Universal Serial Bus Communications Class Subclass Specification for ISDN Devices

Revision 1.2

February 9, 2007

Revision History

Rev	Date	Filename	Comments
1.2	2/9/07		Final edits as per February 2007 CDC meeting

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

Bruce Balden	Belcarra
Diego Friedel	AVM
Jun Guo	Broadcom
CC Hung	Mentor Graphics
Dale Self	Symbian
Janne Rand	Nokia
Joe Decuir	MCCI
Joel Silverman	K-Micro
John Turner	Symbian
Ken Taylor	Motorola
Morten Christiansen	Ericsson AB
Peter FitzRandolph	MCCI
Richard Petrie	Nokia
Saleem Mohammed	Synopsys
Tero Soukko	Nokia

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

The USB Communications Device Class Specification 1.1 contains general Communications Class specifications and details for seven device subclasses. That specification has been editorially reorganized into a common USB CDC 1.2 specification [USBCDC1.2] and a set of subclass specifications. This should help implementers understand what is necessary for each device subclass, and facilitate specification maintenance by the USB Device Working Group.

This document is one of those subclass specifications. It contains material technically identical to that contained in USB CDC 1.1. It is intended to guide implementers of USB-connected devices of the types listed in the following section.

1.2 Scope

This document specifies two device subclass intended for use with Communications devices, based on the Universal Serial Bus Class Definitions for Communications Devices specification [USBCDC1.2]. These device subclasses are:

- Multiple Line Control Model
- CAPI Control Model

The intention of this specification is that all material presented here is technically compatible with the previous version of the USB CDC 1.1 Specification, from which it is derived. Numeric codes are defined for subclass codes, protocol codes, management elements, and notification elements.

In some cases material from [USBCDC1.2] is repeated for clarity. In such cases, [USBCDC1.2] shall be treated as the controlling document.

In this specification, the word 'shall' or 'must' is used for mandatory requirements, the word 'should' is used to express recommendations and the word 'may' is used for options.

1.3 Related Documents

Reference	Description
[USB2.0]	Universal Serial Bus Specification, revision 2.0 (also referred to as the USB Specification). http://www.usb.org
[USBCCS1.0]	Universal Serial Bus Common Class Specification, revision 1.0. http://www.usb.org
[USBCDC1.2]	Universal Serial Bus Class Definitions for Communications Devices, Version 1.2. http://www.usb.org.
[USBWMC1.1]	Universal Serial Bus Subclass Specification for Wireless Mobile Communications, Version 1.1.http://www.usb.org
Bellcore NI-1 (National ISDN 1)	Support network terminating services for ISDN service - available at http://www.bellcore.com.

ITU-T I.430	Basic user-network interface – layer 1 specification, 1995, www.itu.int
ITU-T I.431	Primary rate user-network interface – layer 1 specification, 1993, www.itu.int
ITU-T Q.921	ISDN user-network interface data link layer specification, 1993, www.itu.int
ITU-T Q.922	ISDN data link layer specification for frame mode bearer services, 1992, www.itu.int
ITU-T Q.931	ISDN user-network interface—layer 3 specification for basic Call Control , 1993, www.itu.int
ITU-T Q.2931	B-ISDN DSS2 User Network Interface (UNI) Layer 3 Specification for Basic Call/Connection Control. 1995, www.itu.int
ITU-T.200	Programmable communication interface for terminal equipment connected to ISDN appendix II, www.itu.int
ITU-T V.110	Support of data terminal equipment with V-series type interfaces by an integrated services digital network, 1992, www.itu.int
ITU-T V.120	Support by an ISDN terminal adapter equipment with V-series type interface with provision for statistical multiplexing. 1992, www.itu.int
ITU-T X.25	Interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit, 1993, www.itu.int
ITU-T X.75	Packet switched signaling system between public networks providing data transmission services, 1996, www.itu.int
CAPI2.0	COMMON-ISDN-API Version 2.0 – available at http://www.capi.org
ETSI prETS 300 838	Integrated service digital network (ISDN); harmonized programmable communication interface (HPCI) for ISDN. http://www.etsi.org

1.4 Terms and Abbreviations

Term	Description
BRI	ISDN Basic Rate Interface, consisting of one D channel and two B channels.
Byte	For the purposes of this document, the definition of a byte is 8 bits.
Call Management	Refers to a process that is responsible for the setting up and tearing down of calls. This same process also controls the operational parameters of the call. The term “call,” and therefore “call management,” describes processes which refer to a higher level of call control, rather than those processes responsible for the physical connection.
CAPI	COMMON-ISDN-API
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.
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 Communications Class interface.
Descriptor	Data structure used to describe a USB device capability or characteristic
Device	A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical. When used as a non-specific reference, a USB device is either a hub or a function.
DTE	Data Terminal Equipment; for example, a Personal Computer.
Function	Device capabilities exposed over the USB cable.

HEC	Header Error Control
ISDN	Integrated Services Digital Network.
I.430	ISDN BRI physical interface standard. See ITU-T I.430 above
I.431	ISDN PRI physical interface standard. See ITU-T I.431 above
ITU	International Telecommunications Union (formerly CCITT).
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.
MULTIFUNCTION DEVICE	A device of peripheral that exposes one or more functions or services to an end user. Exposed services can but do have to be exposed as USB functions.
NI-1	National ISDN 1 is intended to be a set of standards, which every manufacture can conform to for building switch independent ISDN devices. Future standards, denoted as NI-2 and NI-3, are currently being developed.
NOTIFICATION ELEMENT	Refers to a type of USB pipe. Although a notification element is not required to be an interrupt pipe, a notification element is typically defined in this way.
NOTIFICATION ELEMENT	Refers to a type of USB pipe. Although a notification element is not required to be an interrupt pipe, a notification element is typically defined in this way.
PDU	Protocol Data Unit - A combination of the SDU and the current protocol layer's header and/or trailer.
PRI	Primary Rate Interface, which consists of one or two D channels and up to 30 B channels.
SDU	Service Data Unit – User data and control information created at the upper protocol layers that is transferred transparently through a primitive by layer (N+1) to layer (N) and subsequently to (N-1).
TA	Terminal Adaptor
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.
VIDEO PHONE	A device which simultaneously sends voice and video with optional data. For example: ITU-T H.324

2 Management Overview

This subclass specification includes specifications for three kinds of devices that connect to the Integrated Services Digital Network (ISDN):

- Basic Rate (BRI) Terminal Adaptors
- Primary Rate (PRI) Terminal Adaptors
- Telephones

This specification address ISDN Terminal Adaptors.

This specification addresses two control models:

- Multi-Line Control Model, which addresses ISDN TA resources individually
- CAPI, Common-ISDN-API, mapped through USB

Devices of these subclasses must conform to:

- USB Specification [USB2.0]
- Device Framework
- Communications Device Class 1.2 [USBCDC1.2]

3 Functional Overview

3.1 Function Models

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

Functions defined in this specification may be used as part of multifunction devices or as single-function devices.

3.2 USB ISDN Device Models

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

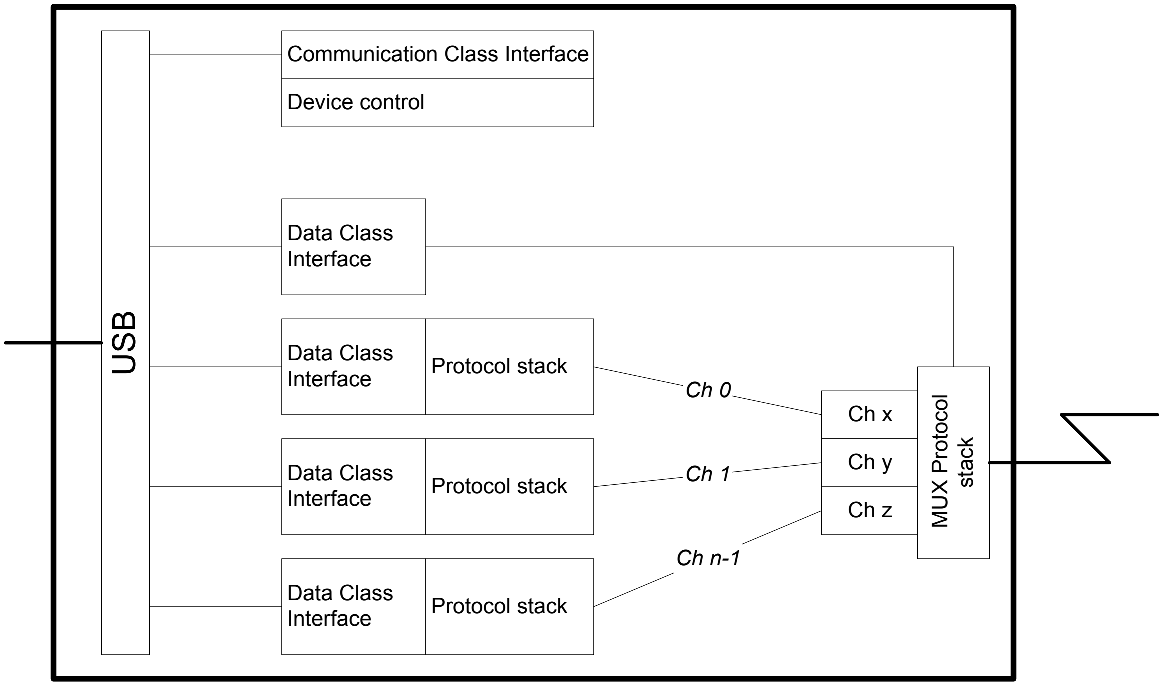
3.3 Multi-Channel Model

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

The prime characteristic of a Multi-Channel device is its ability to multiplex several channels on a physical network interface using a *MUX protocol stack*. Assuming there are *n* channels (*x .. z*) on the physical interface, where *n* is network and device specific, physical channels are mapped to a standard set of channels (*0 .. n-1*) by the protocol stack. The standard channels carrying data with unspecified format are then explicitly mapped to some USB interface (See Section 5.3.2for more details). The protocol stack may also expose an USB interface for protocol management.

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

Figure 1: Multi Line Control Model for a Basic-Rate Configuration



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

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

3.4 Multi Channel Model Topology

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

- **Units:**
  Units are Entities that provide the basic building blocks to describe different protocol stacks. There are two kinds of Units: Protocol and Extension. Protocol Units identify instances of protocols defined

in this document. Extension Units are vendor specific extensions to this set. Each Unit has one or more Child pins used to connect to its immediate neighboring Unit or Terminal below it on the stack.

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

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

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

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

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

3.5 USB CAPI Model

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

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

3.6 CAPI Control Model

With a CAPI Control Model, the USB device understands CAPI commands and CAPI messages. The device will make use of both a Data Class Interface and a Communications Class interface, see Figure 2.

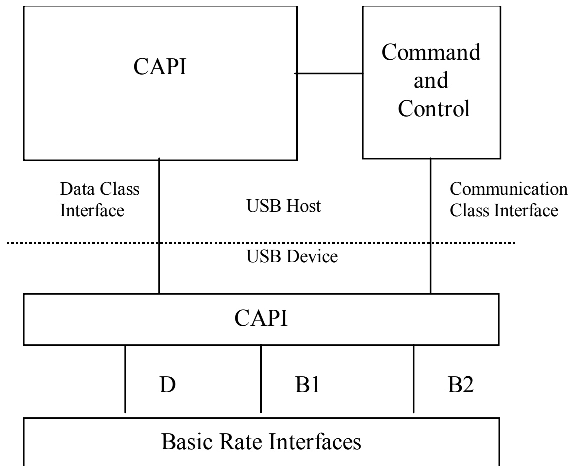


Figure 2: CAPI Control Model for a Basic-Rate Configuration

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

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

The CAPI device reports its implemented functionality using a Communications Class specific request, in order to allow the host to choose an appropriate upper part of the CAPI to run within the host. The request is transported via the Communications Class interface for the device and presented in Table 11.

The only class specific request which is valid for a Communications Class interface with a Communications Class SubClass code of the CAPI Control Model is listed in Table 11 below. The other class specific requests not listed in the above table, such as *SendEncapsulatedCommand*, are inappropriate for a CAPI Control Model and shall generate a STALL condition if sent to such an interface.

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 *DeviceClass*, *bInterfaceClass*, *bInterfaceSubClass*, and *bInterfaceProtocol* fields of the standard device descriptors as defined in chapter 9 of [USB2.0].

4.1 Communications Class Code

This is defined in [USBCDC1.2].

4.2 Communications Class Subclass Codes

The following table defines the Communications Subclass codes:

Table 1 Class Subclass Code

Code	Subclass
04h	Multi Line Control Model
05h	CAPI Control Model

4.3 Communications Class Protocol Codes

[USBCDC1.2] defines Communications Class Protocols.

The following table lists the Protocol codes used in this subclass specification:

Table 2 Class Protocol Code

Code	Protocol
00h	No class specific protocol required

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

4.4 Data Interface Class and Subclass Codes

[USBCDC1.2] defines the data interface class code: 0Ah.

[USBCDC1.2] specifes that the Data Interface Subclass field is unused, and should be set to 00h.

4.5 Data Class Interface Protocol Codes

[USBCDC1.2] defines Data Interface Class Protocols codes.

The following table lists the Protocol codes used in this subclass specification:

Table 3 Data Interface Class Protocol Codes

Code	Protocol	
00h	No class specific protocol required	No class specific protocol required
01h – 2Fh	None	RESERVED (future use)
30h	I.430	Physical interface protocol for ISDN BRI
31h	ISO/IEC 3309-1993	HDLC
32h	None	Transparent
33h – 4Fh	None	RESERVED (future use)
50h	Q.921M	Management protocol for Q.921 data link protocol
51h	Q.921	Data link protocol for Q.931
52h	Q921TM	TEI-multiplexor for Q.921 data link protocol
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-FFh	See [USBCDC1.2]	

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

Devices that conform to this subclass specification need to implement the standard USB descriptors for the Communications Device Class, Communications Interface Class and Data Interface Class. These are defined in [USBCDC1.2].

5.2 Class-Specific Descriptors

Devices that conform to this subclass specification may need to implement class-specific descriptors for the Communications Interface Class and Data Interface Class. These are defined in [USBCDC1.2].

5.3 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.

[USBCDC1.2] describes functional descriptors that may be used in all Communications Subclasses. These include:

- Header Functional Descriptor
- Union Functional Descriptor
- Country Selection Functional Descriptor

The following Functional Descriptors are specific to ISDN subclass devices.

5.3.1 USB Terminal Functional Descriptor

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

Table 4: USB Terminal Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	USB Terminal Functional Descriptor Subtype as defined in [USBCDC1.2]

Offset	Field	Size	Value	Description
3	*bEntityId*	1	Constant	Constant uniquely identifying the Terminal
4	*bInInterfaceNo*	1	Number	The input interface number of the associated USB interface.
5	*bOutInterfaceNo*	1	Number	The output interface number of the associated USB interface.
6	*bmOptions*	1	Bitmap	D7..D1: RESERVED (Reset to zero) D0: Protocol wrapper usage 0 - No wrapper used 1 - Wrapper used
7	*bChildId0*	1	Constant	First ID of lower Terminal or Unit to which this Terminal is connected.
…	…	…	…	
6+N	*bChildIdN-1*	1	Constant	Nth ID of lower Terminal or Unit to which this Terminal is connected.

5.3.2 Network Channel Terminal Functional Descriptor

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

Table 5: Network Channel Terminal Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Network Channel Terminal Functional Descriptor Subtype as defined in [USBCDC1.2]
3	*bEntityId*	1	Constant	Constant uniquely identifying the Terminal
4	*iName*	1	Index	Index of string descriptor, describing the name of the Network Channel Terminal.
5	*bChannelIndex*	1	Number	The channel index of the associated network channel according to indexing rules below.
6	*bPhysicalInterface*	1	Constant	Type of physical interface: 0 – None 1 – ISDN 2 to 200 – RESERVED (future use) 201 to 255 - Vendor specific

Channel Indexing Rule

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

5.3.3 Protocol Unit Functional Descriptor

A communication protocol stack is a combination of communication functions (protocols) into a layered structure. Each layer in the stack presents some abstract function for the layer above according to some layer-interface-standard, making it possible to replace a function with another as long as it conforms to the standard. Each layer may have a set of protocol parameters, defined in Appendix E, to configure it for proper operation in the actual environment and the parameters may be retrieved and/or modified. The Unit state is initially reset. See Section 6.2.1 "*SetUnitParameter*", Section 6.2.2 "*GetUnitParameter*", and Section 6.2.3 "*ClearUnitParameter*" for details.

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

Table 6: Protocol Unit Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Protocol Unit Functional Descriptor Subtype as defined in [USBCDC1.2]
3	*bEntityId*	1	Constant	Constant uniquely identifying the Unit
4	*bProtocol*	1	Protocol	Protocol code as defined in [USBCDC1.2]
5	*bChildId0*	1	Constant	First ID of lower Terminal or Unit to which this Terminal is connected.
…	…	…	…	
4+N	*bChildIdN-1*	1	Constant	Nth ID of lower Terminal or Unit to which this Terminal is connected.

5.3.4 Extension Unit Functional Descriptor

The *Extension Unit Functional Descriptor* provides minimal information about the Extension Unit for a generic driver at least to notice the presence of vendor-specific components within the protocol stack. The *bExtensionCode* field may contain a vendor-specific code that further identifies the Extension Unit. If it is not used, it should be set to zero. The Unit may have a set of vendor specific parameters to configure it for proper operation in the actual environment and the parameters may be retrieved and/or modified. The Unit state is initially reset. Set Section 6.2.1 "SetUnitParameter", Section 6.2.2 "*GetUnitParameter*", and Section 6.2.3 "*ClearUnitParameter*" for details.

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

Table 7: Extension Unit Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Extension Unit Functional Descriptor Subtype as defined in [USBCDC1.2]
3	*bEntityId*	1	Constant	Constant uniquely identifying the Unit

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

5.3.5 Multi-Channel Management Functional Descriptor

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

Table 8: Multi-Channel Management Functional Descriptor

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	Multi-Channel Management functional descriptor subtype, as defined in [USBCDC1.2]
3	*bmCapabilities*	1	Bitmap	The capabilities that this configuration supports. (A value of zero means that the request or notification is not supported.) D7..D3: RESERVED (Reset to zero) D2: 1 – Device supports the request Set_Unit_Parameter. D1: 1 – Device supports the request Clear_Unit_Parameter. D0: 1 – Device stores Unit parameters in non-volatile memory. The previous bits identify which requests are supported by a Communications Class interface with the SubClass code of Multi-Channel Control Model.

5.3.6 CAPI Control Management Functional Descriptor

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

Table 9: CAPI Control Management Functional Descriptor

Offset	Field	Size	Value	Description

Offset	Field	Size	Value	Description
0	*bFunctionLength*	1	Number	Size of this functional descriptor, in bytes.
1	*bDescriptorType*	1	Constant	CS_INTERFACE
2	*bDescriptorSubtype*	1	Constant	CAPI Control Management Functional Descriptor Subtype as defined in [USBCDC1.2]
3	*bmCapabilities*	1	Bitmap	Which capabilities are supported by this configuration. D7..D1: RESERVED (reset to zero). D0: 1 – device is an Intelligent CAPI device 0 – device is a Simple CAPI device The above bits, in combination, identify which requests/notifications are supported by a Communications Class interface with the protocol code of CAPI Control.

6 Communications Class Specific Messages

6.1 Overview

The Communications Interface Class supports the standard requests defined in chapter 9 of [USB2.0]. In addition, the Communications Interface Class has some class-specific requests and notifications. These are used for device and call management.

Requests for controlling the interface between the USB ATM device are presented in Section 6.2. There are also some additional signals that shall go back to the host as notifications, which are represented in section 6.3. These requests and notifications are transported via the Communications Class interface for the device.

6.2 Management Element Requests

Devices conforming to this subclass specification use the following requests. The only class-specific request codes that are valid for a Communications Class interface with a Communications Class SubClass codes of MCM or CAPI are listed in the following tables.

A *Multi-Channel communication device* uses a Communications Class interface for device management with a Communications Class SubClass code of Multi-Channel. The only class-specific request codes that are valid for this SubClass code are listed in Table 10. All other class-specific requests not listed in the table are inappropriate for a Multi-Channel Model and would generate a STALL condition if sent to such an interface.

Table 10: Requests — Multi-Channel Model*

Request	Summary	Req'd/Opt	reference
SetUnitParameter	Used to set a Unit specific parameter	Optional	6.2.1
GetUnitParameter	Used to retrieve a Unit specific parameter	Required	6.2.2
ClearUnitParameter	Used to set a Unit specific parameter to its default state.	Optional	6.2.3

* These requests are specific to the Communications Class.

A *CAPI Control Model device* uses a Communications Class interface for device management with a Communications Class SubClass code of CAPI. The only class specific request which is valid for a Communications Class interface with a Communications Class SubClass code of the CAPI Control Model is listed in Table 11 below. The other class specific requests not listed in the above table, such as *SendEncapsulatedCommand*, are inappropriate for a CAPI Control Model and shall generate a STALL condition if sent to such an interface.

Table 11: Requests — CAPI Control Model*

Request	Summary	Req'd/Opt	reference
GetProfile	Returns the implemented capabilities of the device	Required	6.2.4

* These requests are specific to the Communications Class.

The following table describes the requests that are specific to the Communications Interface Class ISDN Subclass.

Table 12: Class-Specific Request Codes for ISDN subclasses

Request	Value
SET_UNIT_PARAMETER	37h
GET_UNIT_PARAMETER	38h
CLEAR_UNIT_PARAMETER	39h
GET_PROFILE	3Ah
RESERVED (future use)	3Bh-3Fh

6.2.1 SetUnitParameter

This request sets the value of a parameter belonging to a Unit identified by Unit Parameter Structure, see Table 13. The timing of when the new parameter takes effect depends on the protocol or vendor specific function.

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

Table 13: Unit Parameter Structure

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

6.2.2 GetUnitParameter

This request returns the current value of a parameter belonging to a Unit pointed out by Unit Parameter Structure, see Table 13.

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

6.2.3 ClearUnitParameter

This request restores the default value of a parameter belonging to a Unit identified by Unit Parameter Structure, see Table 13. The timing of when the new parameter takes effect depends on the protocol or vendor specific function.

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data

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

6.2.4 GetProfile

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

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

6.3 Management Element Notifications

[USBCDC1.2] defines the common Communications Interface Class notifications that the device uses to notify the host of interface, or endpoint events. There are no Notification Elements that are specific to the ISDN subclasses.

Appendix A: Sample Configurations

A.1 CAPI Device Configuration

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

Table 14: Example CAPI Device Configurations

Example Configuration	Interface (Class Code)	Description
Intelligent CAPI Device	Communications Class	Device Management consisting of a Management Element
	Data Class	CAPI Messages
Simple CAPI Device	Communications Class	Device Management consisting of a Management Element
	Data Class	CAPI Messages for simple CAPI Devices

Communications Class Interfaces and Data Class Interfaces are defined in [USBCDC1.2].

Appendix B: Multi-channel ISDN B-Channel setup

B.1 General

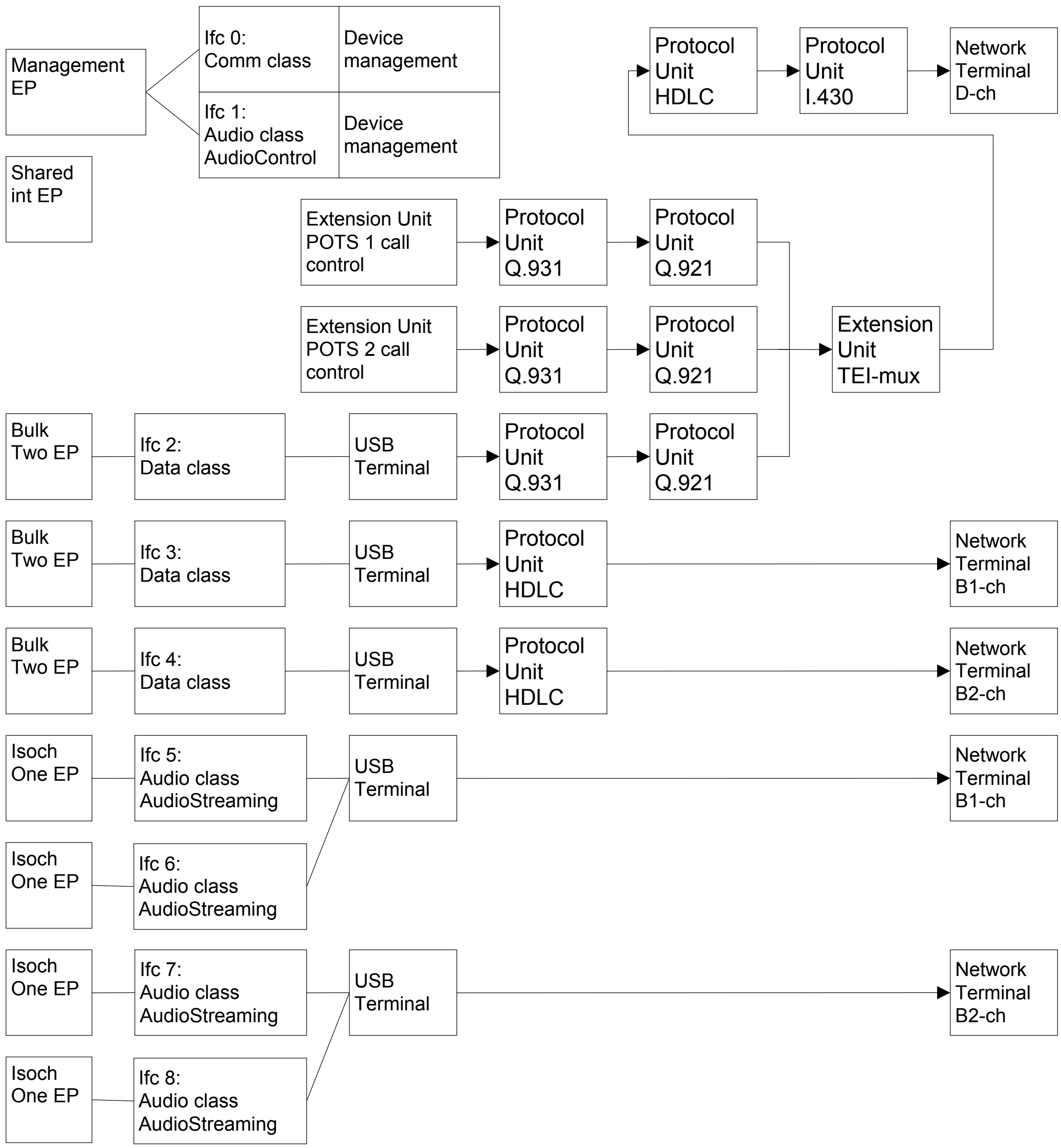
The basic idea is that call control to/from the host is performed through some sort of protocol state machine. This state machine will be specific for each protocol due to the problems in finding one generic that will suit all possible protocol derivatives that are in use, i.e. for ISDN layer 3 there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

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

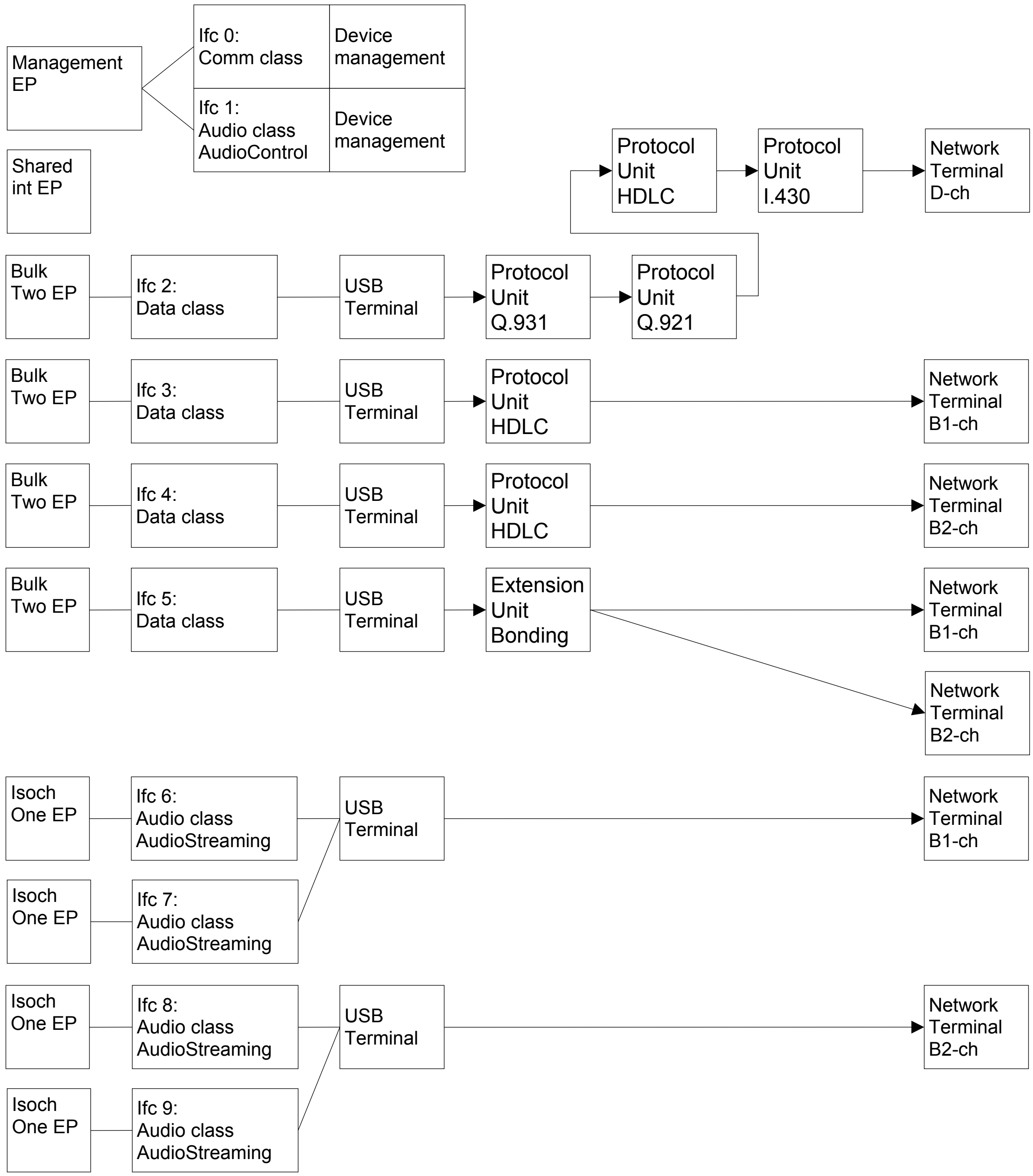
For each interface there exists a number of alternate settings. By incorporating an alternate setting with *bNumEndpoints* = 00h for each interface involved in data transfer, a device offers to the host the option to temporarily relinquish USB bandwidth. If such setting is implemented, it must be as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

Appendix C: Multi-Channel Implementation Examples

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



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



Appendix D: Data Class Protocol Definitions

Definitions

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

Table 15: Command Type Encoding

Command type	Value
REQ	XXXXXX00b
CON	XXXXXX11b
IND	XXXXXX01b
RES	XXXXXX10b

D.1 Physical Interface Protocols

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

Protocol code: According to Table 3.

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

Table 16: I.430 Configuration Parameter List

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

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

Table 17: I.430 Command Message Format

Command	Corresponding ITU I.430 Primitive	ITU I.430 Message Reference
I430_PH_DATA_REQ	PH-DATA request	2.3 Primitives between layer 1 and the other entities, Note 1
I430_PH_ACTIVATE_REQ	PH-ACTIVATE request	6.2.1.3 Activate primitives
I430_PH_ACTIVATE_IND	PH-ACTIVATE indication	6.2.1.3 Activate primitives
I430_PH_ACTIVATE_B_REQ	N.A.	Note 2
I430_PH_DEACTIVATE_IND	PH-DEACTIVATE indication	6.2.1.4 Deactivate primitives, Note 4
I430_PH_DEACTIVATE_B_REQ	N.A.	Note 3
I430_MPH_ERROR_IND	MPH-ERROR indication	6.2.1.5 Management primitives
I430_MPH_ACTIVATE_IND	MPH-ACTIVATE indication	6.2.1.3 Activate primitives
I430_MPH_DEACTIVATE_REQ	MPH-DEACTIVATE request	6.2.1.4 Deactivate primitives
I430_MPH_DEACTIVATE_IND	MPH-DEACTIVATE indication	6.2.1.4 Deactivate primitives
I430_MPH_INFORMATION_IND	MPH-INFORMATION indication	6.2.1.5 Management primitives

Commands according to I.430 "Table E.1 I.430 Primitives associated with layer 1".
Note 1: PH-DATA request does not have a data field since this function is performed by other protocols such as HDLC. Therefore PH-DATA is excluded from the command list since it doesn't have any function.
Note 2: This primitive is an USB extension to enable a specific B-channel.
Note 3: This primitive is an USB extension to disable a specific B-channel.
Note 4: This primitive will deactivate the physical layer connection (including all B-channels).

Table 18: I.430 Commands

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

NOTE 1: 'NN' in **Value** encoded according to Table 15.
NOTE 2: X : Exists
- : Does not exist

Table 19: I.430 Activate, Deactivate Command Wrapper

Offset	Field	Size	Value	Description

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ACTIVATE_REQ, I430_PH_ACTIVATE_IND, I430_PH_DEACTIVATE_IND, I430_MPH_ACTIVATE_IND, I430_MPH_DEACTIVATE_REQ, I430_MPH_DEACTIVATE_IND command as defined in Table 18

Table 20: I.430 PhActivateBReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ACTIVATE_B_REQ command as defined in Table 18
1	*bChannel*	1	Number	Index of B-channel to activate 0 – RESERVED 1 – B1-channel 2 – B2-channel 3 to FFh – RESERVED

Table 21: I.430 PhDeactivateBReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_DEACTIVATE_B_REQ command as defined in Table 18
1	*bChannel*	1	Number	Index of B-channel to deactivate 0 – RESERVED 1 – B1-channel 2 – B2-channel 3 to FFh - RESERVED

Table 22: I.430 PhDataReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_DATA_REQ command as defined in Table 18
1	*bPriority*	1	Number	D-channel transmit data priority class 0 – Priority class 1 1 – Priority class 2 2 to FFh – RESERVED

Table 23: I.430 MphErrorInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_ERROR_IND command as defined in Table 18

Offset	Field	Size	Value	Description
1	*bInfo*	1	Number	Error codes 0 = Error, RX info 0 1 = Error, RX info 2 2 = Error, lost framing 3 = Recover from error, RX info 0 4 = Recover from error, RX info 2 5 = Recover from error, RX info 4 6 = FFh RESERVED

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

Table 24: I.430 MphInformationInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	I430_PH_INFORMATION_IND command as defined in Table 18
1	*bInfo*	1	Number	0 : Physical layer disconnected 1 : Physical layer connected 2 .. FFh: RESERVED (future use)

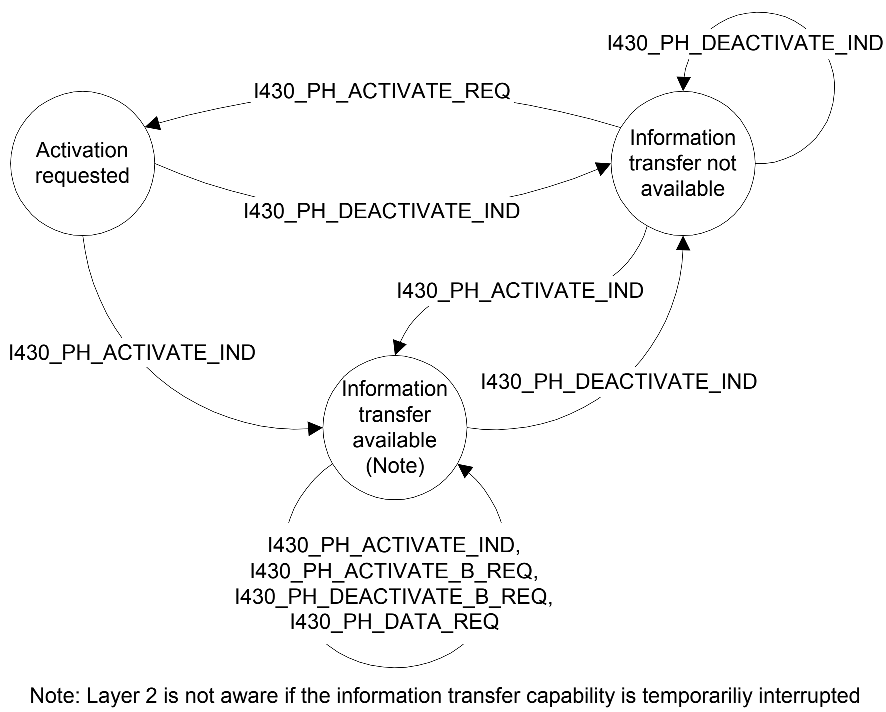


Figure 3: I.430 Layer 1 and Layer 2

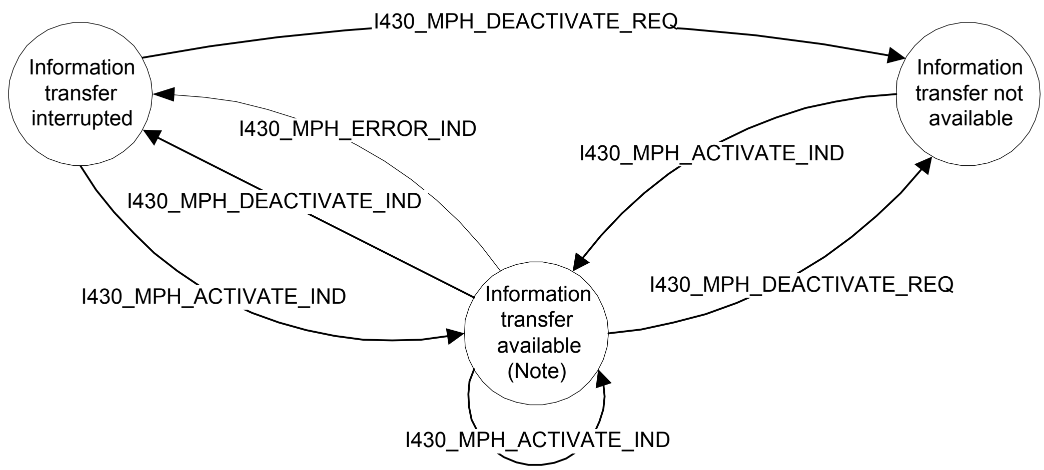


Figure 4: Layer 1 - Management Network Side

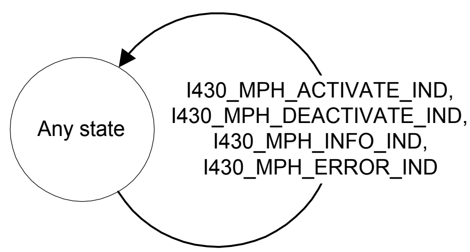


Figure 5: Layer 1 - Management User Side

D.2 Framing Protocols

D.2.1 HDLC Framing

Protocol code: According to Table 3

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

Table 25: HDLC Configuration Parameter List

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

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

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

Table 26: HDLC Commands

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

NOTE 1: 'NN' in **Value** encoded according to Table 15.
NOTE 2: X : Exists
- : Does not exist

Table 27: HDLC ControlRes, StatusReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_CONTROL_RES or HDLC_STATUS_REQ command as defined in Table 26

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

Table 28: HDLC ControlReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_CONTROL_REQ command as defined in Table 26
1	*bmControl*	1	Bitmap	D7..D3: RESERVED (Reset to zero) D2: Receiver abort 0 – No action 1 – Abort ongoing RX and abort pending RX buffers D1: FCS generation 0 – Generate correct FCS 1 – Generate bad FCS D0: Transmitter abort 0 – No action 1 – Abort ongoing TX and discard pending buffers ahead of this command

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

Table 29: HDLC StatusInd/Res Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_STATUS_IND or HDLC_STATUS_RES command as defined in Table 26

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

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

Table 30: HDLC DataReq/Ind Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	HDLC_DATA_REQ or HDLC_DATA_IND command as defined in Table 26
1	*bProtocolData0*	1	Number	First byte with protocol data. FCS excluded.
…	…	…	…	
0+N	*bProtocolDataN-1*	1	Number	Nth byte with protocol data. FCS excluded.

D.2.2 Transparent framing

Protocol code: According to Table 3

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

Table 31: TRANS Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmConfig*	1	Bitmap	D7..D2: RESERVED (Reset to zero) D1: TX and RX data handling 0 – TX data is transmitted to the line. RX data is sent to the host. 1 - TX data is transmitted to the line and looped back to RX. RX data from the line is ignored. D0: Data encoding 0 – NRZ 1 – NRZI

bParameterIndex	Field	Size	Value	Description
1	*bmConfigCapabilities*	1	Bitmap (read only)	D7..D1: RESERVED (Reset to zero) D0: NRZI encoding option 0 – NRZI not available 1 – NRZI available

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

Table 32: TRANS Commands

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

NOTE 1: 'NN' in **Value** encoded according to Table 15.
NOTE 2: X : Exists
- : Does not exist

Table 33: TRANS ControlRes, StatusReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_CONTROL_RES or TRANS_STATUS_REQ command as defined in Table 32

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

Table 34: TRANS ControlReq Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_CONTROL_REQ command as defined in Table 32
1	*bmControl*	1	Bitmap	D7..D3 : RESERVED (Reset to zero) D2: Transmitter data underrun option 0 – Transmit continuous idle mark whenever the transmit buffer is underrun. 1 – Repeatedly transmit the last buffer received from the host whenever the transmit buffer is underrun (for tones, etc) D1: Receiver abort 0 – No action 1 – Abort ongoing RX and abort pending RX buffers. D0: Transmitter abort 0 – No action 1 – Abort ongoing TX and discard pending buffers ahead of this one

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

Table 35: TRANS StatusInd/Res Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_STATUS_IND or TRANS_STATUS_RES command as defined in Table 32
1	*bmStatus*	1	Bitmap	D7 .. D2 : RESERVED (Reset to zero) D1: Transmitter underrun 0 – No transmitter underrun has occurred 1 – Transmitter underrun has occurred D0: Receive overrun 0 – No received data discarded due to overrun 1 – Received data discarded due to overrun

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

Table 36: TRANS DataReq/Ind Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	TRANS_DATA_REQ or TRANS_DATA_IND command as defined in Table 32
1	*bProtocolData0*	1	Number	First byte with protocol data
0+N	*bProtocolDataN-1*	1	Number	Nth byte with protocol data

D.3 Data Link Protocols

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

Protocol code: According to Table 3.

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

Table 37: Q.921M Configuration Parameter List

bParameterIndex	Field	Size	Value	Description

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

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

Table 38: Q.921M Command Message Format

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

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

Table 39: Q.921M Commands

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

NOTE 1: ‘NN’ in Value encoded according to Table 15.
NOTE 2: X : Exists
- : Does not exist

The Q921M_DL_DATA_xxx and Q921M_DL_UNIT_DATA_xxx follow the message structure according to Table 46

Table 40: Q.921M DlAssignReq wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ASSIGN _REQ command as defined Table 39

Table 41: Q.921M DlAssignInd, DlRemoveInd Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ASSIGN _IND, Q921M_DL_REMOVE _IND command as defined Table 39
1	*bTei*	1	Number	TEI value

Table 42: Q.921M DlErrorReq, DlErrorCon Command Wrapper

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Q921M_DL_ERROR_IND, Q921M_DL_ERROR_CON command as defined in Table 39
1	*bError*	1	Number	Error code according to Q.921 "TABLE II.1/Q.921 Management Entity Actions for MDL-Error-Indications" where A=1, B=2, …

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

Protocol code: According to Table 3

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

Table 43: Q.921 Configuration Parameter List

bParameterIndex	Field	Size	Value	Description
0	*bmOptions*	1	Bitmap	D7..D1 RESERVED (Reset to zero) D0 0 – User side 1 – Network side
1	*bT200*	1	Number	Maximum time until an acknowledgment. must be received after the transmission of an I-frame
2	*bN200*	1	Number	Maximum number of retransmissions of a frame
3	*bN201*	1	Number	Maximum number of bytes in an information field
4	*bK*	1	Number	Maximum number of outstanding I-frames

bParameterIndex	Field	Size	Value	Description
5	*bT203*	1	Number	Maximum time allowed without frames being exchanged
6	*bSAPI*	1	Number	SAPI value according Table 2 of Q.921

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

Table 44: Command Message Format

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

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

Table 45: Q.921 commands

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

NOTE 1: 'NN' in Value encoded according to Table 15.
NOTE 2: X : Exists
- : Does not exist

Table 46: Q.921 General Message Structure

Offset	Field	Size	Value	Description
0	*bCommand*	1	Number	Command according to Table 39 and Table 45
1	*bMessageData0*	1	Parameter	First byte with optional parameter data associated with the message
0+N	*bMessageDataN-1*	1	Parameter	Nth byte with optional parameter data associated with the message

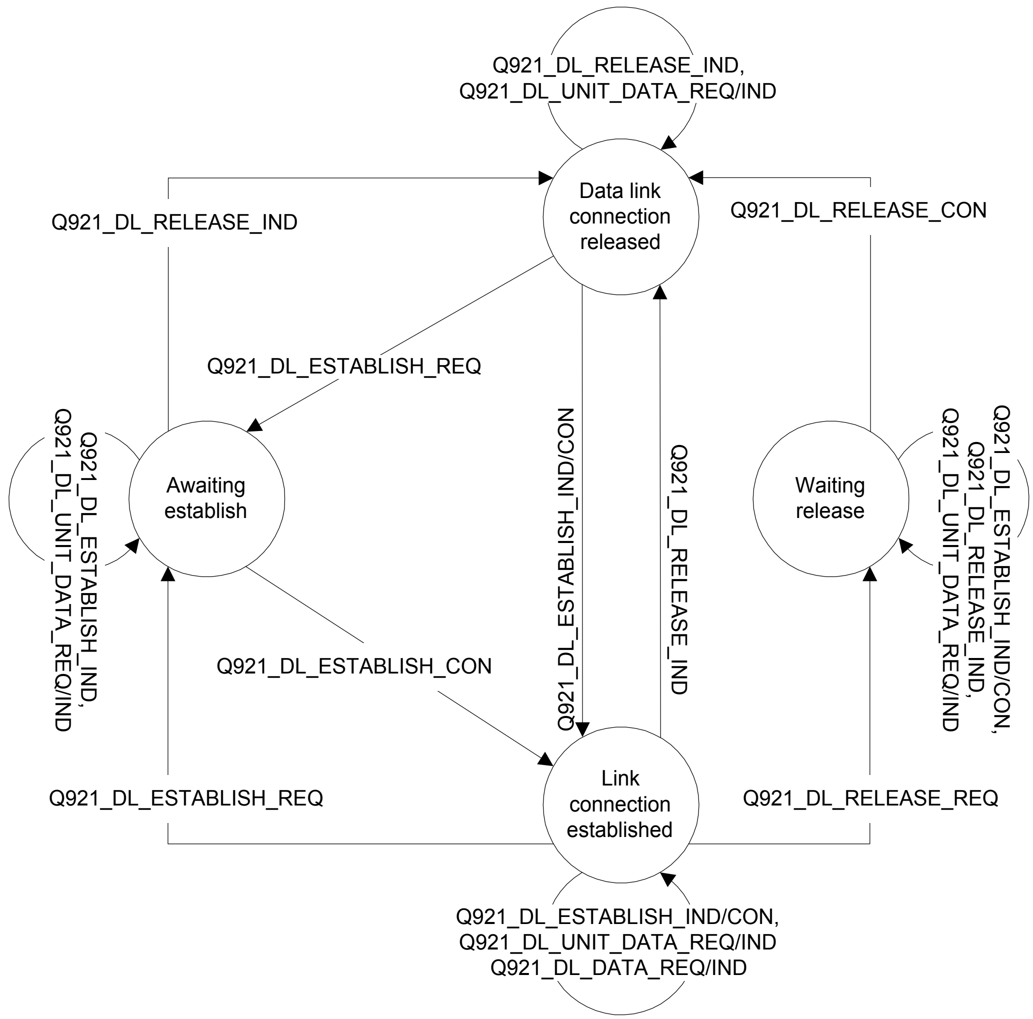


Figure 6: State Transition Diagram for Q.921

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

Protocol code: According to Table 3

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

D.4 Network layer Protocols

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

Protocol code: According to Table 3.

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

NOTE: Extensions for symmetric call operation are not supported.

Most commands use a general command structure as defined in Table 51 that corresponds to the message format defined in Q.931 chapter 4.1. The ones that don't are explicitly defined within this document.

Table 47: Q.931/Euro-ISDN Configuration Parameter List

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

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

Table 48: Q.931/Euro-ISDN Command Message Format

Command	Corresponding ITU Q.931 call control primitive	ITU Q.931 message format reference
Q931_ALERT_REQ	Alerting request	3.1.1 Alerting
Q931_ALERT_IND	Alerting indication	3.1.1 Alerting
Q931_DISC_REQ	Disc request	3.1.5 Disconnect

Command	Corresponding ITU Q.931 call control primitive	ITU Q.931 message format reference
Q931_DISC_IND	Disc indication	3.1.5 Disconnect
Q931_ERROR_IND	Error indication	3.1.5 Disconnect /Note 1
Q931_GET_STATISTICS_REQ	N.A.	N.A.
Q931_GET_STATISTICS_CON	N.A.	N.A.
Q931_INFO_REQ	Info request	3.1.6 Information
Q931_INFO_IND	Info indication	3.1.6 Information
Q931_LINK_FAIL_IND	Link fail indication	3.1.5 Disconnect /Note 1
Q931_MORE_REQ	More info request	3.1.15 Setup acknowledge
Q931_MORE_IND	More info indication	3.1.15 Setup acknowledge
Q931_NOTIFY_REQ	Notify request	3.1.7 Notify
Q931_NOTIFY_IND	Notify indication	3.1.7 Notify
Q931_PROCEED_REQ	Proceeding request	3.1.2 Call proceeding
Q931_PROCEED_IND	Proceeding indication	3.1.2 Call proceeding
Q931_PROGRESS_REQ	Progress request	3.1.8 Progress
Q931_PROGRESS_IND	Progress indication	3.1.8 Progress
Q931_REJECT_REQ	Reject request	3.1.10 Release complete
Q931_REJECT_IND	Reject indication	3.1.10 Release complete
Q931_RELEASE_REQ	Release request	3.1.9 Release
Q931_RELEASE_IND	Release indication	3.1.9 Release
Q931_RELEASE_CON	Release confirm	3.1.9 Release
Q.931_RESTART_REQ	Management restart request	3.4.1 Restart
Q.931_RESTART_CON	Management restart acknowledge	3.4.2 Restart acknowledge
Q931_RESUME_REQ	Resume request	3.1.11 Resume
Q931_RESUME_CON	Resume confirm (ok)	3.1.12 Resume acknowledge
Q931_RESUME_CON	Resume confirm (error)	3.1.12 Resume reject
Q931_SETUP_REQ	Setup request	3.1.14 Setup
Q931_SETUP_IND	Setup indication	3.1.14 Setup
Q931_SETUP_RES	Setup response	3.1.3 Connect
Q931_SETUP_CON	Setup confirm (ok)	3.1.3 Connect
Q931_SETUP_CON	Setup confirm (error)	3.1.5 Disconnect /Note 1
Q931_COMPLETE_IND	Setup complete indication (ok)	3.1.4 Connect acknowledge
Q931_COMPLETE_IND	Setup complete indication (error)	3.1.5 Disconnect /Note 1
Q931_STATUS_IND	Status indication	3.1.16 Status
Q931_SUSPEND_REQ	Suspend request	3.1.18 Suspend
Q931_SUSPEND_CON	Suspend confirm (ok)	3.1.19 Suspend acknowledge
Q931_SUSPEND_CON	Suspend confirm (error)	3.1.20 Suspend reject
Q931_TIMEOUT_IND	Timeout indication	3.1.5 Disconnect /Note 1
Q931_USERINFO_REQ	N.A.	3.3.13 User information

Command	Corresponding ITU Q.931 call control primitive	ITU Q.931 message format reference
Q931_USERINFO_IND	N.A.	3.3.13 User information

Note 1: Only mandatory fields are used

Table 49: Q.931/Euro-ISDN Commands

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

Note 1: 'NN' in Value encoded according to Table 15.
Note 2: X : Exists
- : Does not exist

Table 50: Q.931/Euro-ISDN System Management Commands

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

Note 1: 'NN' in **Value** encoded according to Table 15.
Note 2: X: Exists
- : Does not exist

Table 51: Q.931/Euro-ISDN General Command Structure

Offset	Field	Size	Value	Reference
0	*bCommand*	1	Number	Command according to Table 49
1	*iCallReference*	2 to N	Info element	Q.931 Chapter 4.3

Offset	Field	Size	Value	Reference
1+N	*iInformationElement*	Size of info element	Info element	Optional information element according to command.

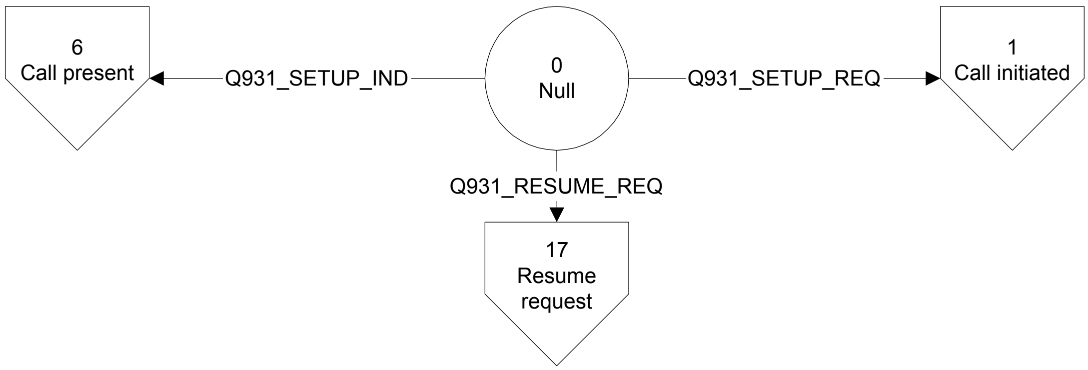


Figure 7: Q.931 Handling of Null State

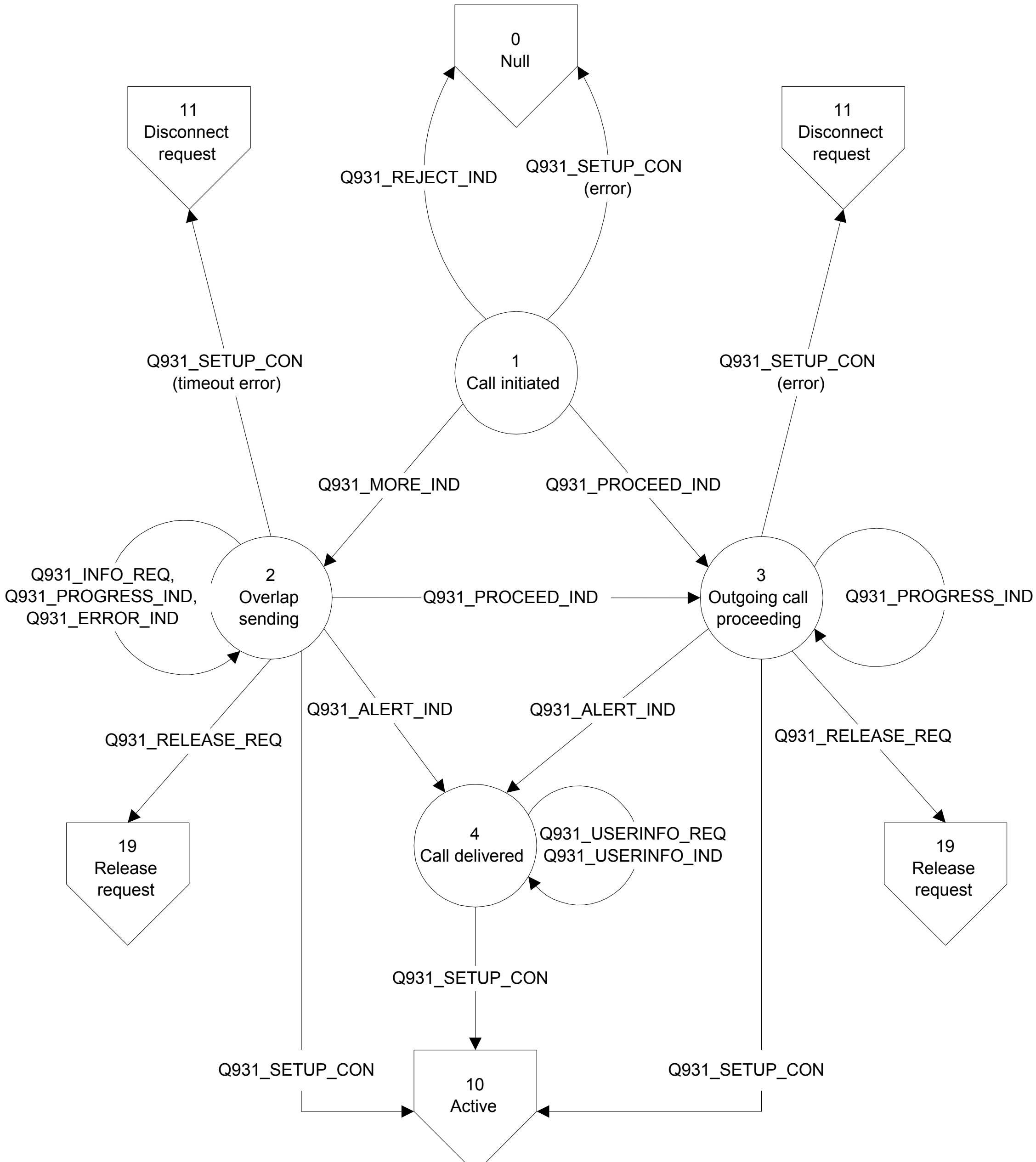


Figure 8: Q.931 Handling of Outgoing-Call States

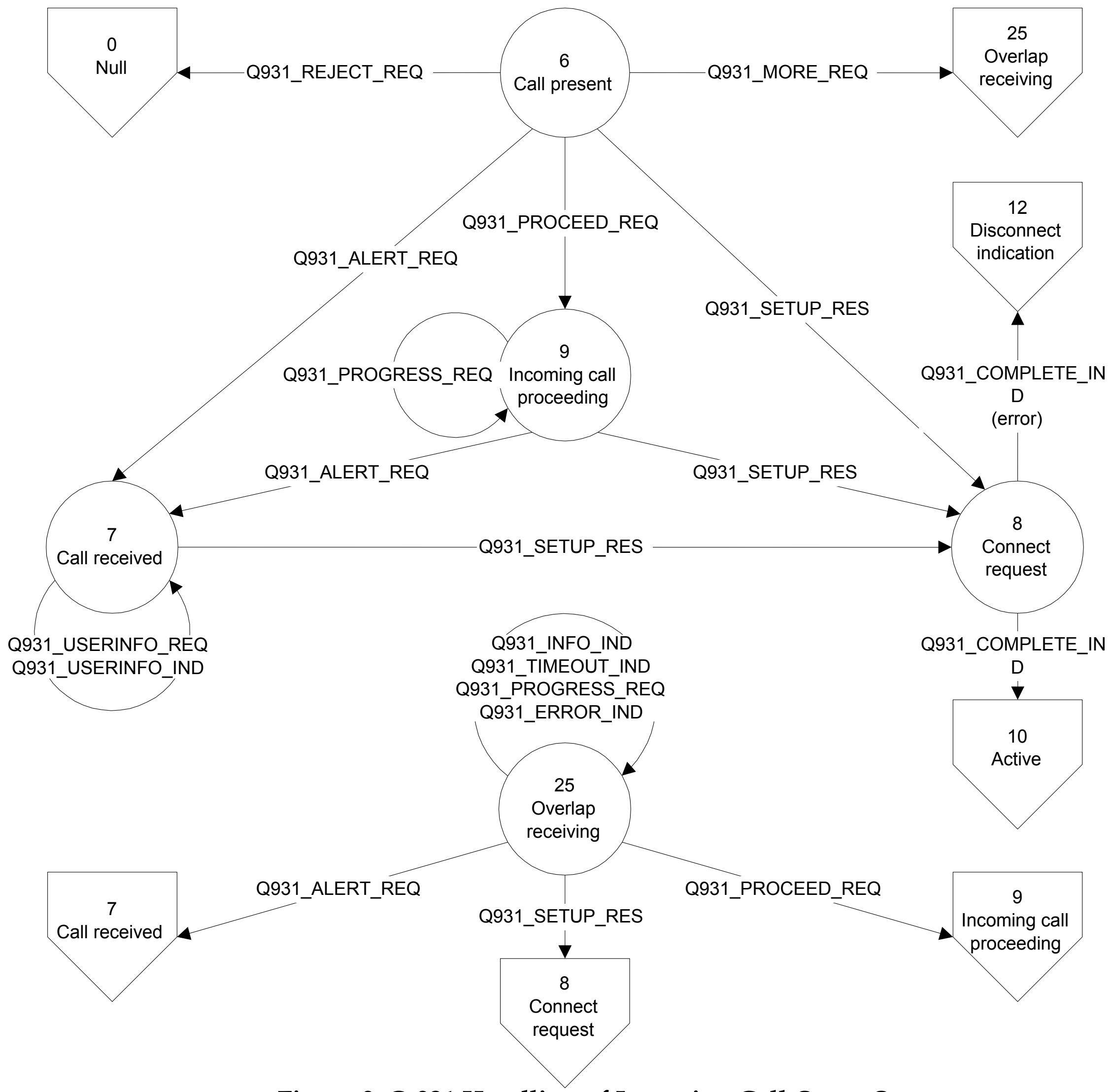


Figure 9: Q.931 Handling of Incoming Call-Setup States

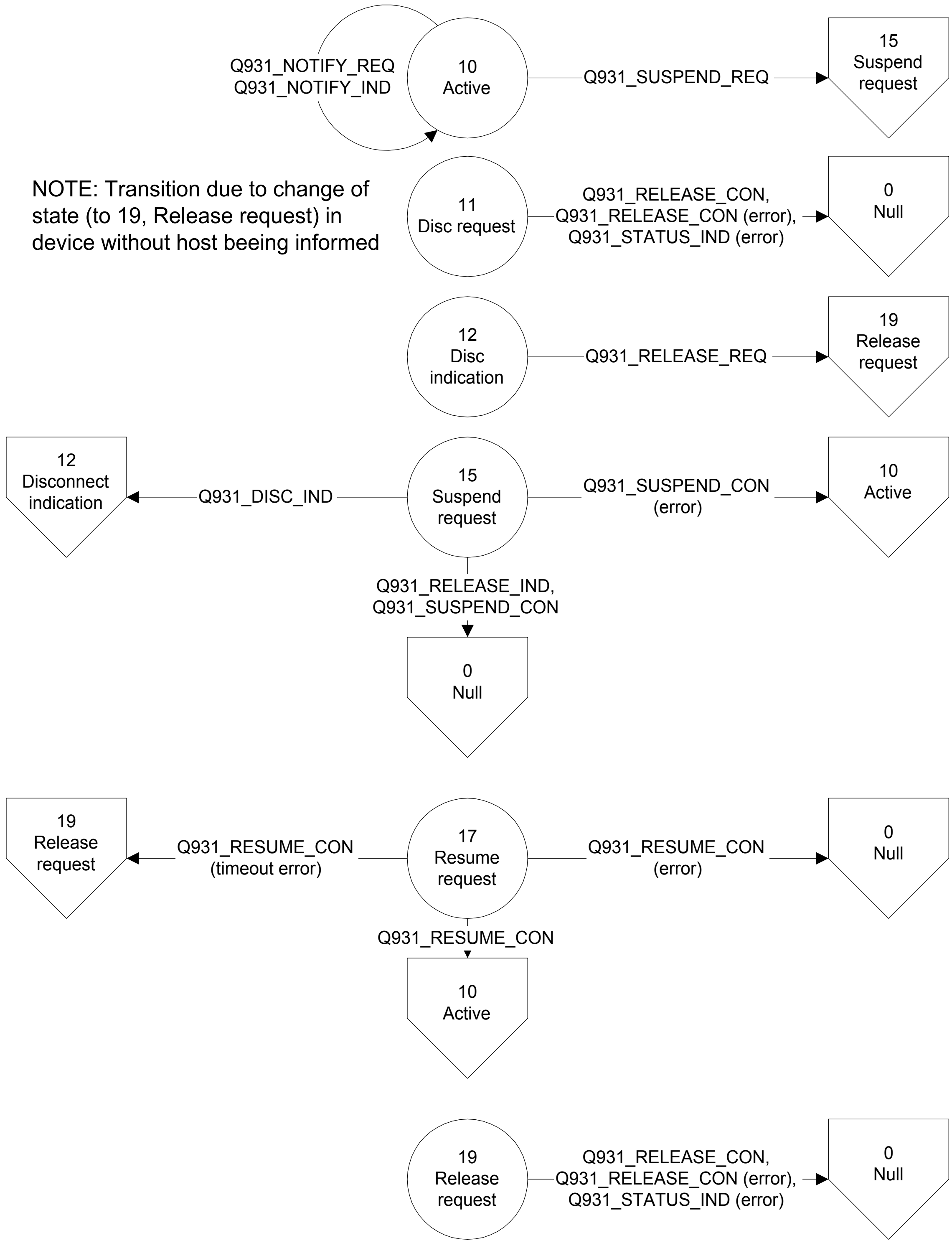


Figure 10: Q.931 Handling of Specific States

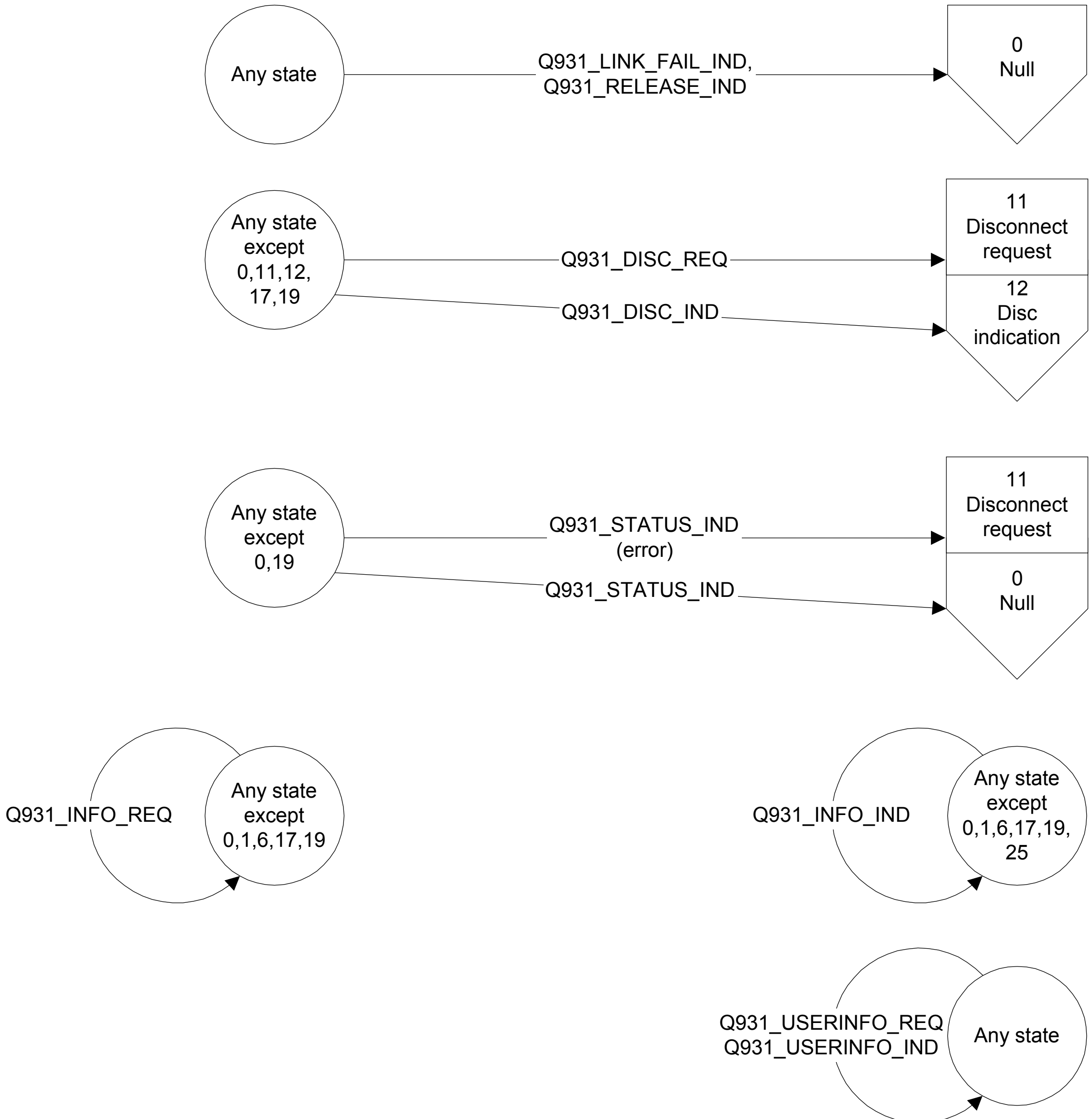


Figure 11: Q.931 Handling of Generic States

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

Protocol code: According to Table 3

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

Table 52: V.42*bis* Configuration Parameter List

bParameterIndex	Field	Size	Value	Description

bParameterIndex	Field	Size	Value	Description
0	*bP0*	1	Number	V.42bis data compression request
1	*wP1*	2	Number	Number of code words
2	*bP2*	1	Number	Maximum string size

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

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

Protocol code: According to Table 3.

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