Universal Serial Bus Class Definitions for Communication Devices

Version 1.0
May 8, 1998

Scope of this Revision

This version 1.0 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
0.8 – 0.8h	December 11, 1996 — May 20, 1997	0.8 releases.
0.9a- 0.9f	May 26, 1997 — October 10, 1997	0.9 releases.
1.0RCb	October 22, 1997	RR53, 76, 77. Accepted changes from the technical editor.
1.0RCc	October 29, 1997	Verified Xref, re-ordered requests
1.0RC1	November 9, 1997	Approved by CDC – Posted to Web
1.0RC2	December 18, 1997	RR79-84, Approved by CDC – Posted to Web
1.0RC3	January 28, 1998	RR85-90, Approved by CDC. Atlanta, GA
1.0RC4	March 19, 1998	Addition of Device Class code.
1.0RC5	April 2, 1998	Minor syntax changes, approved by CDC
1.0	May 7, 1998	Approved by DWG 1.0

Contributors

Shelagh Callahan	Intel Corporation
Paul Chehowski	Mitel Corporation
Joe Decuir	Microsoft Corporation
Ed Endejan	3Com Corporation
Randy Fehr	Northern Telecom
John Howard	Intel Corporation
Dan Moore	Diamond Multimedia Systems
Terry Moore	Moore Computer Consultants Inc.
Andy Nicholson	Microsoft Corporation
Nathan Peacock	Northern Telecom
Dave Perry	Mitel Corporation
Kenny Richards	3Com Corporation
Mats Webjörn	Universal Access
Jim Wilson	U.S. Robotics

USB Class Definitions for Communication Devices
Copyright © 1996-1998 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 electronic mail to usbdevice@mailbag.intel.com

Contents

List of Tables

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 property 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. However, other uses for devices or interfaces of this class are also valid. The specification currently outlines the following types of devices: analog modems, digital telephones, and analog telephones. Additional models for multi-channel devices, such as ISDN or ADSL, are being developed and will become available as soon as they are ready. Additionally, support for non-telecommunication devices, specifically the “serial dongle” type of device, is being addressed. This support will be made available with the multi-channel models.

This specification does not attempt to redefine existing standards for connection and control of communication services. The Communication Interface 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 (also referred to as the *USB Specification*). 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**

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

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

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.
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.
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.
DSVD	Digital Simultaneous Voice and Data, signaling method mixes data and digitized voice.
DTE	Data Terminal Equipment; for example, a PC.
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.
ISDN	Integrated Services Digital Network.
ITU	International Telecommunications Union (formerly CCITT).
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.
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.
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.
TA	Terminal Adapter, which is the equivalent of a modem for ISDN.
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.
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.

2. Management Overview

Several types of communication devices can benefit from the USB. This specification provides models for telecommunication devices, such as telephones and analog modems. 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, a digital 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 optional 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
- Call 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 that are 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 the *USB Specification*, Sections 9.1 through 9.4. 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 5.2.1.8, “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 Human Interface. To allow the appropriate class driver to manage that data, the host is presented with an interface(s), which obeys the specification of that class. The interface that is 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.

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 are 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 or endpoint 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 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). Currently, there is no Control Model to represent this situation.

- 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 3.5.1.1 “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 3.5.1.2, “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 (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 3.5.1.2, “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 same time as the definition of 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 unknown, 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 are needed. Both data wrapper forms impose no other restrictions on the data format, other than the general requirement of it being byte data and the source protocol ID of 00h being reserved for the source protocol ID.

Note: Use of a Protocol Data Wrapper on a 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	*bmOption*	2	Bitmap	D15..D0: Size of wrapper in bytes
2	*bDstProtocol*	1	Protocol	Destination protocol ID.
3	*bSrcProtocol*	1	Protocol	Source protocol ID.
4 to n-1	*bData*	n-4	Number	Data bytes

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 8. 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.

3.5.1 USB POTS Modem 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.

3.5.1.1 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.

3.5.1.1.1 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.

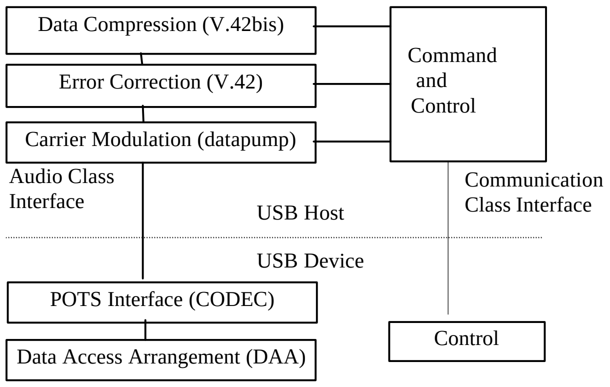
Data Compression (V.42bis)
Error Correction (V.42)
Carrier Modulation (datapump)
Command and Control
Audio Class Interface
USB Host
Communication Class Interface
USB Device
POTS Interface (CODEC)
Data Access Arrangement (DAA)
Control

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
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 would 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.

3.5.1.1.2 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 would be used.

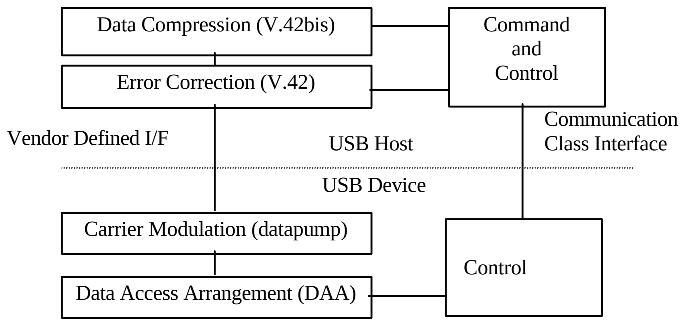
Data Compression (V.42bis)
Error Correction (V.42)
Command and Control
Communication Class Interface
Vendor Defined I/F
USB Host
USB Device
Carrier Modulation (datapump)
Data Access Arrangement (DAA)
Control

Figure 2: Datapump Model

3.5.1.2 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.

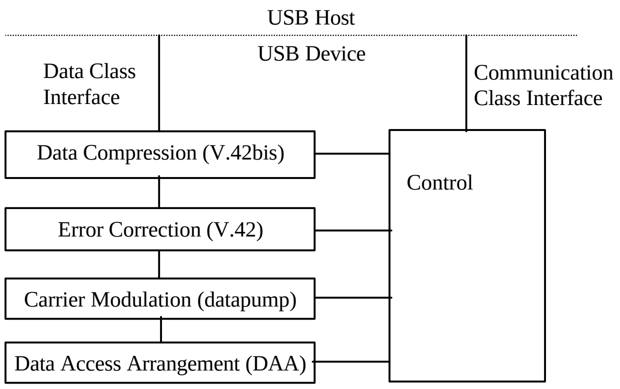
USB Host
USB Device
Data Class Interface
Communication Class Interface
Data Compression (V.42bis)
Error Correction (V.42)
Carrier Modulation (datapump)
Data Access Arrangement (DAA)
Control

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.

3.5.1.2.1 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 5.2.1.2) 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.

3.5.2 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.

3.5.2.1 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 5.2.1, "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.

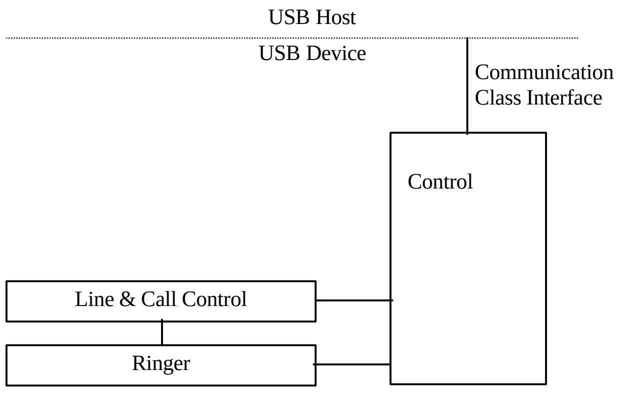

Figure 4: Telephone Control Model

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
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*

Request	**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.

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 *USB Specification*, Section 9.6.1 and 9.6.3.

4.1 Communication Device Class Code

The following table defines the Communication Device Class code:

Table 8: 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 9: 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 10: Communication Interface Class SubClass Codes

Code	**SubClass**
00h	RESERVED
01h	Direct Line Control Model
02h	Abstract Control Model
03h	Telephone Control Model
04h-7Fh	RESERVED (future use)
80h-FEh	RESERVED (vendor-specific)

The Datapump Model, as described in Section 3.5.1.1.2, “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.

Table 11: Communication Interface Class Control Protocol Codes

Protocol code	**Reference document**	**Description**
00h	*USB Specification*	RESERVED
01h	V.25ter	Common AT commands (also known as “Hayes™ compatible”)
02h-FEh		RESERVED (future use)
FFh	*USB Specification*	RESERVED (vendor-specific)

4.5 Data Interface Class Codes

The following table defines the Data Interface Class code:

Table 12: Data Interface Class Code

Code	**Class**
0Ah	Data Interface Class

This specification does not address the format of the data transferred over the data interface; it is expected that this interface will be implemented via the asynchronous or synchronous data modes, over bulk or isochronous pipes, respectively. This specification places no other requirements on the Interface descriptor.

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 13: Communication Device Class Descriptor Requirements

Offset	Field	Size	Value	Description
4	*bDeviceClass*	1	02h	Communication Device Class code as defined in Table 8.
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 Section 9.6.2, “Configuration” of the *USB Specification*.

5.1.3 Interface Descriptors

The Communication Interface Class uses the standard Interface descriptor as defined in Section 9.6.3, “Interface” 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 14: Communication Class Interface Descriptor Requirements

Offset	Field	Size	Value	Description
5	*bInterfaceClass*	1	Class	Communication Interface Class code, as defined in Table 9.
6	*bInterfaceSubClass*	1	SubClass	Communication Interface Class SubClass code, as defined in Table 10.
7	*bInterfaceProtocol*	1	Protocol	Communication Interface Class Protocol code, which applies to the subclass, as specified in the previous field, is defined in Table 11.

The Data Interface Class also uses the standard Interface descriptor as defined in Section 9.6.3, “Interface” 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 15: Data Class Interface Descriptor Requirements

Offset	Field	Size	Value	Description
5	*bInterfaceClass*	1	Class	Data Interface Class code, as defined in Table 12.
6	*bInterfaceSubClass*	1	00h	Data Interface Class SubClass code, unused at this time.
7	*bInterfaceProtocol*	1	00h	Data Interface Class Protocol code, unused at this time.

5.1.4 Endpoint Descriptors

The Communication Interface Class and Data Interface Class use the standard Endpoint descriptor, as defined in Section 9.6.4, “Endpoint” 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 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.

Table 16: Functional Descriptor General Format

Offset	**Field**	**Size**	**Value**	**Description**
0	*bFunctionLength*	1	Number	Size of this descriptor, in bytes (*n*).
1	*bDescriptorType*	1	Constant	CS_INTERFACE, as defined in Table 17.
2	*bDescriptorSubtype*	1	Constant	Identifier (ID) of functional descriptor. For a list of the supported values, see Table 18.
3 to *n-1*	*(function specific data)*	*n*-3	Misc.	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 the *USB Specification* in Table 9-4 and by setting the class-specific bit defined within the Common Class Specification to generate corresponding class-specific constants.

Table 17: Type Values for the bDescriptor Field

Descriptor type	**Value**
CS_INTERFACE	24h
CS_ENDPOINT	25h

Table 18: 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-FFh	N/A	N/A	RESERVED (future use)

5.2.1.1 Header Functional Descriptor

The class-specific descriptor shall start with a header that is defined in Table 16. 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 19: 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 18.
3	*bcdCDC*	2	Number	*USB Class Definitions for Communication Devices Specification* release number in binary-coded decimal.

5.2.1.2 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 20: 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 18.
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*)

5.2.1.3 Abstract Control Management Functional Descriptor

The Abstract Control Management functional descriptor describes the commands supported by the Communication Class interface, as defined in Section 3.5.1.2, with the SubClass code of Abstract Control. It can only occur within the class-specific portion of an Interface descriptor.

Table 21: 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 18.
3	*bmCapabilitie*s	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.

5.2.1.4 Direct Line Management Functional Descriptor

The Direct Line Management functional descriptor describes the commands supported by the Communication Class interface, as defined in Section 3.5.1.1, with the SubClass code of Direct Line. It can only occur within the class-specific portion of an Interface descriptor.

Table 22: 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 18.
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.

5.2.1.5 Telephone Ringer Functional Descriptor

The Telephone Ringer functional descriptor describes the ringer capabilities supported by the Communication Class interface, as defined in Section 3.5.2.1, 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 23: 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 18.
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). 2: 2 discrete volume steps (i.e., 0 and 255). 3: 3 discrete volume steps.
4	*bNumRingerPatterns*	1	Number	Number of ringer patterns supported, values of 1 to 255 with a value of 0 being reserved for future use.

5.2.1.6 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 3.5.2.1, 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 37 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 24: 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 18.
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.

5.2.1.7 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 3.5.2.1, 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 25: 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 18.
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.

5.2.1.8 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 26: 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
2	*bDescriptorSubtype*	1	Constant	Union functional descriptor SubType as defined in Table 18.
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*)

5.2.1.9 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 27: 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 18.
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.

5.3 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.

5.3.1 Class-Specific Configuration Descriptor

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

5.3.2 Class-Specific Interface Descriptor

The Communication Class uses a class-specific descriptor at the Interface level, which can handle one or more functional descriptors. The Data Class does not currently use any class-specific descriptor information at the Interface level.

Table 28: Sample 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	01h	Header. This is defined in Table 18, which defines this as a header.
3	*bcdCDC*	2	0100h	*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 18.
8	*bmCapabilities*	1	0Fh	This field contains the value 0Fh, because the device supports all the corresponding commands for the Abstract Control Model interface.

* 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 Section 9.4, “Standard Device Requests” 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 29: Class-Specific Requests

bmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**	**Data**
00100001B 00100010B	SEND_ENCAPSULATED _COMMAND	Zero	Interface Endpoint	Amount of data, in bytes, associated with this recipient.	Control protocol-based command
10100001B 10100010B	GET_ENCAPSULATED _RESPONSE	Zero	Interface Endpoint	Amount of data, in bytes, associated with this recipient.	Protocol-dependent data response
00100001B 00100010B	SET_COMM_FEATURE	Feature Selector	Interface Endpoint	Length of State Data	State
10100001B 10100010B	GET_COMM_FEATURE	Feature Selector	Interface Endpoint	Length of Status Data	Status
00100001B 00100010B	CLEAR_COMM_ FEATURE	Feature Selector	Interface Endpoint	Zero	None
00100001B 00100010B	SET_AUX_LINE_STATE	0 –Disconnect 1 – Connect	Interface Endpoint	Zero	None
00100001B 00100010B	SET_HOOK_STATE	Relay Config.	Interface Endpoint	Zero	None
00100001B 00100010B	PULSE_SETUP	Enable/ Disable	Interface Endpoint	Zero	None

bmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**	**Data**
00100001B 00100010B	SEND_PULSE	Cycles	Interface Endpoint	Zero	None
00100001B 00100010B	SET_PULSE_TIME	Timing	Interface Endpoint	Zero	None
00100001B 00100010B	RING_AUX_JACK	Number of Rings	Interface Endpoint	Zero	None
00100001B 00100010B	SET_LINE_CODING	Zero	Interface Endpoint	Size of properties	Line Coding Structure
10100001B 10100010B	GET_LINE_CODING	Zero	Interface Endpoint	Size of Structure	Line Coding Structure
00100001B 00100010B	SET_CONTROL_LINE_STATE	Control Signal Bitmap	Interface Endpoint	Zero	None
00100001B 00100010B	SEND_BREAK	Duration of Break	Interface Endpoint	Zero	None
00100001B 00100010B	SET_RINGER_PARMS	Zero	Interface Endpoint	4	Ringer Configuration bitmap
10100001B 10100010B	GET_RINGER_PARMS	Zero	Interface Endpoint	4	Ringer Configuration bitmap
00100001B 00100010B	SET_OPERATION_PARMS	Operation Mode	Interface Endpoint	Zero	None
10100001B 10100010B	GET_OPERATION_PARMS	Zero	Interface Endpoint	2	Operation mode
00100001B 00100010B	SET_LINE_PARMS	Lines State Change	Interface Endpoint	Length of Data	None/Data
10100001B 10100010B	GET_LINE_PARMS	Zero	Interface Endpoint	Size of Structure	Line Status Information structure
00100001B 00100010B	DIAL_DIGITS	Zero	Interface Endpoint	Length of Dial String	Dialing string

Table 30: 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

Request	Value
SET_LINE_PARMS	34h
GET_LINE_PARMS	35h
DIAL_DIGITS	36h
RESERVED (future use)	37h-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 00100010B	SEND_ENCAPSULATED _COMMAND	Zero	Interface Endpoint	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 10100010B	GET_ ENCAPSULATED_ RESPONSE	Zero	Interface Endpoint	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 00100010B	SET_COMM_FEATURE	Feature Selector	Interface Endpoint	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 10100010B	GET_COMM_ FEATURE	Feature Selector	Interface Endpoint	Length of Status Data	Status

Table 31 Communication Feature Selector Codes

Feature selector	Code	Targets	Length of Data	Description
RESERVED	00h	None	None	Reserved for future use
ABSTRACT_STATE	01h	Interface or Endpoint	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 or Endpoint	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 32 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 00100010B	CLEAR_COMM_ FEATURE	Feature Selector	Interface Endpoint	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 00100010B	SET_AUX_LINE_STATE	0 –Disconnect 1 - Connect	Interface Endpoint	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 00100010B	SET_HOOK_STATE	Relay Config.	Interface Endpoint	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 33: 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 00100010B	PULSE_SETUP	Enable/ Disable	Interface Endpoint	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 5.2.1.4.

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 00100010B	SEND_PULSE	Cycles	Interface Endpoint	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 00100010B	SET_PULSE_TIME	Timing	Interface Endpoint	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 00100010B	RING_AUX_JACK	Number of Rings	Interface Endpoint	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 00100010B	SET_LINE_CODING	Zero	Interface Endpoint	Size of Structure	Line Coding Structure

For the definition of valid properties, see Table 34, 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 10100010B	GET_LINE_CODING	Zero	Interface Endpoint	Size of Structure	Line Coding Structure

The line coding properties are defined in the following table:

Table 34: 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 00100010B	SET_CONTROL_LINE _STATE	Control Signal Bitmap	Interface Endpoint	Zero	None

Table 35: Control Signal Bitmap Values for SetControlLineState

Bit position	Description
D15..D2	RESERVED (Reset to zero)
D1	Carrier control. This signal corresponds to V.24 signal 105 and RS-232 signal RTS. 0 - Deactivate carrier 1 - Activate carrier
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 00100010B	SEND_BREAK	Duration of Break	Interface Endpoint	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 00100010B	SET_RINGER_PARMS	Zero	Interface Endpoint	4	Ringer Configuration bitmap

This command is sent to the interface; 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 36: 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 10100010B	GET_RINGER_PARMS	Zero	Interface Endpoint	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 36.

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 00100010B	SET_OPERATION_PARMS	Operation Mode	Interface Endpoint	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 37: 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 10100010B	GET_OPERATION_PARMS	Zero	Interface Endpoint	2	Operation mode

The returned operation mode value describes the current operational mode of the device, as specified in Table 37.

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 00100010B	SET_LINE_PARMS	Line State Change	Interface Endpoint	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 38: 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 10100010B	GET_LINE_PARMS	Zero	Interface Endpoint	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 39: 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 36.
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 40: 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 41: 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 47.)
D7..D0	Call state value. (For definitions of call state values, see Table 42.)

Table 42: 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 47.
05h	Connected. Call state additional data, D15..D8, contains extra information, as defined in Table 47.
06h	Incoming call. Call state additional data, D15..D8, contains extra information, as defined in Table 47.

6.2.22 DialDigits

This request dials the DTMF digits over the specified line.

bmRequestType	**bRequest**	**wValue**	**wIndex**	**wLength**	**Data**
00100001B 00100010B	DIAL_ DIGITS	Zero	Interface Endpoint	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 43: 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.
,	(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.

6.3 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 44: Class-Specific Notifications

bmRequestType	**bNotification**	**wValue**	**wIndex**	**wLength**	**Data**
10100001B 10100010B	NETWORK_CONNECTION	0 - Disconnect 1 - Connected	Interface Endpoint	Zero	None
10100001B 10100010B	RESPONSE_AVAILABLE	Zero	Interface Endpoint	Zero	None
10100001B 10100010B	AUX_JACK_HOOK_ STATE	0 – On hook 1 – Off hook	Interface Endpoint	Zero	None
10100001B 10100010B	RING_DETECT	Zero	Interface Endpoint	Zero	None
10100001B 10100010B	SERIAL_STATE	Zero	Interface Endpoint	2	UART State bitmap
10100001B 10100010B	CALL_STATE_CHANGE	Call Index and Call State Change Value	Interface Endpoint	Length of data	Variable-length structure containing additional information for call state change.
10100001B 10100010B	LINE_STATE_CHANGE	Value	Interface Endpoint	Length of data	Variable length Line State structure.

Table 45: 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
RESERVED (future use)	2Ah-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 10100010B	NETWORK_CONNECTION	0 - Disconnect 1 - Connected	Interface Endpoint	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 10100010B	RESPONSE_AVAILABLE	Zero	Interface Endpoint	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 10100010B	AUX_JACK_ HOOK_STATE	0 – On hook 1 – Off hook	Interface Endpoint	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 10100010B	RING_DETECT	Zero	Interface Endpoint	Zero	None

6.3.5 *SerialState*

This notification sends asynchronous notification of UART status.

bmRequestType	**bNotification**	**wValue**	**wIndex**	**wLength**	**Data**
10100001B 10100010B	SERIAL_STATE	Zero	Interface Endpoint	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 46: 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 10100010B	CALL_STATE_CHANGE	Call index and call state change value.	Interface Endpoint	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 47: 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 1 = fax machine connection 2 = data modem connection 3-254 = reserved for future use 255 = unknown connection type
05h	Incoming Call, with the following extra bytes of data (minimum of 4 extra bytes): <u>Extra data byte 0</u> - 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 <u>Extra data byte 1</u> - 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 <u>not</u> null terminated and is encoded using one character per byte. It is <u>not</u> a UNICODE string. If time is not available then a size of 0 is required as a place setter. <u>Next data byte following number</u> - Size of string (next n bytes) which contains the phone number of calling party as delivered via Caller ID. The string is <u>not</u> null terminated and is encoded using one character per byte. It is <u>not</u> a UNICODE string. If no number is available then a size of 0 is required as a place-setter. <u>Next data byte following name</u> - Size of string (next n bytes) which contains the name of the calling party as delivered via Caller ID. The string is <u>not</u> 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 an union sending the notification message.

bmRequestType	**bNotification**	**wValue**	**wIndex**	**wLength**	**Data**
10100001B 10100010B	LINE_STATE_CHANGE	Value	Interface Endpoint	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 48: 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 online.
0003h	Hook-switch has gone on hook online.

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 3.5.1.2.1, “Abstract Control Model Serial Emulation.”

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 49: 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 50: 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.
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.