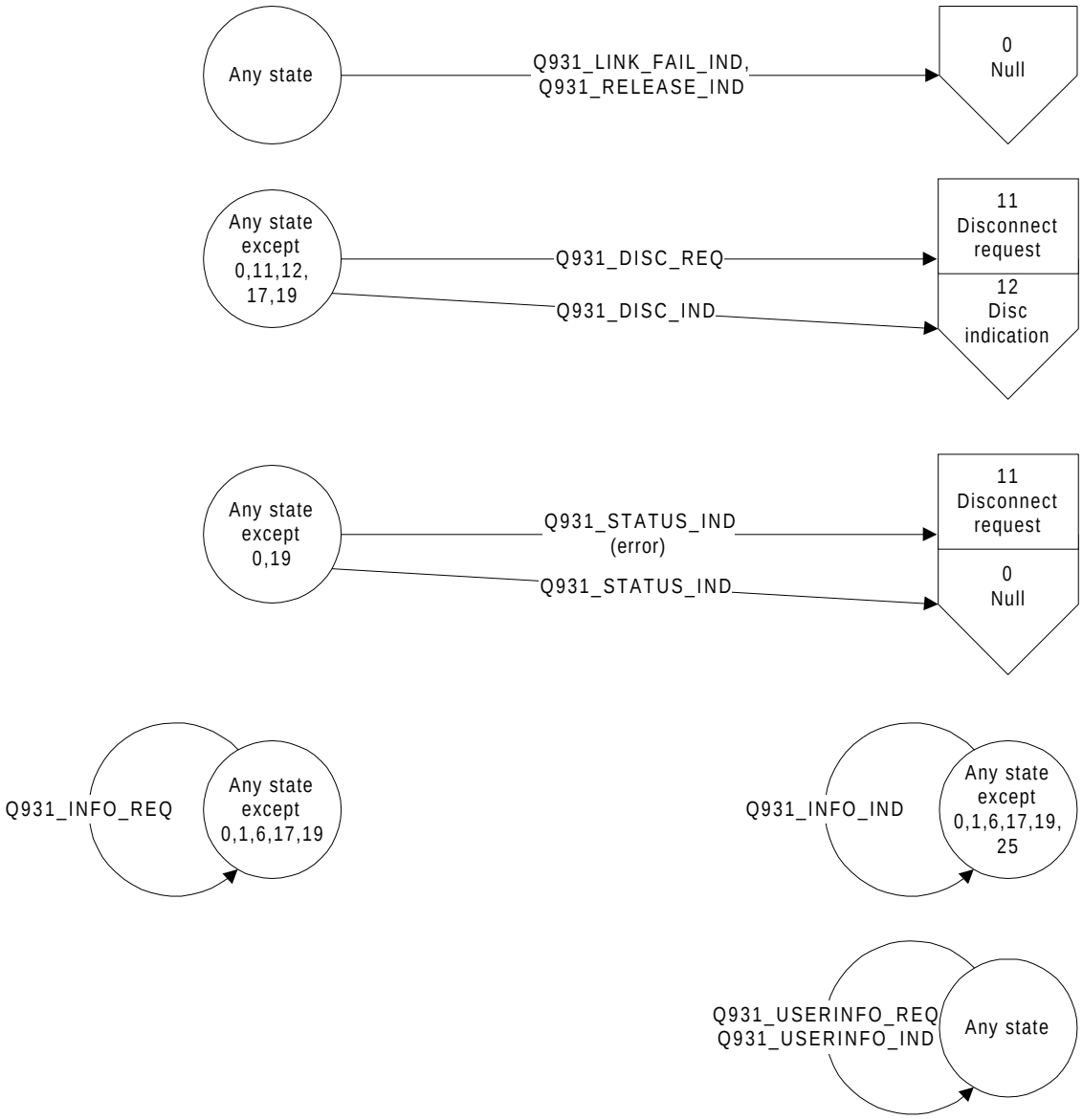


Figure 20: Q.931 Handling of Generic States

E.4.2 V.42*bis*: Data compression procedures for DCE using error correction procedures

Protocol code: According to Table 19.

Description: V.42*bis* is a data compression protocol

Table 113: V.42*bis* Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bP0*	1	Number	V.42bis data compression request

bParameterIndex	Field	Size	Value	Description
1	*wP1*	2	Number	Number of code words
2	*bP2*	1	Number	Maximum string size

Note 1: Parameters according to V.42*bis* ”10 Parameters”
Note 2: The parameter list is read by the protocol on activation of Protocol Unit

E.4.3 V.120: V.24 rate adaptation to ISDN

Protocol code: According to Table 19.

Description: V.120 is a Rate adaptation protocol. The protocol has no configurable parameters