# Universal Serial Bus Class Definitions for Communications Devices

Revision 1.2

December 6, 2012

## Revision History

| Revision | Issue date | Comments |
|---|---|---|
| 1.1 | December 18, 1998 | 1.1 releases |
| 1.2 | February 9, 2007 | Final editorial changes as per February 2006 CDC meeting<br>Specify reserved code in Table 13. |
| 1.2 | November 3, 2010 | Updated with information about NCM 1.0 |
| 1.2 | Aug 02, 2012 | Updated with information about MBIM 1.0 |

**Please send comments or questions to : cdc@usb.org**

## Contributors, v1.1

| | |
|---|---|
| Paul E. Berg | Moore Computer Consultants, Inc. |
| Chuck Brabenac | Intel Corporation |
| Shelagh Callahan | Intel Corporation |
| Paul Chehowski | Mitel Corporation |
| Joe Decuir | Microsoft Corporation |
| Stefan Eder | Siemens Semiconductors |
| Ed Endejan | 3Com Corporation |
| Randy Fehr | Northern Telecom |
| Diego Friedel | AVM |
| John Howard | Intel Corporation |
| Ken Lauffenburger | Efficient Networks, Inc. |
| Ron Lewis | Rockwell Semiconductors |
| 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 | Microsoft Corporation |
| Charlie Tai | Intel Corporation |
| Mats Webjörn | Universal Access |

## Contributors, V1.2

| | |
|---|---|
| Alan Berkema | Hewlett Packard |
| Alexey Orishko | ST-Ericsson |
| Brian Meads | MCCI |
| Bruce Balden | Belcarra |
| Diego Friedel | AVM |
| Jun Guo | Broadcom |
| CC Hung | Mentor Graphics |
| Dale Self | Symbian |
| Gabriel Montenegro | Microsoft |
| Greg Scaffidi | MCCI |
| Janne Rand | Nokia |
| Joe Decuir | MCCI |
| Joel Silverman | K-Micro |
| John Turner | Symbian |
| Ken Taylor | Motorola |
| Patrik Olesen | Ericsson |
| Peter FitzRandolph | MCCI |
| Morten Christiansen | Ericsson AB |
| Richard Petrie | Nokia |

| | |
|---|---|
| Saleem Mohammed | Synopsys |
| Srinivasan Malayala | Microsoft |
| Tero Soukko | Nokia |
| Vladimir Semenyuk | Smith Micro Software |

# Table of Contents

## List of Tables

# 1 Introduction

There are three classes that make up the definition for communications devices:

- Communications Device Class
- Communications Interface Class
- Data Interface Class.

The Communications Device Class is a device-level definition and is used by the host to properly identify a communications device that may present several different types of interfaces.

The Communications Interface Class defines a general-purpose mechanism that can be used to enable all types of communications 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 communications equipment, this specification does not attempt to dictate how all communications equipment should use the USB. Rather, it defines an architecture that is capable of supporting any communications device. The current release of the specification focuses on supporting connectivity to telecommunications services (devices that have traditionally terminated an analog or digital telephone line), and medium speed networking services ("Always Connected" LAN/WAN media types). The specification currently outlines the common elements needed to support the following types of devices:

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

This specification, and associated subclass specifications, do not attempt to redefine existing standards for connection and control of communications services. The Communications 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 communications devices can be defined and implemented.

## 1.2 Purpose

This specification, and associated subclass specifications, provides information to guide implementers in using the USB logical structures for communications devices. This information applies to manufacturers of communications devices and system software developers.

## 1.3 Related Documents

Detailed examples of communications device classes are provided in separate subclass specifications that are not a part of this specification:

| Reference | Title | Revision |
|---|---|---|
| [USB2.0] | Universal Serial Bus Specification (also referred to as the USB Specification). | 2.0 |
| [USBCCS1.0] | Universal Serial Bus Common Class Specification | 1.0 |
| [USBPSTN1.2] | USB CDC Subclass Specification for PSTN Devices | 1.2 |
| [USBISDN1.2] | USB CDC Subclass Specification for ISDN Devices | 1.2 |
| [USBECM1.2] | USB CDC Subclass Specification for Ethernet Devices | 1.2 |
| [USBNCM1.0] | Universal Serial Bus Communications Class Subclass Specifications for Network Control Model Devices | 1.0 |
| [USBMBIM1.0] | Universal Serial Bus Communications Class Subclass Specifications for Mobile Broadband Interface Model | 1.0 |
| [USBATM1.2] | USB CDC Subclass Specification for ATM Devices | 1.2 |
| [USBWMC1.1] | USB CDC Subclass Specification for Wireless Mobile Communications Devices | 1.1 |
| [USBEEM1.0] | USB CDC Subclass Specification for Ethernet Emulation Devices | 1.0 |
| [ISO3166] | ISO 3166:1993, Codes for the representation of the names of countries. http://www.iso.org | |

## 1.4 Terms and Abbreviations

| | |
|---|---|
| **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. |
| **COMMUNICATIONS INTERFACE** | Refers to a USB interface that identifies itself as using the Communications Class definition. |
| **DATA INTERFACE** | Refers to a USB interface that identifies itself as using the Data Class definition. |
| **DEVICE MANAGEMENT** | Refers to requests and responses that control and configure the operational state of the device. Device management requires the use of a Communications Class interface. |
| **DOWNLINK** | Direction from mobile network to the mobile device |
| **MANAGEMENT ELEMENT** | Refers to a type of USB pipe that manages the communications device and its interfaces. Currently, only the Default Pipe is used for this purpose. |
| **MASTER INTERFACE** | A Communications Class interface which has been designated the master of zero or more interfaces that implement a complete function in a USB communications 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. |
| **TERMINAL** | Entity that represents a starting/ending point for a protocol stack. |

| | |
|---|---|
| **UNION** | A relationship between a collection of one or more interfaces that can be considered to form a functional unit. |
| **UNIT** | Entity that provides the basic building blocks to describe a protocol stack. |
| **UPLINK** | Direction from mobile device to mobile network |

# 2 Management Overview

Several types of communications devices can benefit from the USB. This specification provides common specifications for communications devices. Associated subclass specifications describe models for various telecommunications and networking devices, such as:

1. telephones
2. PSTN (analog) modems
3. ISDN devices
4. Ethernet networking devices
5. ATM networking devices
6. Broadband modems: Cable Modems, DSL Modems
7. Wireless Mobile Communications devices
8. Wireless networking devices
9. Ethernet Emulation Devices

It describes:

- Specifications for:
  - Communications Device Class
  - Communications Interface Class
  - Data Interface Class
- Framework for building a communications device:
  - Assembling the relevant USB logical structures into configurations.
  - Communications Class interface and its usage.
  - Data Class interface and its usage.
  - Usage of additional class types or vendor specific interfaces.

# 3 Functional Overview

This section describes the functional characteristics of the Communications Device Class, Communications 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 Communications Interface Class and Data Interface Class, they are two different classes. All communications devices shall have an interface using the Communications Class to manage the device and optionally specify themselves as communications devices by using the Communications 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 Function Models

A communications device has three basic responsibilities:

- Device management
- Operational management
- Data transmission

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

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

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

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

## 3.2 Communications Device Management

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

To allow device management at the communications device level, a Union shall be made between all the interfaces that make up the functional unit of the device. A functional descriptor is used to define the group of interfaces that make up a functional unit within a device and is outlined in Section 5.2.3.2, "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 communications 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 Communications Class interface providing call control and the various interfaces it was managing at the moment.

## 3.3 Device Operation

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

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

Note: In the case where the device does not choose to identify itself at the device level with the Communications Device Class code, the device shall employ a USB Common Class Feature mechanism that associates multiple interfaces on the device with a single driver in the host. This mechanism is specified in the Interface Association Descriptor ECN to USB 2.0.

Static characteristics of the device, such as the physical connections, are described in terms of the USB device, interface, and endpoint descriptors. The data that moves over the physical interfaces is dynamic in nature, causing the characteristics of the interfaces to change as the data requirements change. These dynamic changes are defined in terms of messages transmitted between the device and host over the Communications 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.

## 3.4 Interface Definitions

Two classes of interfaces are described in this specification: Communications Class interfaces and Data Class interfaces. The Communications Class interface is a management interface and is required of all communications 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 Communications Class interface.

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

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

The Communications 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 Communications Class interface. The management element also meets the requirements for devices as outlined in the *USB Specification*. Call management is provided in the communications 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 Communications Class interface:

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

- 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 Communications Class interface. This corresponds to the Direct Line Control Model, as described in the PSTN Subclass Specification."

- The device provides an internal implementation of call management over the Data Class interface but not the Communications Class interface. In this case, the Communications 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 the PSTN Subclass Specification.

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

### 3.4.2 Data Class Interface

The Data Class defines a data interface as an interface with a class type of Data Class. Data transmission on a communications 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 communications 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 Communications Class interface) to determine the appropriate format to use. As more complicated communications 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 Communications 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.

#### 3.4.2.1 Protocol Data Wrapper

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

The host and device agree upon the wrapper feature at the time the protocol of the data is established. It is the responsibility of the host software and the device to communicate with each other over some other interface (such as a Communications 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 communications devices, two general forms have been defined for the data wrapper header as defined in Table 1. The structure for both forms is the same, the only difference is the usage of the source protocol ID. If no source protocol is needed or known, then offset 3, *bSrcProtocol* is set to 00h. The second form of the data wrapper header allows for both a source and destination protocol for the more structured protocol stack where both a source and destination protocol are needed.

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

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

**Table 1: Data Class Protocol Wrapper Layout**

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

## 3.5 Endpoint Requirements

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

### 3.5.1 Communications Class Endpoint Requirements

The Communications 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 Communications Class-specific requests. The notification element normally uses an interrupt endpoint.

### 3.5.2 Data Class Endpoint Requirement

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).

## 3.6 Device Models

Particular USB communications device configurations are constructed from the interfaces described in previous sections and those described by other class specifications. All communications devices consist of a Communications 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 communications device:

- Device descriptor contains the class code of the Communications Device Class, defined in Table 2. 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 Communications 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 Communications 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.

## 3.7 Function Models

USB 2.0 defines a “function” as a “USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers”. Further, in section 5.2.3, it says “Multiple functions may be packaged into what appears to be a single physical device. A device that has multiple interfaces controlled independently of each other is referred to as a composite device.” We therefore adopt the term “function” to describe a set of one or more interfaces which taken together provide a capability to the host.

This document does not define any communications functions. These are contained in the CDC Subclass documents listed in section 1.3.

# 4 Class-Specific Codes

This section lists the codes for the Communications Device Class, Communications Interface Class and Data Interface Class, including subclasses and protocols. These values are used in the b*DeviceClass*, *bInterfaceClass*, *bInterfaceSubClass*, and *bInterfaceProtocol* fields of the standard device descriptors as defined in chapter 9 of [USB2.0].

## 4.1 Communications Device Class Code

The following table defines the Communications Device Class code:

**Table 2: Communications Device Class Code**

| Code | Class |
|---|---|
| 02h | Communications Device Class |

## 4.2 Communications Interface Class Code

The following table defines the Communications Class code:

**Table 3: Communications Interface Class Code**

| Code | Class |
|---|---|
| 02h | Communications Interface Class |

## 4.3 Communications Class Subclass Codes

The following table defines the currently defined Subclass codes for the Communications Interface Class:

**Table 4 Class Subclass Code**

| Code | Subclass | Reference |
|---|---|---|
| 00h | RESERVED | |
| 01h | Direct Line Control Model | [USBPSTN1.2] |
| 02h | Abstract Control Model | [USBPSTN1.2] |
| 03h | Telephone Control Model | [USBPSTN1.2] |
| 04h | Multi-Channel Control Model | [USBISDN1.2] |
| 05h | CAPI Control Model | [USBISDN1.2] |
| 06h | Ethernet Networking Control Model | [USBECM1.2] |
| 07h | ATM Networking Control Model | [USBATM1.2] |
| 08h | Wireless Handset Control Model | [USBWMC1.1] |
| 09h | Device Management | [USBWMC1.1] |
| 0Ah | Mobile Direct Line Model | [USBWMC1.1] |
| 0Bh | OBEX | [USBWMC1.1] |
| 0Ch | Ethernet Emulation Model | [USBEEM1.0] |
| 0Dh | Network Control  Model | [USBNCM1.0] |

| | | |
|---|---|---|
| 0Eh | Mobile Broadband Interface Model | [USBMBIM1.0] |
| 0Dh-7Fh | RESERVED (future use) | |
| 80-FEh | RESERVED (vendor specific) | |

## 4.4 Communications Class Protocol Codes

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

**Table 5 Communications Interface Class Control Protocol Codes**

| Protocol Code | Reference document | Description |
|---|---|---|
| 00h | USB specification | No class specific protocol required |
| 01h | ITU-T V.250 | AT Commands: V.250 etc |
| 02h | PCCA-101 | AT Commands defined by PCCA-101 |
| 03h | PCCA-101 | AT Commands defined by PCCA-101 & Annex O |
| 04h | GSM 7.07 | AT Commands defined by GSM 07.07 |
| 05h | 3GPP 27.07 | AT Commands defined by 3GPP 27.007 |
| 06h | C-S0017-0 | AT Commands defined by TIA for CDMA |
| 07h | USB EEM | Ethernet Emulation Model |
| 08h-FDh | | RESERVED (future use) |
| FEh | | External Protocol: Commands defined by Command Set Functional Descriptor |
| FFh | USB Specification | Vendor-specific |

## 4.5 Data Class Interface Codes

The following table defines the Data Interface Class code:

**Table 6: Data Interface Class Code**

| Code | Class |
|---|---|
| 0Ah | Data Interface Class |

## 4.6 Data Interface Class SubClass Codes

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

## 4.7 Data Interface Class Protocol Codes

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

**Table 7: Data Interface Class Protocol Codes**

| Protocol Code | Reference document | Description |
|---|---|---|
| 00h | USB specification | No class specific protocol required |
| 01h | [USBNCM1.0] | Network Transfer Block |
| 02h | [USBMBIM1.0] | Network Transfer Block (IP + DSS) |
| 03h – 2Fh | None | RESERVED (future use) |
| 30h | I.430 | Physical interface protocol for ISDN BRI |
| 31h | ISO/IEC 3309-1993 | HDLC |
| 32h | None | Transparent |
| 33h – 4Fh | None | RESERVED (future use) |
| 50h | Q.921M | Management protocol for Q.921 data link protocol |
| 51h | Q.921 | Data link protocol for Q.931 |
| 52h | Q921TM | TEI-multiplexor for Q.921 data link protocol |
| 53h – 8Fh | None | RESERVED (future use) |
| 90h | V.42bis | Data compression procedures |
| 91h | Q.931/Euro- ISDN | Euro-ISDN protocol control |
| 92h | V.120 | V.24 rate adaptation to ISDN |
| 93h | CAPI2.0 | CAPI Commands |
| 94h - FCh | None | RESERVED (future use) |
| FDh | None | Host based driver.<br>Note: This protocol code should only be used in messages between host and device to identify the host driver portion of a protocol stack. |
| FEh | CDC Specification | The protocol(s) are described using a Protocol Unit Functional Descriptors on Communications Class Interface |
| FFh | USB Specification | Vendor-specific |

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

# 5 Descriptors

## 5.1 Standard USB Descriptor Definitions

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

### 5.1.1 Device Descriptor

Communications device functionality resides at the interface level, with the exception being the definition of the Communications Device Class code. The device code is used solely to identify the device as a communications 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 communications devices will have at least one Communications 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 8: Communications Device Class Descriptor Requirements**

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 4 | bDeviceClass | 1 | 02h | Communications Device Class code as defined in Table 2 |
| 5 | bDeviceSubClass | 1 | 00h | Communications Device Subclass code, unused at this time. |
| 6 | bDeviceProtocol | 1 | 00h | Communications Device Protocol code, unused at this time. |

### 5.1.2 Configuration Descriptor

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

### 5.1.3 Interface Descriptor

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

**Table 9: Communications Class Interface Descriptor Requirements**

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 5 | bInterfaceClass | 1 | Class | Communications Interface Class code as defined in Table 3. |
| 6 | bInterfaceSubClass | 1 | Subclass | Communications Interface Subclass code as defined in Table 4. |
| 7 | bInterfaceProtocol | 1 | Protocol | Communications Interface Class Protocol code which applies to the subclass, as specified in the previous field, is defined in Table 5. |

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

**Table 10: Data Class Interface Descriptor Requirements**

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

## 5.2 Endpoint Descriptors

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

### 5.2.1 Class-Specific Device Descriptor

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

### 5.2.2 Class-Specific Configuration Descriptor

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

### 5.2.3 Functional Descriptors

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

**Table 11: Functional Descriptor General Format**

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | *bFunctionLength* | 1 | Number | Size of this descriptor. |
| 1 | *bDescriptorType* | 1 | Constant | CS_INTERFACE, as defined in Table 12. |
| 2 | *bDescriptorSubtype* | 1 | Constant | Identifier (ID) of functional descriptor. For a list of the supported values, see Table 13. |
| 3 | (function specific data0) | 1 | Misc. | First function specific data byte. These fields will vary depending on the functional descriptor being represented. |
| … | … | … | … | … |
| N+2 | (functional specific data N-1) | 1 | Misc. | Nth function specific data byte. These fields will vary depending on the functional descriptor being represented. |

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

**Table 12: Type Values for the bDescriptorType Field**

| Descriptor type | Value |
|---|---|
| CS_INTERFACE | 24h |
| CS_ENDPOINT | 25h |

**Table 13: bDescriptor SubType in Communications Class Functional Descriptors**

| Descriptor subtype | Functional description |
|---|---|
| 00h | Header Functional Descriptor, which marks the beginning of the concatenated set of functional descriptors for the interface. |
| 01h | Call Management Functional Descriptor. |
| 02h | Abstract Control Management Functional Descriptor. |
| 03h | Direct Line Management Functional Descriptor. |
| 04h | Telephone Ringer Functional Descriptor. |
| 05h | Telephone Call and Line State Reporting Capabilities Functional Descriptor. |
| 06h | Union Functional Descriptor |
| 07h | Country Selection Functional Descriptor |
| 08h | Telephone Operational Modes Functional Descriptor |
| 09h | USB Terminal Functional Descriptor |
| 0Ah | Network Channel Terminal Descriptor |
| 0Bh | Protocol Unit Functional Descriptor |
| 0Ch | Extension Unit Functional Descriptor |
| 0Dh | Multi-Channel Management Functional Descriptor |
| 0Eh | CAPI Control Management Functional Descriptor |
| 0Fh | Ethernet Networking Functional Descriptor |
| 10h | ATM Networking Functional Descriptor |
| 11h | Wireless Handset Control Model Functional Descriptor |
| 12h | Mobile Direct Line Model Functional Descriptor |
| 13h | MDLM Detail Functional Descriptor |
| 14h | Device Management Model Functional Descriptor |
| 15h | OBEX Functional Descriptor |
| 16h | Command Set Functional Descriptor |
| 17h | Command Set Detail Functional Descriptor |
| 18h | Telephone Control Model Functional Descriptor |
| 19h | OBEX Service Identifier Functional Descriptor |

| Descriptor subtype | Functional description |
|---|---|
| 1Ah | NCM Functional Descriptor |
| 1Bh | MBIM Functional Descriptor |
| 1Ch | MBIM Extended Functional Descriptor |
| 1Dh-7Fh | RESERVED (future use) |
| 80h-FEh | RESERVED (vendor specific) |

**Table 14: bDescriptor SubType in Data Class Functional Descriptors**

| Code | Description |
|---|---|
| 00h | Header Functional Descriptor, which marks the beginning of the concatenated set of functional descriptors for the interface. |
| 01h-7Fh | RESERVED (future use) |
| 80h-FEh | RESERVED (vendor specific) |

#### 5.2.3.1 Header Functional Descriptor

The class-specific descriptor shall start with a header that is defined in Table 11. The *bcdCDC* field identifies the release of the *USB Class Definitions for Communications Devices Specification* (this Specification) with which this interface and its descriptors comply.

**Table 15: 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 13. |
| 3 | *bcdCDC* | 2 | Number | USB Class Definitions for Communications Devices Specification release number in binary-coded decimal. |

#### 5.2.3.2 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 Communications, Data, or any other valid USB interface class (including, but not limited to, Audio, HID, and Monitor).

**Table 16: 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 13. |
| 3 | *bControlInterface* | 1 | Constant | The interface number of the Communications or Data Class interface, designated as the controlling interface for the union.* |
| 4 | *bSubordinateInterface0* | 1 | Number | Interface number of first subordinate interface in the union. * |
| … | … | … | … | |
| N+3 | *bSubordinateInterfaceN-1* | 1 | Number | Interface number of N-1 subordinate interface in the union. * |

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

### 5.2.3.3 Country Selection Functional Descriptor

The Country Selection Functional Descriptor identifies the countries in which the communications 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 differing regulations according to 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 Communications 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 [ISO3166] specification for more information.

**Table 17: 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 13. |
| 3 | *iCountryCodeRelDate* | 1 | Index | Index of a string giving the release date for the implemented ISO 3166 Country Codes.<br>Date shall be presented as *ddmmyyyy* with *dd*=day, *mm*=month, and *yyyy*=year. |
| 4 | *wCountryCode0* | 2 | Number | Country code in the format as defined in [ISO3166], 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 [ISO3166], release date as specified in offset 3 for Nth country supported. |

## 5.3 Class-Specific Descriptors

Table 18 presents an example of the Communications Class Functional Descriptors for a simple Abstract Control Model device [USBPSTN1.2].

**Table 18: Sample Communications Class Specific Interface Descriptor***

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | *bFunctionLength* | 1 | 05h | Size of this functional descriptor, in bytes. |
| 1 | *bDescriptorType* | 1 | 24h | CS_INTERFACE |
| 2 | *bDescriptorSubtype* | 1 | 00h | Header. This is defined in Table 13, which defines this as a header. |
| 3 | *bcdCDC* | 2 | 0110h | USB Class Definitions for Communications 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 13. |
| 8 | *bmCapabilities* | 1 | 0Fh | This field contains the value 0Fh, because the device supports all the corresponding commands for the Abstract Control Model interface. |
| 9 | *bFunctionLength* | 1 | 05h | Size of this functional descriptor, in bytes |
| 10 | *bDescriptorType* | 1 | 24h | CS_INTERFACE |
| 11 | *bDescriptorSubtype* | 1 | 06h | Union Descriptor Functional Descriptor subtype as defined in Table 13. |
| 12 | *bControlInterface* | 1 | 00h | Interface number of the control (Communications Class) interface |
| 13 | *bSubordinateInterface0* | 1 | 01h | Interface number of the subordinate (Data Class) interface |
| 14 | *bFunctionLength* | 1 | 05h | Size of this functional descriptor, in bytes |
| 15 | *bDescriptorType* | 1 | 24h | CS_INTERFACE |
| 16 | *bDescriptorSubtype* | 1 | 01h | Call Management Functional Descriptor subtype as defined in Table 13. |
| 17 | *bmCapabilities* | 1 | 03h | Indicate that the device handles call management itself (bit D0 is set), and will process commands multiplexed over the data interface in addition to commands sent using *SendEncapsulatedCommand* (bit D1 is set). |
| 18 | *bDataInterface* | 1 | 01h | Indicates that multiplexed commands are handled via data interface 01h (same value as used in the UNION Functional Descriptor) |

* This descriptor is specific to the Communications Class.

# 6 Communications Class Specific Messages

## 6.1 Overview

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

## 6.2 Management Element Requests

The Communications Interface Class supports class-specific requests, defined in this document and in the communications subclass specifications. This section describes the requests that are specific to the Communications Interface Class and common to several subclasses. These requests are sent over the management element and can apply to different device views as defined by the Communications Class interface codes. Detailed descriptions of those requests are provided in those subclass specifications. Table 19 provides a summary table of the Request Code values.

**Table 19: Class-Specific Request Codes**

| Request Code | Value | reference |
|---|---|---|
| SEND_ENCAPSULATED_COMMAND | 00h | 6.2.1 |
| GET_ENCAPSULATED_RESPONSE | 01h | 6.2.2 |
| SET_COMM_FEATURE | 02h | [USBPSTN1.2] |
| GET_COMM_FEATURE | 03h | [USBPSTN1.2] |
| CLEAR_COMM_FEATURE | 04h | [USBPSTN1.2] |
| RESET_FUNCTION | 05h | [USBMBIM1.0] |
| RESERVED (future use) | 06h-0Fh | |
| SET_AUX_LINE_STATE | 10h | [USBPSTN1.2] |
| SET_HOOK_STATE | 11h | [USBPSTN1.2] |
| PULSE_SETUP | 12h | [USBPSTN1.2] |
| SEND_PULSE | 13h | [USBPSTN1.2] |
| SET_PULSE_TIME | 14h | [USBPSTN1.2] |
| RING_AUX_JACK | 15h | [USBPSTN1.2] |
| RESERVED (future use) | 16h-1Fh | |
| SET_LINE_CODING | 20h | [USBPSTN1.2] |
| GET_LINE_CODING | 21h | [USBPSTN1.2] |
| SET_CONTROL_LINE_STATE | 22h | [USBPSTN1.2] |
| SEND_BREAK | 23h | [USBPSTN1.2] |
| RESERVED (future use) | 24h-2Fh | |
| SET_RINGER_PARMS | 30h | [USBPSTN1.2] |
| GET_RINGER_PARMS | 31h | [USBPSTN1.2] |
| SET_OPERATION_PARMS | 32h | [USBPSTN1.2] |
| GET_OPERATION_PARMS | 33h | [USBPSTN1.2] |
| SET_LINE_PARMS | 34h | [USBPSTN1.2] |

| Request Code | Value | reference |
|---|---|---|
| GET_LINE_PARMS | 35h | [USBPSTN1.2] |
| DIAL_DIGITS | 36h | [USBPSTN1.2] |
| SET_UNIT_PARAMETER | 37h | [USBISDN1.2] |
| GET_UNIT_PARAMETER | 38h | [USBISDN1.2] |
| CLEAR_UNIT_PARAMETER | 39h | [USBISDN1.2] |
| GET_PROFILE | 3Ah | [USBISDN1.2] |
| RESERVED (future use) | 3Bh-3Fh | |
| SET_ETHERNET_MULTICAST_FILTERS | 40h | [USBECM1.2] |
| SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER | 41h | [USBECM1.2] |
| GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER | 42h | [USBECM1.2] |
| SET_ETHERNET_PACKET_FILTER | 43h | [USBECM1.2] |
| GET_ETHERNET_STATISTIC | 44h | [USBECM1.2] |
| RESERVED (future use) | 45h-4Fh | |
| SET_ATM_DATA_FORMAT | 50h | [USBATM1.2] |
| GET_ATM_DEVICE_STATISTICS | 51h | [USBATM1.2] |
| SET_ATM_DEFAULT_VC | 52h | [USBATM1.2] |
| GET_ATM_VC_STATISTICS | 53h | [USBATM1.2] |
| RESERVED (future use) | 54h-5Fh | |
| MDLM Semantic-Model specific Requests | 60h – 7Fh | [USBWMC1.2] |
| GET_NTB_PARAMETERS | 80h | [USBNCM1.0] |
| GET_NET_ADDRESS | 81h | [USBNCM1.0] |
| SET_NET_ADDRESS | 82h | [USBNCM1.0] |
| GET_NTB_FORMAT | 83h | [USBNCM1.0] |
| SET_NTB_FORMAT | 84h | [USBNCM1.0] |
| GET_NTB_INPUT_SIZE | 85h | [USBNCM1.0] |
| SET_NTB_INPUT_SIZE | 86h | [USBNCM1.0] |
| GET_MAX_DATAGRAM_SIZE | 87h | [USBNCM1.0] |
| SET_MAX_DATAGRAM_SIZE | 88h | [USBNCM1.0] |
| GET_CRC_MODE | 89h | [USBNCM1.0] |
| SET_CRC_MODE | 8Ah | [USBNCM1.0] |
| RESERVED (future use) | 8Bh-FFh | |

### 6.2.1 *SendEncapsulatedCommand*

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

| bmRequestType | bRequestCode | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|

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

### 6.2.2 *GetEncapsulatedResponse*

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

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

## 6.3 Management Element Notifications

This section defines the Communications Interface Class notifications that the device uses to notify the host of interface, or endpoint events, that are common to several Communications subclasses. Detailed descriptions of those notifications are provided below and in those subclass specifications. Table 20 provides a summary table of the notification code values.

**Table 20: Class-Specific Notification Codes**

| Notification Code | Value | Reference |
|---|---|---|
| NETWORK_CONNECTION | 00h | 6.3.1 |
| RESPONSE_AVAILABLE | 01h | 6.3.2 |
| RESERVED (future use) | 02h-07h | |
| AUX_JACK_HOOK_STATE | 08h | [USBPSTN1.2] |
| RING_DETECT | 09h | [USBPSTN12] |
| RESERVED (future use) | 0Ah-1Fh | |
| SERIAL_STATE | 20h | [USBPSTN1.2] |
| RESERVED (future use) | 21h-27h | |
| CALL_STATE_CHANGE | 28h | [USBPSTN1.2] |
| LINE_STATE_CHANGE | 29h | [USBPSTN1.2] |
| CONNECTION_SPEED_CHANGE | 2Ah | 6.3.3 |
| RESERVED | 2Bh-3Fh | |
| MDML SEMANTIC-MODEL-SPECIFIC NOTIFICATION | 40h-5Fh | [USBWMC1.1] |
| RESERVED (future use) | 60h-FFh | |

### 6.3.1 *NetworkConnection*

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

| bmRequestType | bNotificationCode | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|

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

### 6.3.2 *ResponseAvailable*

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

| bmRequestType | bNotificationCode | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 10100001B | RESPONSE_AVAILABLE | Zero | Interface | Zero | None |

### *6.3.3 ConnectionSpeedChange*

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

| bmRequestType | bNotification | wValue | wIndex | wLength | Data |
|---|---|---|---|---|---|
| 10100001B | CONNECTION_SPEED_ CHANGE | Zero | Interface | 8 | Connection Speed Change Data Structure |

The data phase for this notification contains a data structure with two 32 bit unsigned integers. The two values are the downlink bit rate, followed immediately by the uplink bit rate. (Table 21)

To assure that the host networking driver can always report the correct link speed, the device must send a *ConnectionSpeedChange* notification immediately after every *NetworkConnection* notification is sent. This normally occurs when the physical layer makes or loses a connection, but additionally appears implicitly after the device is reset.

**Table 21: ConnectionSpeedChange Data Structure**

| Offset | Field | Size | Value | Description |
|---|---|---|---|---|
| 0 | DLBitRate | 4 | Number | Contains the downlink bit rate, in bits per second, as sent on the IN pipe |
| 4 | ULBitRate | 4 | Number | Contains the uplink bit rate, in bits per second, as sent on the OUT pipe |

Note that uplink/downlink is the 3gpp definition which is not the same as the upstream/downstream in the [USB2.0] specification.