Communications Device Interface Specification

Communications Device Interface Specification

UNIX International
OSI Work Group
Revision: 2.0.0
August 20, 1991

Version 0.9.2 Edition 7
Updated 2008-10-31
Distributed with Package strxns-0.9.2.7

Brian Bidulock <bidulock@openss7.org> for
The OpenSS7 Project <http://www.openss7.org/>

Published by:

OpenSS7 Corporation
1469 Jefferys Crescent
Edmonton, Alberta T6L 6T1
Canada

for further information, contact:
OpenSS7 Corporation

Phone: +1 780 490 1141
Fax: +1 780 490 1241

Published by:

OpenSS7 Corporation
1469 Jefferys Crescent
Edmonton, Alberta T6L 6T1
Canada

Short Contents

Table of Contents

List of Figures

List of Tables

1 Introduction

This document specifies a STREAMS kernel-level instantiation of the ISO Data Link Service Definition DIS 8886[1] and Logical Link Control DIS 8802/2 (LLC)[2]. Where the two standards do not conform, DIS 8886 prevails.

The Communications Device Interface (CDI) enables a communications device service user to access and use any of a variety of conforming communications device service providers without special knowledge of the provider's protocol. Specifically, the interface is intended to support X.25 LAPB, BX.25 level 2, SDLC, ISDN LAPD, Ethernet(TM), CSMA/CD, FDDI, token ring, token bus, and Bisync. Among the expected communications device service users are implementations of the OSI network data link layer.

The interface specifies access to communications device service providers, and does not define a specific protocol implementation. Thus, issues of network management, protocol performance, and performance analysis tools are beyond the scope of this document and should be addressed by specific implementations of a communications device provider. However, accompanying each provider implementation should be information that describes the protocol-specific behaviour of that provider. Currently, there are plans to come up with a set of implementor's agreements/guidelines for common communications device providers. These agreements will address issues such as CDSAP address space, subsequent address, PPA access and control, QOS, supported services, etc.

This specification assumes the reader is familiar with OSI Reference Model[4] terminology, OSI Data Link Services, and STREAMS.

1.1 Document Organization

This specification is organized as follows:

- Chapter 2 [Model of the Communications Device Layer], page 5, presents background on the structure of the communications device layer of the OSI Reference Model, and explains the intended architecture in the STREAMS environment. Communications Device addressing concepts are also presented.
- Chapter 3 [CDI Services], page 11, presents an overview of the services provided by CDI.
- Chapter 4 [CDI Primitives], page 29, describes the detailed syntax and semantics of each CDI primitive that crosses the communications device interface.
- Chapter 5 [Allowable Sequence of CDI Primitives], page 91, describes the allowable sequence of CDI primitives that may be issued across the interface.
- Chapter 6 [Precedence of CDI Primitives], page 93, presents a summary of the precedence of CDI primitives as they are queued by the CDS provider and/or CDS user.
- Appendix A [Guidelines for Protocol Independent CDS Users], page 95, summarizes guidelines a CDS user implementation must follow to be fully protocol-independent.

[1] International Organization for Standardization, "Data Link Service Definition for Open Systems Interconnection," DIS 8886, February 1987.

[2] International Organization for Standardization, "Logical Link Control," DIS 8802/2, 1985.

- Appendix B [Required Information for CDS Provider-Specific Addenda], page 97, presents the information that should be documented for each CDS provider implementation.
- Appendix C [CDI Header File], page 99, presents the header file containing CDI structure and constant definitions needed by a CDS user or provider implemented to use the interface.
- Appendix D [Glossary of CDI Terms and Acronyms], page 109, presents a Glossary of CDI Terms and Acronyms.

2 Model of the Communications Device Layer

The communications device layer (layer 1 in the OSI Reference Model) is responsible for the transmission and error-free delivery of bits of information over a physical communications medium.

The model of the communications device layer is presented here to describe concepts that are used throughout the specification of CDI. It is described in terms of an interface architecture, as well as addressing concepts needed to identify different components of that architecture. The description of the model assumes familiarity with the OSI Reference Model.

2.1 Model of the Service Interface

Each layer of the OSI Reference Model has two standards:

- one that defines the services provided by the layer, and
- one that defines the protocol through which layer services are provided.

CDI is an implementation of the first type of standard. It specifies an interface to the services of the communications device layer. Figure 2.1 depicts the abstract view of CDI.

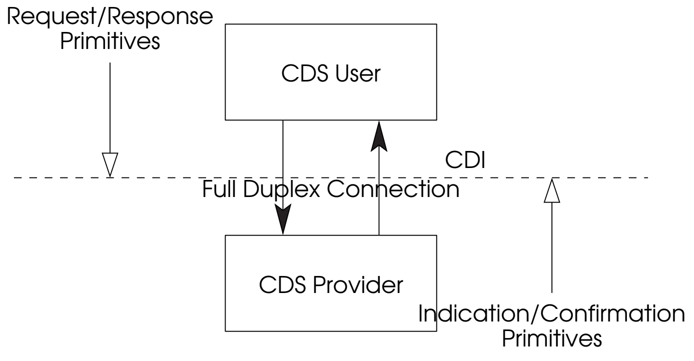

Figure 2.1: *Abstact View of CDI*

The communications device interface is the boundary between the data link and physical layers of the OSI Reference Model. The data link layer entity is the user of the services of the communications device interface (CDS user), and the communications device entity is the provider of those services (CDS provider). This interface consists of a set of primitives that provide access to the communications device layer services, plush the rules for using those primitives (state transition rules). A communications device interface service primitive might request a particular service or indicate a pending event.

To provide uniformity among the various UNIX system networking products, an effort is underway to develop service interfaces that map to the OSI Reference Model. A set of kernel-level interfaces, based on the STREAMS development environment, constitute a major portion of this effort. The service primitives that make up these interfaces are defined as STREAMS messages that are transferred between the user and provider of the service. CDI is one such kernel-level interface, and is targeted for STREAMS protocol modules that either use or provide communications device services. Also, user programs that wish to access a STREAMS-based communications device provider directly may do so using the `putmsg(2)` and `getmsg(2)` system calls.

Referring to the abstract view of CDI (Figure 2.1), the CDS provider is configured as a STREAMS driver, and the CDS user accesses the provider using `open(2)` to establish a stream to the CDS provider. The stream acts as a communication endpoint between a CDS user and the CDS provider. After the stream is created, the CDS user and CDS provider communicate via tht messages presented later in this specification.

CDI is intended to free communications device users from specific knowledge of the characteristics of the communications device provider. Specifically, the definition of CDI hopes to achieve the goal of allowing a CDS user to be implemented independent of a specific communications medium. Any communications device provider (supporting any communications medium) that conforms to the CDI specification may be substituted beneath the CDS user to provide communications device services. Support of a new CDS provider should not require any changes to the implementation of the CDS user.

2.2 Modes of Communication

The communications device interface supports full-duplex and half-duplex communications on a medium.

For half-duplex communications, either the input section or the output section can be active at any point in time, but not both. For full-duplex communications, both the input section and the output section are both active or both inactive at any point in time. A particular CDS provider for a half-duplex device can give the appearance of a full-duplex device for the purposes of the communications device interface presented by the CDS provider. The communications device interface provides a specialized set of services for half-duplex communications.

The communications device interface supports three output styles: unacknowledged output, acknowledged output and paced output.

Unacknowleged output is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Because there is no acknowledgement of each data unit transmission, this output style can be unreliable in the most general case. However, a specific CDS provider can provide assurance that messages will not be lost, duplicated or reordered.

Acknowledged output is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Because there is acknowledgement of each data unit transmission, this output style can be reliable in the most general case. Specific CDS providers can provide assurance that messages will not be lost, duplicated or reordered.

Paced output is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Because there is no peer acknowledgement of each data unit transmission, this output style can be unreliable in the most general case.

However, a specific CDS provider can provide assurance that messages will not be lost, duplicated or reordered. Acknowledgements are used to pacing output, and are typically issued once the data unit has been transmittted on the medium.

The communications device interface supports three modes of communication: connection, connectionless and acknowledged connectionless. The connection mode is circuit-oriented and enables data to be transferred over a pre-established connection in a sequenced manner. Data may be lost or corrupted in this service mode, however, due to provider-initiated resynchronization or connection aborts.

The connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Because there is no acknowledgement of each data unit transmission, this service mode can be unreliable in the most general case. However, a specific CDS provider can provide assurance that messages will not be lost, duplicated, or reordered.

The acknowledged connectionless mode provides the means by which a communications device user can send data and request the return of data at the same time. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The data unit transfer is point-to-point.

2.2.1 Connection-mode Service

The connection-mode service is characterized by four phases of communication: local management, connection establishment, data transfer, and connection release.

2.2.1.1 Local Management

This phase enables a CDS user to initialize a stream for use in communication and establish an identity with the CDS provider.

2.2.1.2 Connection Establishment

This phase enables two CDS users to establish a communications device connection between them to exchange data. One user (the calling CDS user) initiates the connection establishment procedures, while another user (the called CDS user) waits for incoming connect requests. The called CDS user is identified by an address associated with its stream (as will be discussed shortly).

A called CDS user may either accept or deny a request for communications device connection. If the request is accepted, a connection is established between the CDS users and they enter into the data transfer phase. For both the calling and called CDS users, only one connection may be established per stream. Thus, the stream is the communication endpoint for a communications device connection. The called CDS user may choose to accept a connection on the stream where it received the connect request, or it may open a new stream to the CDS provider and accept the connection on this new, responding stream. By accepting the connection on a separate stream, the initial stream can be designated as a listening stream through which all connect requests will be processed. As each request arrives, a new stream (communication endpoint) can be opened to handle the connection, enabling subsequent requests to be queued on a single stream until they can be processed.

2.2.1.3 Data Transfer

In this phase, the CDS users are considered peers and may exchange data simultaneously in both directions over an established communications device connection. Either CDS user may send data to its peer CDS user at any time. Data set by a CDS user is guaranteed to be delivered to the remote user in the order in which it was sent.

2.2.1.4 Connection Release

This phase enables either the CDS user, or the CDS provider, to break an established connection. The release procedure is considered abortive, so any data that has not reached the destination user when the connection is released may be discarded by the CDS provider.

2.2.2 Connectionless-mode Service

The connectionless mode service does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, however, the connectionless data transfer phase is immediately entered. Because there is no established connection, however, the connectionless data transfer phase requires the CDS user to identify the destination of each data unit to be transferred. The destination CDS user is identified by the address associated with that user (as will be discussed shortly).

Connectionless data transfer does not guarantee that data units will be delivered to the destination user in the order in which they were sent. Furthermore, it does not guarantee that a given data unit will reach the destination CDS user, although a given CDS provider may provide assurance that data will not be lost.

2.2.3 Acknowledged Connectionless-mode Service

The acknowledged connectionless mode service also does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, the acknowledged connectionless data transfer phase is immediately entered.

Acknowledged connectionless data transfer guarantees that data units will be delivered to the destination user in the order in which they were sent. A data link user entity can send a data unit to the destination CDS user, request a previously prepared data unit from the destination CDS user, or exchange data units.

2.3 CDI Addressing

Each user of CDI must establish an identity to communicate with other communications device users. The CDS user must identify the physical medium over which it will communicate. This is particularly evident on systems that are attached to multiple physical media. Figure 2.2 illustrates the identification approach, which is explained below.

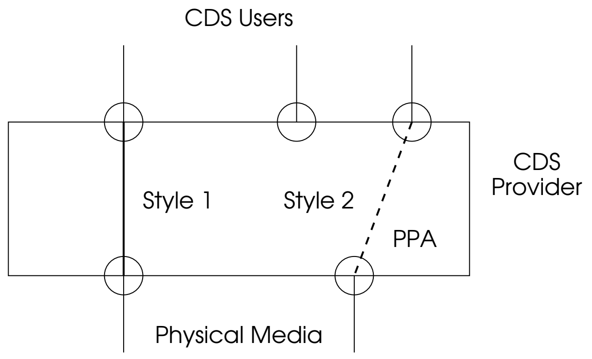

Figure 2.2: *Communications Device Addressing Components*

2.3.1 Physical Attachment Identification

The physical point of attachment (PPA in Figure 2.2) is the point at which a system attaches itself to a physical communications medium (a channel, facility or network interface). All communication on that physical medium funnels through the PPA associated with that physical medium. On systems where a CDS provider supports more than one physical medium, the CDS user must identify which medium it will communicate through. A PPA is identified by a unique PPA identifier.

Unlike the Data Link Provider Interface (DLPI), which also uses the concept of a PPA, CDI does not define a SAP for a CDS user. Once a stream has been associated with a PPA, all messages received on that medium are delivered to the attached CDS user. Only one major/minor device number combination (Stream head) can be associated with a given PPA at any point in time. Attempting to attach a second stream to the same PPA to which another stream is attached will fail.

2.3.2 CDS Provider Styles

Two styles of CDS provider are defined by CDI, distinguished by the way they enable a CDS user to choose a particular PPA.

2.3.2.1 Style 1 CDS Provider

The *Style 1* provider assigns a PPA based on the major/minor device the CDS user opened. One possible implementation of a *Style 1* driver would reserve a major device for each PPA the communications device driver would support. This would allow the STREAMS clone open feature to be used for each PPA configured. This style of provider is appropriate when few PPAs will be supported.

For example, a PCI card that supports two V.35 ports could assign a major device number to the card driver and a minor device number to each of the ports on each card in the system. To establish a stream to a CDS provider for a given port, the minor device number 1 or 2 could be opened for port 1 or 2 on card 1, minor device number 3 or 4 could be opened for port 1 or 2 on card 2, and so on. One major device number for the driver could easily support 127 cards in a system, which is not possible for typical PCI systems and, therefore, is ample.

Style 1 providers do not use the CD_ATTACH_REQ or CD_DETACH_REQ primitives and when freshly opened are in the CD_DISABLED state. That is, as illustrated in Figure 2.2, the *Style 1* CDS provider associates the stream with the PPA during the `open(2)` call.

2.3.2.2 Style 2 CDS Provider

If the number of PPAs a CDS provider will support is large, a *Style 2* provider implementation is more suitable. The *Style 2* provider requires a CDS user to explicitly identify the desired PPA using a special attach service primitive. For a *Style 2* driver, the `open(2)` creates a stream between the CDS user and CDS provider, and the attach primitive then associates a particular PPA with that stream. The format of the PPA identifier is specific to the CDS provider, and should be described in the provider-specific addendum documentation.

The CDS user uses the support primitives (CD_ATTACH_REQ, CD_ENABLE_REQ) to associate a stream with a given Physical Point of Appearance. *Style 2* CDS providers, when freshly opened, are in the CD_UNATTACHED state. That is, the *Style 2* CDS provider does not associated the stream with the PPA during the `open(2)` call, but only later when the CD_ATTACH_REQ primitive is issued by the CDS user.

3 CDI Services

The various features of the CDI interface are defined in terms of the services provided by the CDS provider, and the individual primitives that may flow between the CDS user and CDS provider.

The communications device interface supports two modes of communication (full-duplex and half-duplex) and three output styles (unacknowledged, acknowledged and paced).

The full-duplex mode permits both the input and output sections of the communications device to be active at the same time; whereas, the half-duplex mode only permits either the input or output section of the communications device to be active.

The unacknowledged output style provides no acknowledgement for transmitted data units to the CDS user. This is the typical arrangement for CDS users that are expecting a best-effort delivery of transmitted data units, or that are not concerned about recovery of loss of data units, either because the CDS provider provides a reliable delivery of data units, or because the CDS user expects to provide its own mechanisms should reliable data delivery be required. For example, LLC Type 1 provides just such an unacknowledged delivery of transmitted data units.

The acknowledged output style provides separate acknowledgement of each transmitted data unit. This is the typical arrangement for CDS users that are expecting a reliable delivery of transmitted data units and require acknowledgement of their delivery. For example, LLC Type 2 provides just such an acknowledgement of transmitted data units.

The paced output style provides acknowledgements of transmitted data units, but only as timing hints to the CDS user. This is the typical arrangement where a CDS provider can provide an acknowledgement when the data is actually transmitted on the physical medium and such acknowledgement can be used a timing hints to the CDS user. For example, this is possible with LLC Type 1, where the CDS provider driver implementation has knowledge of when the transmitted data units are emmitted to the medium.

The services are tabulated below and described more fully in the remainder of this section.

Phase	Service	Primitives
Local Management	Information Reporting	CD_INFO_REQ, CD_INFO_ACK, CD_ERROR_ACK
	Attach	CD_ATTACH_REQ, CD_DETACH_REQ, CD_OK_ACK, CD_ERROR_ACK
	Multiplex Name	CD_MUX_NAME_REQ

Table 3.1: *Cross-Reference of CDS Services and Primitives*

Phase	Service	Primitives
Device Management	Enable	CD_ENABLE_REQ, CD_ENABLE_CON, CD_ERROR_IND, CD_ERROR_ACK
	Disable	CD_DISABLE_REQ, CD_DISABLE_CON, CD_ERROR_ACK

Table 3.2: *Cross-Reference of CDS Services and Primitives*

Phase	Service	Primitives
Data Transfer	Unacknowledged	CD_UNITDATA_REQ, CD_UNITDATA_IND
	Acknowledged	CD_UNITDATA_REQ, CD_UNITDATA_IND, CD_UNITDATA_ACK
	Paced	CD_UNITDATA_REQ, CD_UNITDATA_IND, CD_UNITDATA_ACK
Duplex Management	Input Section	CD_READ_REQ, CD_ALLOW_INPUT_REQ, CD_HALT_INPUT_REQ, CD_UNITDATA_IND
	Output Section	CD_ABORT_OUTPUT_REQ, CD_UNITDATA_REQ, CD_UNITDATA_ACK
	Input-Output	CD_WRITE_READ_REQ
Event	Error Reporting	CD_ERROR_IND, CD_BAD_FRAME_IND
	Modem Signals	CD_MODEM_SIG_REQ, CD_MODEM_SIG_IND, CD_MODEM_SIG_POLL

Table 3.3: *Cross-Reference of CDS Services and Primitives*

3.1 Local Management Services

The local management services apply to both full- and half-duplex operation as well as unacknowledged, acknowledged and paced output styles. These services, that fall outside the scope of standards specifications, define the method for intitializing a stream that is connected to a CDS provider. CDS provider information reporting services are also supported by the local management facilities.

3.1.1 Information Reporting Service

This service provides information about the CDI stream to the CDS user.

- **CD_INFO_REQ**: The message CD_INFO_REQ requests the CDS provider to return operating information about the stream.
- **CD_INFO_ACK**: The CDS provider returns the information in a CD_INFO_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the information request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal message sequence is illustrated in Figure 3.1.

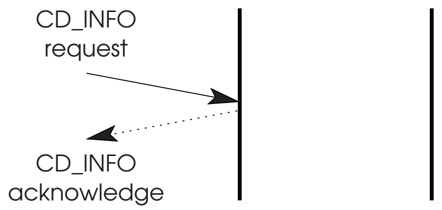

Figure 3.1: *Message Flow: Information Reporting*

In Figure 3.1, the CDS user requests information with a CD_INFO_REQ message and the local CDS provider responds with the requested information in a CD_INFO_ACK message.

3.1.2 Attach Service

The attach service assigns a physical point of attachment (PPA) to a stream. This service is required for *Style 2* CDS providers (see Section 2.3.1 [Physical Attachment Identification], page 9) to specify the physical medium over which communication will occur.

- **CD_ATTACH_REQ**: The CDS user requests the attach service with a CD_ATTACH_REQ message.
- **CD_OK_ACK**: The CDS provider indicates success with a CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider indicates failure with a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal message sequence is illustrated in Figure 3.2.

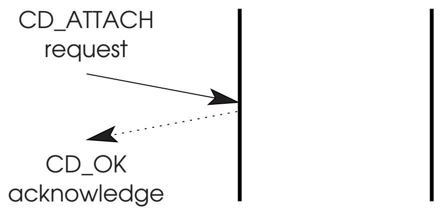

Figure 3.2: *Message Flow: Attaching a stream to a Physical Line*

In Figure 3.2, the CDS user issues a CD_ATTACH_REQ message for a Style 2 CDS provider which results in the association of the stream with the requested PPA and possible enabling of the medium associated with the PPA. The CDS provider acknowledges the attach with a CD_OK_ACK message.

3.1.3 Detach Service

The detach service disassociates a physical point of attachment (PPA) with a stream. This service is required for *Style 2* CDS providers (see Section 2.3.1 [Physical Attachment Identification], page 9) to disassociate the physical medium from the stream over which communication has occurred.

- **CD_DETACH_REQ**: The CDS user request the detach service with a CD_DETACH_REQ message.
- **CD_OK_ACK**: The CDS provider indicates success with a CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider indicates failure with a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal message sequence is illustrated in Figure 3.3.

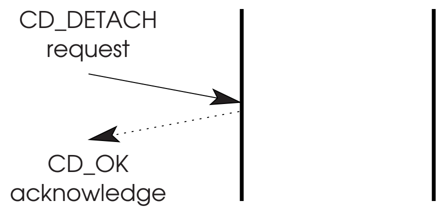

Figure 3.3: *Message Flow: Detaching a Stream from a Physical Line*

In Figure 3.3, the CDS user issues a CD_DETACH_REQ message for a Style 2 CDS provider which results in the disassociation of the stream with the attached PPA and possible disabling of the medium associated with the PPA. The CDS provider acknowledges the detach with a CD_OK_ACK message.

3.1.4 Multiplex Name Service

3.2 Device Management Services

The device management services allow a CDS user to enable or disable a communications device.

3.2.1 Enable Service

The enable service allows a CDS user to enable a communications device. Enabling a communications device may consist of dialling a modem to establish a switched connection, or may consist of simply enabling the communications device attached to a permanent medium.

- **CD_ENABLE_REQ**: The message CD_ENABLE_REQ is used to request that a communications device be enabled and to optionally provide a dial string for a modem.
- **CD_ENABLE_CON**: The CDS provider confirms that the communications device was successfully enabled using a CD_ENABLE_CON message.
- **CD_ERROR_ACK**: The CDS provider indicates a failure to enable the communications device using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal message sequence is illustrated in Figure 3.4.

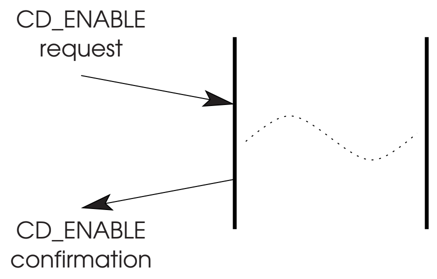

Figure 3.4: *Message Flow: Enabling a Stream*

In Figure 3.4, the CDS user issues a CD_ENABLE_REQ message requesting that the communications device and medium be enabled or connected. Enabling can be soley a local matter, affecting only the local communications device, or can be an end-to-end matter, where the underlying protocol exchanges PDUs necessary to dial, connect or enable the medium.

3.2.2 Disable Service

The disable service allows a CDS user to disable a communications device. Disabling a communications device may consist of disconnecting a modem on a previously established switched connection, or may consist of simply disabling the communications device attached to a permanent medium.

- **CD_DISABLE_REQ**: The CD_DISABLE_REQ message is used to request that a communications device be disabled and to optionally provide for the disposition of unsent data units.
- **CD_DISABLE_CON**: The CDS provider confirms that the communications device was successfully disable, and that unsent data units were properly disposed of, using a CD_DISABLE_CON message.
- **CD_ERROR_IND**: The CDS provider indicats a failure of the communications device resulting in it being disabled locally using a CD_ERROR_IND message. (The CD_ERROR_IND message normally has an error number indicating that the communications device was disconnected, i.e. CD_DISC.)
- **CD_ERROR_ACK**: The CDS provider indicates a failure to disable the communications device using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal message sequence is illustrated in Figure 3.5.

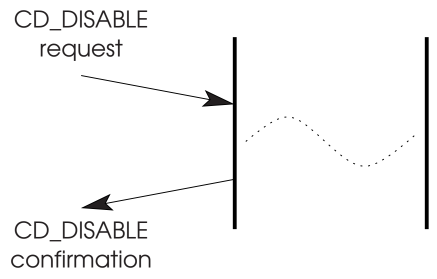

Figure 3.5: *Message Flow: Disabling a Stream*

In Figure 3.5, the CDS user issues a CD_DISABLE_REQ message requesting that the communications device and medium be disabled or disconnected. Disabling can be soley a local matter, affecting only the local communications device, or can be an end-to-end matter, where the underlying protocol exchanges PDUs necessary to disconnect or disable the medium.

3.3 Data Transfer Services

Data transfer services provide for the transfer of data between CDS users on a communications device. There are three output styles for data transfer: unacknowledged, acknowledged and paced. In all modes, data is transferred in self-contained units and there is not necessarily any relationship between independent units of data. Data can be trasnferred both in a connectionless sense, in that addresses are associated with the data transfer, or in a connection-mode sense, in that no addresses are associated with the data transfer. In all output styles, the receiving CDS user is selected, not with addresses, but by selecting the communications device stream upon which the data is transmitted. The receiving CDS user is implied: it is the CDS user that is at the other end of the communications device medium as selected by the PPA. Addresses and priorities associated with the user data are for use by the receiving CDS user in de-multiplexing the data within the CDS user. The CDS provider does not de-multiplex data and any data received on the communications devices associated with a physical point of appearance are delivered to the CDS user that is attached and enabled for that communications device.

3.3.1 Unacknowledged Data Transfer Service

Unacknowledged data transfer service provides for the transfer of data between CDS users without acknowledgement. In the general case, this is an unreliable data transfer. However, the CDS provide may provide assurances with regard to the loss, duplication and reordering of data.

- **CD_UNITDATA_REQ**: The sending CDS user transfers data to the receiving CDS user with the CD_UNITDATA_REQ message.
- **CD_UNITDATA_IND**: Upon receiving user data, the CDS provider indicates the received data to the local CDS user with the CD_UNITDATA_IND message.
- **CD_ERROR_IND**: If the local CDS provider is unable to transmit CDS user data

requested in a CD_UNITDATA_REQ message, it responds to the local sending CDS user with a CD_ERROR_IND message.

- **CD_BAD_FRAME_IND**: If the local CDS provider is unable to receive CDS user data correctly, it is indicated to the local receiving CDS user with the CD_BAD_FRAME_IND message.

The normal sequence of primitives for a successful unacknowledged transmission and reception is illustrated in Figure 3.6.

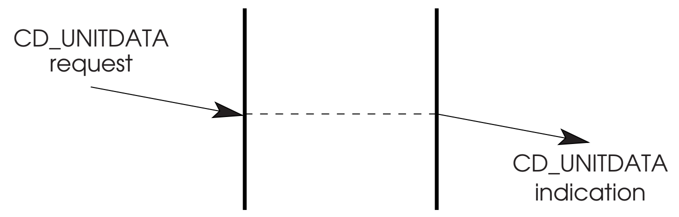

Figure 3.6: *Message Flow: Successful Unacknowleged Data Transfer*

The normal sequence of primitives for an unsuccessful unacknowledged transmission is illustrated in Figure 3.7.

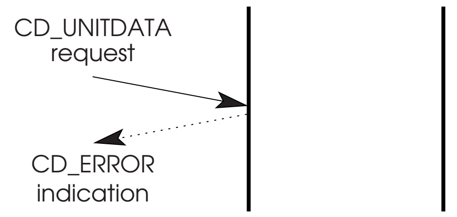

Figure 3.7: *Message Flow: Unsuccessful Unacknowleged Data Transmission*

The normal sequence of primitives for an unsuccessful unacknowledged reception is illustrated in Figure 3.8.

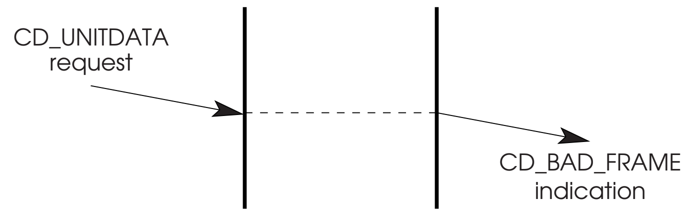

Figure 3.8: *Message Flow: Unsuccessful Unacknowleged Data Reception*

In Figure 3.6, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the receiving side indicates the data in a CD_UNITDATA_IND message. No acknowledgments or receipt confirmation is indicated at the sending CDS provider, regardless of whether the underlying protocol supports receipt confirmation.

In Figure 3.7, the CD_UNITDATA_REQ message at the sending CDS user cannot have its data transmitted by the CDS provider due to a transmission error (e.g. the communications medium has disconnected). The CDS provider indicates the transmission error the CDS user with a CD_ERROR_IND message. This is the same in the unacknowledged, acknowledged and paced cases.

In Figure 3.8, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the receiving side detects an error in the transmission (e.g. a CRC error) and indicates an errored frame to the receiving CDS user with a CD_BAD_FRAME_IND message. No error is indicated to the sending CDS user, regardless of whether the underlying protocol supports negative acknowledgements of received data.

3.3.2 Acknowledged Data Transfer Service

Acknowledged data transfer service provides for the acknowledged transfer of data between CDS users. In the general case, this is an unreliable data transfer with indication of loss. However, the CDS provider may provide assurances with regard to the loss, duplication and reordering of data.

The acknowledged data transfer service requires support from the underlying protocol and CDS provider implementation.

- **CD_UNITDATA_REQ**: The sending CDS user transfers data to the receiving CDS user with the CD_UNITDATA_REQ message.
- **CD_UNITDATA_IND**: Upon receiving user data, the CDS provider indicates the received data to the local CDS user with the CD_UNITDATA_IND message.
- **CD_UNITDATA_ACK**: Upon successful receipt acknowledgement, the CDS provider indicates receipt acknowledgement to the local sending CDS user with the CD_UNITDATA_ACK message.
- **CD_ERROR_IND**: If the local CDS provider is unable to transmit CDS user data requested in a CD_UNITDATA_REQ message, or a negative acknowledgement is received by the peer CDS provider, it responds to the local sending CDS user with a CD_ERROR_IND message.
- **CD_BAD_FRAME_IND**: If the local CDS provider is unable to receive CDS user data correctly, it is indicated to the local receiving CDS user with the CD_BAD_FRAME_IND message.

The normal sequence of primitives for a successful acknowledged transmission and reception is illustrated in Figure 3.9.

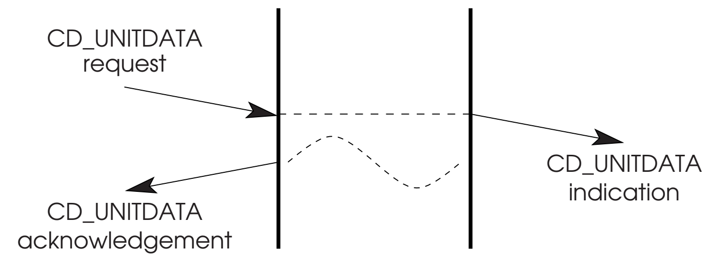

Figure 3.9: *Message Flow: Successful Acknowleged Data Transfer*

The normal sequence of primitives for an unsuccessful acknowledged transmission is illustrated in Figure 3.10.

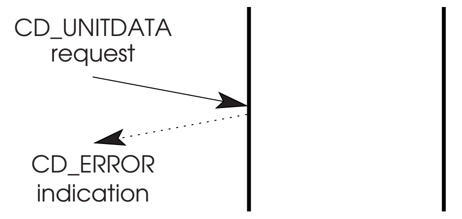

Figure 3.10: *Message Flow: Unsuccessful Acknowleged Data Transmission*

The normal sequence of primitives for an unsuccessful acknowledged reception is illustrated in Figure 3.11.

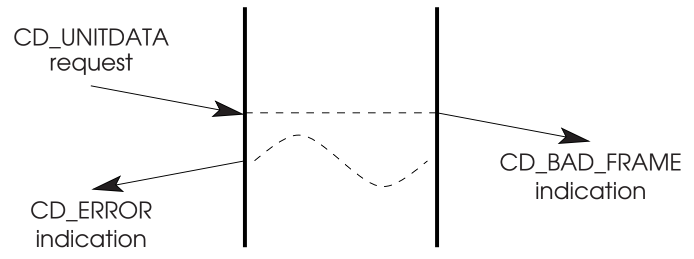

Figure 3.11: *Message Flow: Unsuccessful Acknowleged Data Reception*

In Figure 3.9, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the receiving side indicates the data in a CD_UNITDATA_IND message and provides a positive acknowledgement or receipt confirmation to the sending CDS provider. The sending CDS provider, upon receipt of the positive acknolwedgement or receipt confirmation indicates acknowledgement to the local sending CDS user with the CD_UNITDATA_ACK message.

In Figure 3.10, the CD_UNITDATA_REQ message at the sending CDS user cannot have its data transmitted by the CDS provider due to a transmission error (e.g. the communications medium has disconnected). The CDS provider indicates the transmission error the CDS user with a CD_ERROR_IND message. This is the same in the unacknowledged, acknowledged and paced cases.

In Figure 3.11, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the receiving side detects an error in the transmission (e.g. a CRC error) and indicates an errored frame to the receiving CDS user with a CD_BAD_FRAME_IND message. A negative acknowledgement is sent to the sending CDS provider using the underlying protocol. The sending CDS provider, upon receipt of the negative acknowledgement, indicates the reception error to the sending CDS user with a CD_ERROR_IND message.

3.3.3 Paced Data Transfer Service

Paced data transfer service provides for the paced transfer of data between CDS users. In the general case, this is an unreliable data transfer. Acknowledgements of data transfer only indicate timing hints to the sending CDS user and do not constitute receipt confirmation. However, the CDS provider may provide assurances with regard to loss, duplication and reordering of data.

The paced data transfer service requires support from the sending CDS provider.

- **CD_UNITDATA_REQ**: The sending CDS user transfers data to the receiving CDS user with the CD_UNITDATA_REQ message.
- **CD_UNITDATA_ACK**: Upon successful *transmission* of the user data, the CDS provider acknowledges the data transmission to the local sending CDS user with the CD_UNITDATA_ACK message.
- **CD_UNITDATA_IND**: Upon receiving user data, the CDS provider indicates the received data to the local CDS user with the CD_UNITDATA_IND message.
- **CD_ERROR_IND**: If the local CDS provider is unable to transmit CDS user data requested in a CD_UNITDATA_REQ message, it responds to the local sending CDS user with a CD_ERROR_IND message.
- **CD_BAD_FRAME_IND**: If the local CDS provider is unable to receive CDS user data correctly, it is indicated to the local receiving CDS user with the CD_BAD_FRAME_IND message.

The normal sequence of primitives for a successful paced transmission and reception is illustrated in Figure 3.12.

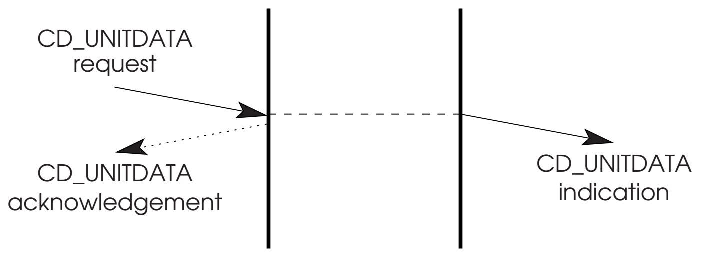

Figure 3.12: *Message Flow: Successful Paced Data Transfer*

The normal sequence of primitives for an unsuccessful paced transmission is illustrated in Figure 3.13.

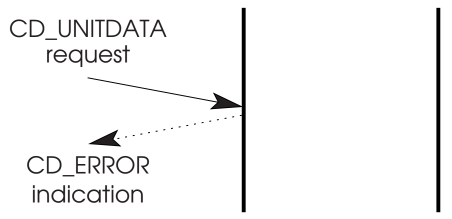

Figure 3.13: *Message Flow: Unsuccessful Paced Data Transmission*

The normal sequence of primitives for an unsuccessful paced reception is illustrated in Figure 3.14.

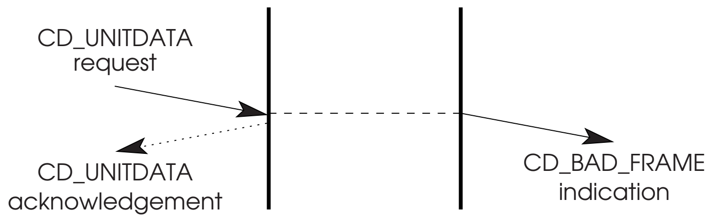

Figure 3.14: *Message Flow: Unsuccessful Paced Data Reception*

In Figure 3.12, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the sending side, once the data has been transmitted, or using some other timing que, issues an acknowledgement of the transmission to the local CDS user with the CD_UNITDATA_ACK message. The CDS provider at the receiving side indicates the data in a CD_UNITDATA_IND message. No receipt confirmation is indicated at the sending CDS provider, regardless of whether the underlying protocol supports receipt confirmation.

In Figure 3.13, the CD_UNITDATA_REQ message at the sending CDS user cannot have its data transmitted by the CDS provider due to a transmission error (e.g. the communications medium has disconnected). The CDS provider indicates the transmission error the CDS user with a CD_ERROR_IND message. This is the same in the unacknowledged, acknowledged and paced cases.

In Figure 3.14, the CD_UNITDATA_REQ message at the sending CDS user results in data transfer to the receiving CDS user. The CDS provider at the sending side, once the data has been transmitted, or using some other timing que, issues an acknowledgement of the transmission to the local CDS user with the CD_UNITDATA_ACK message. The CDS provider at the receiving side detects an error in the transmission (e.g. a CRC error) and indicates an errored frame to the receiving CDS user with a CD_BAD_FRAME_IND message. No negative acknowledgement is indicated to the sending CDS user, regardless of whether the underlying protocol supports negative acknowledgements.

3.4 Duplex Management Services

Duplex management services allow fine-grained control of the half-duplex mechanism. These services logically distinguish between the input section of the communications device and the output section of the communications device. The input section can be enabled (disabling the output section on half-duplex devices) and disabled (enabling the output section). The output section can have output aborted. And output-input operations are also possible where data units are transmitted and then a response is awaited.

These duplex management services are only necessary or CDS providers that expose the activation and deactivation of the input and output sections to the CDS user. CDS providers that control half-duplex communications devices, but which do not expose the half-duplex nature to the CDS user, can use the normal data transfer services used for full-duplex devices. The CDS user can determine the style of the CDS provider using the information reporting service (see Section 3.1.1 [Information Reporting Service], page 12).

3.4.1 Input Section Service

Input section services control the activation of the input section (and resulting deactivation of the output section). The read service provides the ability to activate the input section and await data or the expiry of a time interval for which to wait. The input allow and halt services provide the ability to permanently active or deactivate the input section.

3.4.1.1 Read Service

The read service is for half-duplex operation and temporarily enables the input section until data has been received, or until an time interval has passed, whichever comes first.

- **CD_READ_REQ**: The CDS user requests the read service using the CD_READ_REQ message. This message also specifies the period of time to await input data before failing with a CD_READTIMEOUT error.
- **CD_OK_ACK**: The CDS provider acknowledges successful receipt of the CD_READ_REQ message using the CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the read request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

- **CD_UNITDATA_IND**: If data is available to be read, the CDS provider confirms the read request using a CD_UNITDATA_IND message.
- **CD_ERROR_IND**: The CDS provider indicates the failure of the read request (the interval of time has elapsed before data was available to be read) using a CD_ERROR_IND message containing the error CD_READTIMEOUT.

The normal sequence of primitives for a successful read request is illustrated in Figure 3.15.

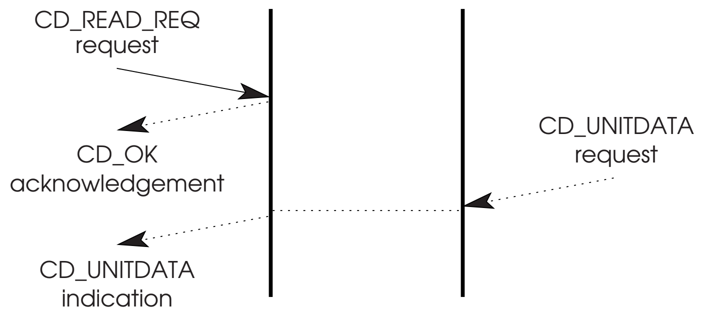

Figure 3.15: *Message Flow: Successful Read Request*

The normal sequence of primitives for an unsuccessful read request is illustrated in Figure 3.16.

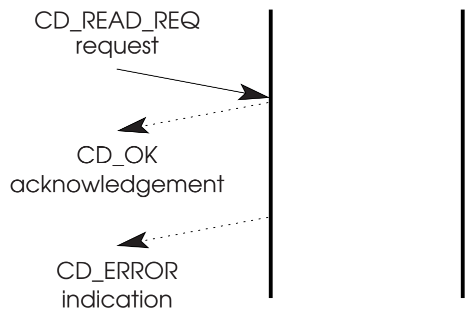

Figure 3.16: *Message Flow: Unsuccessful Read Request*

3.4.1.2 Input Allow Service

The input allow service enables the CDS user to allow the input section (disabling the output section) until further notice. The allow input service is typically used with the halt input service (see Section 3.4.1.3 [Input Halt Service], page 24).

- **CD_ALLOW_INPUT_REQ**: The CDS user requests that the input section be allowed using the CD_ALLOW_INPUT_REQ message.
- **CD_OK_ACK**: The CDS provide acknowledges successful receipt of the message using the CD_OK_ACK message.

- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the allow input request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal sequence of messages is illustrated in Figure 3.17.

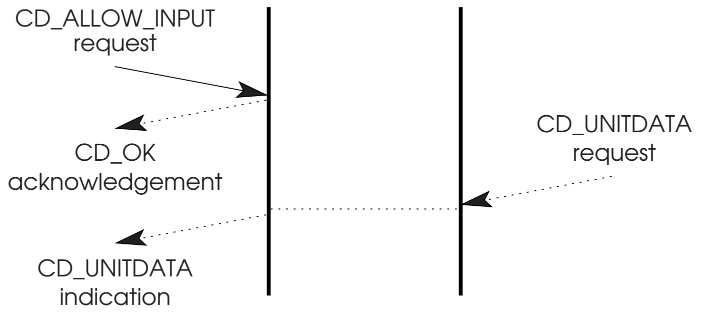

Figure 3.17: *Message Flow: Allow Input*

3.4.1.3 Input Halt Service

The input halt service enables the CDS user to halt the input section (enabling the output section) until further notice. The halt input service is typically used following the allow input service (see Section 3.4.1.2 [Input Allow Service], page 23).

- **CD_HALT_INPUT_REQ**: The CDS user request that the input section be halted using the CD_HALT_INPUT_REQ message.
- **CD_OK_ACK**: The CDS provide acknowledges successful receipt of the message using the CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the allow input request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal sequence of primitives for a successful halt input request is illustrated in Figure 3.18.

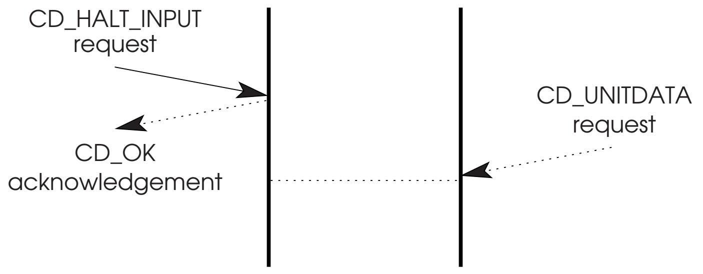

Figure 3.18: *Message Flow: Halt Input*

3.4.2 Output Section Service

The output section can be controlled using the abort service. The output abort service provides the ability for the CDS user to abort any output currently being transmitted by the communications device.

- **CD_ABORT_OUTPUT_REQ**: The CDS user request that output be aborted using the CD_ABORT_OUTPUT_REQ message.
- **CD_OK_ACK**: The CDS provide acknowledges successful receipt of the message using the CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the allow input request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

The normal sequence of primitives for a successful abort output request is illustrated in .

3.4.3 Input-Output Service

A smooth transition from transmission to reception of data units can be accomplished using the write-read service. This service provides the CDS user with the ability to transmit data and then await data reception. The service is like a unit data request service followed by a read service.

- **CD_WRITE_READ_REQ**: The CDS user request that a write read request be performed using the CD_WRITE_READ_REQ message.
- **CD_OK_ACK**: The CDS provide acknowledges successful receipt of the write read request using the CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the write read request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.
- **CD_UNITDATA_IND**: If data is available to be read, the CDS provider confirms the write read request using a CD_UNITDATA_IND message.
- **CD_ERROR_IND**: The CDS provider indicates the failure of the write read request (the interval of time has elapsed before data was available to be read) using a CD_ERROR_IND message containing the error CD_READTIMEOUT.

The normal sequence of primitives for a successful write read service request is illustrated in Figure 3.19.

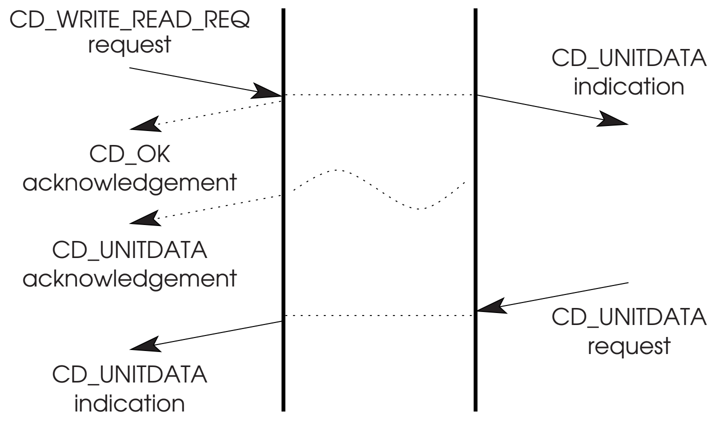

Figure 3.19: *Message Flow: Successful Write Read Request*

The normal sequence of primitives for an unsuccessful write read service request is illustrated in Figure 3.20.

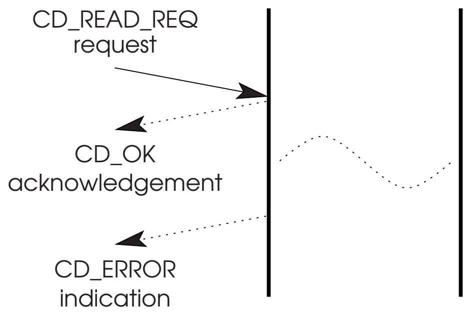

Figure 3.20: *Message Flow: Unsuccessful Write Read Request*

3.5 Event Services

3.5.1 Error Reporting Service

- **CD_ERROR_IND**:
- **CD_ERROR_ACK**: See Section 3.5.1 [Error Reporting Service], page 26.

3.5.2 Modem Signals Services

3.5.2.1 Assert Modem Signals Service

- **CD_MODEM_SIG_REQ**:
- **CD_OK_ACK**: The CDS provide acknowledges successful receipt of the modem signal request using the CD_OK_ACK message.
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the modem signal re-

quest using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

3.5.2.2 Poll Modem Signals Service

- **CD_MODEM_SIG_POLL_REQ**:
- **CD_MODEM_SIG_IND**:
- **CD_ERROR_ACK**: The CDS provider acknowledges failure for the modem signal poll request using a CD_ERROR_ACK message. See Section 3.5.1 [Error Reporting Service], page 26.

3.6 An Example

To bring it all together, the following example illustrates the primitives that flow during a complete, connection-mode sequence between stream open and stream close.

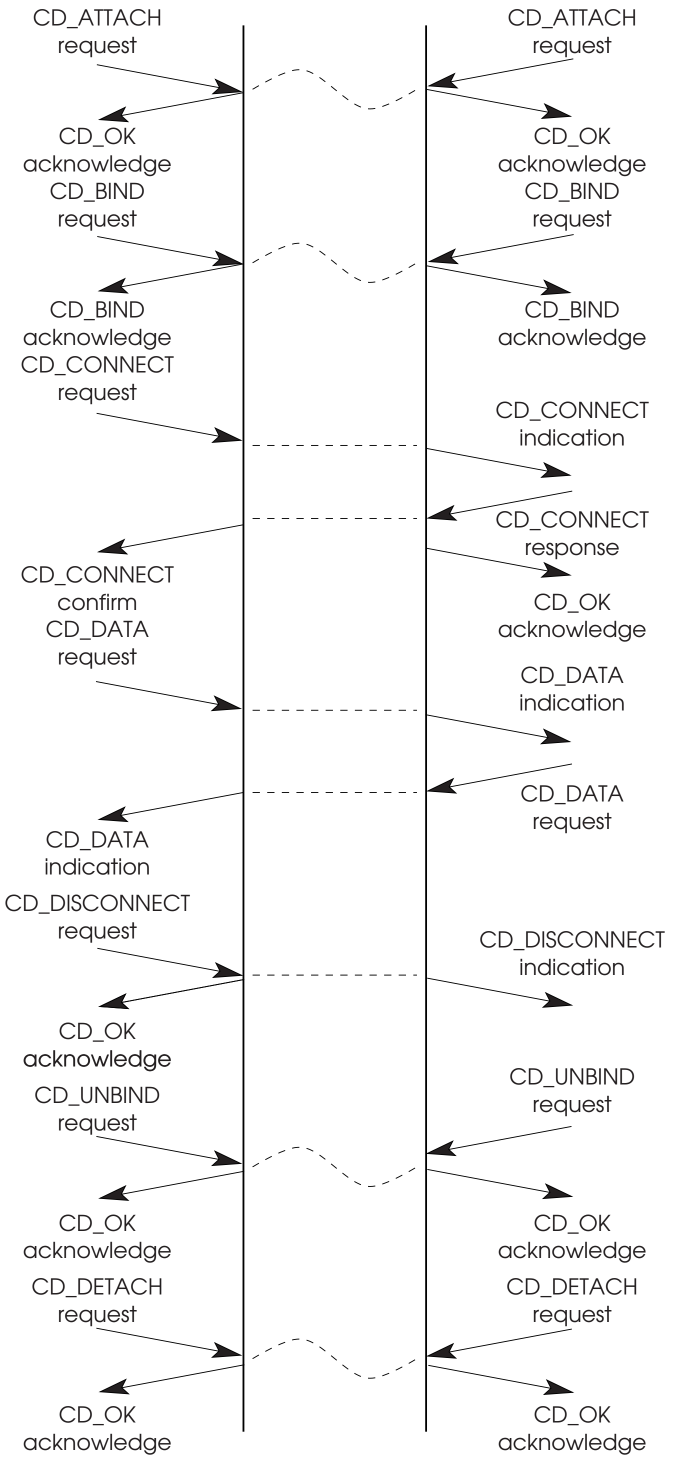

Figure 3.21: *Message Flow: A Connection-mode Example*

4 CDI Primitives

4.1 Local Management Service Primitives

This section describes the local management service primitives that are common to all service modes. These primitives support the Information Reporting, Attach and Acknowledgement services. Once a stream has been opened by a CDS user, these primitive initialize the stream, preparing it for use.

4.1.1 PPA Initialization/De-initialization

The PPA associated with each stream must be initialized before the CDS provider can transfer data over the medium. The initialization and de-initialization of the PPA is a network management issue, but CDI must address the issue because of the impact such actions will have on a CDS user. More specifically, CDI requires the CDS provider to initialize the PPA associated with a stream at some point before it completes the processing of the CD_ENABLE_REQ. Guidelines for initialization and de-initialization of a PPA by a CDS provider are presented here.

4.1.1.1 PPA Initialization

A CDS provide may initialize a PPA using the following methods:

- pre-initialized by some network management mechanism before the CD_ENABLE_REQ primitive is received; or
- automatic initialization on receipt of a CD_ENABLE_REQ or CD_ATTACH_REQ primitive.

A specific CDS provider may support either of these methods, or possibly some combination of the two, but the method implemented has no impact on the CDS user. From the CDS user's viewpoint, the PPA is guaranteed to be initialized on receipt of a CD_ENABLE_CON primitive. For automatic initialization, this implies that the CD_ENABLE_CON primitive may not be issued until the initialization has completed.

If pre-initialization has not been performed and/or automatic initialization fails, the CDS provider will fail the CD_ENABLE_REQ. Two errors, CD_INITFAILED and CD_FATALERR may be returned in the CD_ERROR_ACK primitive in response to a CD_ENABLE_REQ primitive if PPA initialization fails. CD_INITFAILED is returned when a CDS provider supports automatic PPA initialization, but the initialization attempt failed. CD_FATALERR is returned wen the CDS provider requires pre-initialization, but the PPA is not initialized before the CD_ENABLE_REQ is received.

4.1.1.2 PPA De-initialization

A CDS provider may handle PPA de-initialization using the following methods:

- automatic de-initialization upon receipt of the final CD_DETACH_REQ (for *Style 2* providers) or CD_DISABLE_REQ (for *Style 1* providers), or upon closing of the last stream associated with the PPA;

- automatic de-initialization after expiration of a timer following the last CD_DETACH_REQ, CD_DISABLE_REQ, or close as appropriate; or
- no automatic de-initialization; administrative intervention is required to de-initialize the PPA at some point after it is no longer being accessed.

A specific CDS provider may support any of these methods, or possibly some combination of them, but the method implemented has no impact on the CDS user. From the CDS user's viewpoint, the PPA is guaranteed to be initialized and available for transmission until it closes or disables the stream associated with the PPA.

CDS provider-specific addendum documentation should describe the method chosen for PPA initialization and de-initialization.

4.1.2 Message CD_INFO_REQ (cd_info_req_t)

This user originated primitive requests that the provider acknowledge the primitive with a CD_INFO_ACK primitive indicating protocol and option information.

Message Format

This primitive consists fo one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_info_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

State

This primitive is valid in any state other than CD_UNUSABLE where a local acknowledgement is not pending.

New State

The stat is unchanged as a result of the primitive.

Response

This primitive requires the provider to acknowledge receipt of the primitive as follows:

- **Successful:** When successful, the provider acknowledges the primitive with the CD_INFO_ACK.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider acknowledges the primitive with the CD_ERROR_ACK indicating the reason for failure of the primitive.

Reasons for Failure

CD_BADPRIM
Unrecognized primitive.

CD_FATALERR
Device has become unusable.

CD_NOTSUPP
Primitive not supported by device.

CD_OUTSTATE
Primitive was issued from an invalid state.

CD_PROTOSHORT
M_PROTO block too short.

CD_SYSERR
UNIX system error.

4.1.3 Message CD_INFO_ACK (cd_info_ack_t)

This provider originated primitive acknowledges a previously issued CD_INFO_REQ primitive, and provides protocol and limits information for the stream upon which the primitive is issued.

If the stream is in state CD_UNATTACHED, the information returned by CD_INFO_ACK might be different after a successful CD_ATTACH_REQ than it was before the attach was completed. This is because the CD provider might not yet have all protocol information concerning the underlying communications device until after it has been attached to a specific Physical Point of Attachment.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_max_sdu;
    cd_ulong cd_min_sdu;
    cd_ulong cd_class;
    cd_ulong cd_duplex;
    cd_ulong cd_output_style;
    cd_ulong cd_features;
    cd_ulong cd_addr_length;
    cd_ulong cd_ppa_style;
} cd_info_ack_t;
```

Parameters

cd_primitive
Indicates the primitive type.

cd_state Indicates the state of the CDI provider. The *cd_state* can be one of the following values:

CD_UNATTACHED
No Physical Point of Attachment (PPA) is associated with the stream. Only Style 2 communications devices (streams that return CD_STYLE2 in the *cd_ppa_style* field) can exist in this state. CD_STYLE2 communication devices start in this state after `open(2)`.

CD_UNUSABLE
PPA cannot be used.

CD_DISABLED
A Physical Point of Attachment (PPA) is associated with the stream, but the communications device is disabled. Style 1

communications devices (streams that return CD_STYLE1 in the *cd_ppa_style* field) start in this state after `open(2)`.

CD_ENABLE_PENDING
A CD_ENABLE_REQ has been issued and is pending. The provider is waiting for enabling of the communications device to complete before confirmation with CD_ENABLE_CON or error acknowledgement with CD_ERROR_ACK.

CD_ENABLED
The communications device is enabled and is awaiting use. Either the input or output must be active or allowed before data can be transferred.

CD_READ_ACTIVE
The input section is temporarily enabled and will be disabled after data arrives.

CD_INPUT_ALLOWED
The input section is permanently enabled.

CD_DISABLE_PENDING
A CD_DISABLE_REQ has been issued and is pending. The provider is waiting for disabling of the communications device to complete before confirmation with CD_DISABLE_CON or error acknowledgement with CD_ERROR_ACK.

CD_OUTPUT_ACTIVE
Output section active only.

CD_XRAY
X-raying another PPA.

cd_max_sdu
The maximum size of the Signalling Data Unit (SDU) in octets.

cd_min_sdu
The minimum size of the Signalling Data Unit (SDU) in octets.

cd_class Indicates the class of the communications device. *cd_class* can be one of the following values:

CD_HDLC
Bit-synchronous.

CD_BISYNC
Character-synchronous.

CD_LAN ISO 8802-3,4,5 local-area network MAC.

CD_NODEV
No device, PPA used for X-ray.

cd_duplex Indicates full or half duplex operation. *cd_duplex* can be one of the following values:

CD_FULLDUPLEX
Full duplex; allow input supported.

CD_HALFDUPLEX
Half duplex; read write/read supported.

cd_output_style
Indicates the output style. *cd_output_style* can be one of the following values:

CD_UNACKEDOUTPUT
The communications device does not issue CD_UNITDATA_ACK primitives.

CD_ACKEDOUTPUT
The communications device issues CD_UNITDATA_ACK primitives in acknowledgement of CD_UNITDATA_REQ primitives.

CD_PACEDOUTPUT
The communications device issues CD_UNITDATA_ACK primitives only as output timing hints.

cd_features
Indicates the features supported by the communications device. *cd_features* can be a bitwise OR of the following flags:

CD_CANREAD
Read request supported on full duplex.

CD_CANDIAL
Dial information supported.

CD_AUTOALLOW
CD_INPUT_ALLOWED as soon as enabled.

CD_KEEPALIVE
Do not send off at CD_DISABLE_REQ. This is a Gcom extension.

cd_addr_length
The maximum size of an address for use with CD_UNITDATA_REQ, CD_UNITDATA_IND.

cd_ppa_style
Indicates the Physical Point of Attachment (PPA) style. *cd_ppa_style* can be one of the following values:

CD_STYLE1
The communications device is already attached to the physical point of appearance at `open(2)`. The device starts in the CD_DISABLED state.

CD_STYLE2
: The communications device is not attached to the physical point of appearance at `open(2)`, and must be attached with CD_ATTACH_REQ. The device starts in the CD_UNATTACHED state.

State

This primitive is valid in any state where a local acknowledgement (requiring response with a CD_OK_ACK) is not pending, and only in response to a CD_INFO_REQ primitive.

New State

The new state is unchanged.

4.1.4 Message CD_ATTACH_REQ (cd_attach_req_t)

This user originated primitive requests that the requesting stream be attached to the physical device indicated by the Physical Point of Attachment (PPA).

When a Style 2 CDI stream is first opened, it is opened in the CD_UNATTACHED state and is not associated with a Physical Point of Appearance (PPA). The CD_ATTACH_REQ primitive requests that the provider associate the stream with the specified PPA and move the stream to the CD_DISABLED state.

Style 1 CDI streams open in the CD_DISABLED state, and a CD_ATTACH_REQ primitive issued on a Style 1 stream will fail.

This primitive is only valid for devices that return CD_STYLE2 in the *cd_ppa_style* field in a CD_INFO_ACK.

Addressing

A Physical Point of Appearance corresponds to the hardware interface associated with a specific communications device. A PPA number is associated with each hardware interface for a specific provider or device. PPA numbers are a *cd_ulong*, but which PPA number corresponds to which Physical Point of Appearance is a provider-specific configuration matter. Specific providers should document the mapping of PPA numbers to actual Physical Points of Appearance as part of the provider-specific documentation.

Message Format

This primitive consists of on M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_ppa;
} cd_attach_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

cd_ppa Specifies the Physical Point of Attachment (PPA). The format of this field is provider- and device-specific.

State

This primitive is only valid in state CD_UNATTACHED.

New State

The new state is CD_DISABLED.

Response

This primitive requires the provider to return an acknowledgement indicating the success or failure of the CD_ATTACH_REQ.

- **Successful:** When successful, the provider responds with a CD_OK_ACK primitive acknowledging successful processing of the CD_ATTACH_REQ. The new state is CD_DISABLED.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider responds with a CD_ERROR_ACK indicating the non-fatal error. The state is unchanged.

Reasons for Failure

CD_BADPPA
: Invalid PPA identifier.

CD_BADPRIM
: Unrecognized primitive.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

4.1.5 Message CD_DETACH_REQ (cd_detach_req_t)

This user originated primitive requests that the requesting stream be detached from the Physical Point of Attachment (PPA) to which it was previously attached with a successful CD_ATTACH_REQ primitive.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_detach_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

State

This primitive is only valid in state CD_DISABLED.

New State

The new state is CD_UNATTACHED.

Response

This primitive requires the provider to acknowledge the receipt of the CD_DETACH_REQ primitive as follows:

– **Successful:** When the primitive is successful, the provider acknowledges receipt of the primitive with the CD_OK_ACK primitive.
– **Unsuccessful (non-fatal errors):** When unsuccessful, the provider acknowledges receipt of the primitive with the CD_ERROR_ACK primitive indicating the error.

Reasons for Failure

Non-Fata Errors: applicable non-fatal errors are as follows:

CD_BADPRIM
Unrecognized primitive.

CD_EVENT
Protocol-specific event occurred.

CD_FATALERR
Device has become unusable.

CD_NOTSUPP
Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

4.1.6 Message CD_OK_ACK (cd_ok_ack_t)

This provider originated primitive acknowledges that a primitive requiring local acknowledgement with the CD_OK_ACK has been received and successfully processed.

Message Format

This primitive consists of one M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_correct_primitive;
} cd_ok_ack_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Indicates the primitive type.

cd_state
: Indicates the new state of the CD provider following successful processing of the request message that elicited the CD_OK_ACK.

  CD_UNATTACHED
  : No PPA attached.

  CD_UNUSABLE
  : PPA cannot be used.

  CD_DISABLED
  : PPA attached.

  CD_ENABLE_PENDING
  : Waiting acknowledgement of enable request.

  CD_ENABLED
  : Awaiting use.

  CD_READ_ACTIVE
  : Input section enabled; disabled after data arrives.

  CD_INPUT_ALLOWED
  : Input section permanently enabled.

  CD_DISABLE_PENDING
  : Waiting acknowledgement of disable request.

  CD_OUTPUT_ACTIVE
  : Output section active only.

  CD_XRAY
  : X-raying another PPA.

cd_correct_primitive
: Indicates the primitive that was successfully received. *cd_correct_primitive* can be one of the following values:

  CD_ABORT_OUTPUT_REQ
  : abort output.

  CD_ALLOW_INPUT_REQ
  : allow input.

  CD_ATTACH_REQ
  : attach to a physical point of attachment.

  CD_DETACH_REQ
  : detach from a physical point of attachment.

  CD_HALT_INPUT_REQ
  : halt input.

  CD_MODEM_SIG_REQ
  : assert modem signals.

  CD_MUX_NAME_REQ
  : get multiplexer name.

State

This primitive is valid in any state where a local acknowledgement is pending and the primitive is required by the request primitive.

New State

The new state is the state described under the corresponding request primitive.

4.1.7 Message CD_ERROR_ACK (cd_error_ack_t)

This provider originated primitive acknowledges that the previously receive primitive requiring an acknowledgement was received in error. The primitive to which the error acknowledgement applies and the error code are indicated.

Message Format

This primitive consists of one M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_error_primitive;
    cd_ulong cd_errno;
    cd_ulong cd_explanation;
} cd_error_ack_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Indicates the primitive type.

cd_state
: Indicates the current state of the interface. *cd_state* can be one of the following values:

  CD_UNATTACHED
  : No PPA attached.

  CD_UNUSABLE
  : PPA cannot be used.

  CD_DISABLED
  : PPA attached.

  CD_ENABLE_PENDING
  : Waiting acknowledgement of enable request.

  CD_ENABLED
  : Awaiting use.

  CD_READ_ACTIVE
  : Input section enabled; disabled after data arrives.

  CD_INPUT_ALLOWED
  : Input section permanently enabled.

  CD_DISABLE_PENDING
  : Waiting acknowledgement of disable request.

  CD_OUTPUT_ACTIVE
  : Output section active only.

CD_XRAY
: X-raying another PPA.

cd_error_primitive
: Indicates the primitive was received in error. *cd_error_primitive* can be one of the following values:

  CD_ABORT_OUTPUT_REQ
  CD_ALLOW_INPUT_REQ
  CD_ATTACH_REQ
  CD_DETACH_REQ
  CD_DISABLE_REQ
  CD_ENABLE_REQ
  CD_HALT_INPUT_REQ
  CD_INFO_REQ
  CD_MODEM_SIG_REQ
  CD_MUX_NAME_REQ
  CD_READ_REQ
  CD_UNITDATA_REQ
  CD_WRITE_READ_REQ

cd_errno
: Indicates the reason for the error. *cd_errno* can be one of the following values:

  CD_BADADDRESS
  : Address was invalid.

  CD_BADADDRTYPE
  : Invalid address type.

  CD_BADDIAL
  : Dial information was invalid.

  CD_BADDIALTYPE
  : Invalid dial information type.

  CD_BADDISPOSAL
  : Invalid disposal parameter.

  CD_BADFRAME
  : Defective SDU received.

  CD_BADPPA
  : Invalid PPA identifier.

  CD_BADPRIM
  : Unrecognized primitive.

  CD_DISC
  : Disconnected.

  CD_EVENT
  : Protocol-specific event occurred.

  CD_FATALERR
  : Device has become unusable.

CD_INITFAILED
Line initialization failed.

CD_NOTSUPP
Primitive not supported by this device.

CD_OUTSTATE
Primitive was issued from an invalid state.

CD_PROTOSHORT
M_PROTO block too short.

CD_READTIMEOUT
Read request timed out before data arrived.

CD_SYSERR
UNIX system error.

CD_WRITEFAIL
Unit data request failed.

cd_explanation
Indicates a futher explanation of the error. When *cd_errno* is *CD_SYSERR*, this field contains the *UNIX* system error as described in `errno(3)`. Otherwise, *cd_explanation* may contain one of the following values:

CD_CRCERR
CRC or FCS error.

CD_DLE_EOT
DLE EOT detected.

CD_FORMAT
Format error detected.

CD_HDLC_ABORT
Aborted frame detected.

CD_OVERRUN
Input overrun.

CD_TOOSHORT
Frame too short.

CD_INCOMPLETE
Partial frame received.

CD_BUSY
Telephone was busy.

CD_NOANSWER
Connection went unanswered.

CD_CALLREJECT
Connection rejected.

CD_HDLC_IDLE
: HDLC line went idle.

CD_HDLC_NOTIDLE
: HDLC line no longer idle.

CD_QUIESCENT
: Line being reassigned.

CD_RESUMED
: Line has been reassigned.

CD_DSRTIMEOUT
: Did not see DSR in time.

CD_LAN_COLLISIONS
: LAN excessive collisions.

CD_LAN_REFUSED
: LAN message refused.

CD_LAN_NOSTATION
: LAN no such station.

CD_LOSTCTS
: Lost Clear to Send signal.

CD_DEVERR
: Start of device-specific codes.

In addition, when the explanation is *CD_DEVERR* or greater, the explanation may be a device-specific explanation code.

State

This primitive is valid in any state where a local acknowledgement is pending in response to one of the following primitives: CD_ABORT_OUTPUT_REQ, CD_ALLOW_INPUT_REQ, CD_ATTACH_REQ, CD_DETACH_REQ, CD_DISABLE_REQ, CD_ENABLE_REQ, CD_HALT_INPUT_REQ, CD_INFO_REQ, CD_MODEM_SIG_REQ, CD_MUX_NAME_REQ, CD_READ_REQ, CD_UNITDATA_REQ, CD_WRITE_READ_REQ.

New State

The new state remains unchanged from the state in which the request primitive was issued that elicited the error acknowledgement.

4.1.8 Message CD_MUX_NAME_REQ (cd_mux_name_req_t)

This user originated primitive request is not documented.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_mux_name_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

State

Not documented.

New State

Not documented.

Response

- **Successful:** Not documented.
- **Unsucessful (non-fatal errors):** Not documented.

Reasons for Failure

Non-Fatal Errors: Not documented.

CD_BADADDRESS
: Address was invalid.

CD_BADADDRTYPE
: Invalid address type.

CD_BADDIAL
: Dial information was invalid.

CD_BADDIALTYPE
: Invalid dial information type.

CD_BADDISPOSAL
: Invalid disposal parameter.

CD_BADFRAME
: Defective SDU received.

CD_BADPPA
: Invalid PPA identifier.

CD_BADPRIM
: Unrecognized primitive.

CD_DISC
: Disconnected.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_INITFAILED
: Line initialization failed.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_READTIMEOUT
: Read request timed out before data arrived.

CD_SYSERR
: UNIX system error.

CD_WRITEFAIL
: Unit data request failed.

4.2 Device Management Service Primitives

This section describes the service primitives that support the enabling and disabling service of the communications device. These primitives support the Enable and Disable services described earlier.

4.2.1 Message CD_ENABLE_REQ (cd_enable_req_t)

This user originated primitive requests that the communications device be prepared for service and enabled.

A CDI stream that is in the CD_DISABLED stat is not yet ready for transmission. Before the stream can be used for transmission, ti must be successfully enabled with the CD_ENABLE_REQ primitive. Successful processing of the CD_ENABLE_REQ primitive moves the stream to the CD_ENABLED state.

If the communications device returns the CD_CANDIAL flag in the *cd_features* field of the CD_INFO_ACK, the device is capable of dialling and a dial string can be provided, specified by the *cd_dial_length* and *cd_dial_offset* fields. The specification of the dial string is provider- and device-specific.

In the CD_ENABLED state, the stream is able to transmit must have not yet necessarily been allowed for input. If the stream returns CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK, the communications device will be allowed for both transmission and reception upon successful completion of the CD_ENABLE_REQ; however, if the CD_AUTOALLOW flag is not returned, the CD user must first call CD_ALLOW_INPUT_REQ before reception can begin.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_dial_type;
    cd_ulong cd_dial_length;
    cd_ulong cd_dial_offset;
} cd_enable_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive

Specifies the primitive type.

cd_dial_type

Specifies the type of the provided dial string. The type can be set to a provider- or device-specific type, or can be set as follows:

CD_NODIAL

Specifies that there is no dial string associated with the CD_ENABLE_REQ.

cd_dial_length

Specifies the length of the dial string. Specification of dial strings is only allowed when the provider returns CD_CANDIAL in the *cd_features* field of the CD_INFO_ACK. When no dial string is specified by the user, or *cd_dial_type* is set to CD_NODIAL, this field is set to zero (0).

cd_dial_offset
: Specifies the offset of the dial string from the beginning of the M_PROTO message block. When *cd_dial_length* is zero (0), this field is ignored.

State

This primitive is valid in state CD_DISABLED.

New State

The new state is CD_ENABLED for stream that do not return CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK, or the new state is CD_INPUT_ALLOWED for those streams that do return CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK.

Response

This primitive requires that the provider acknowledge the receipt of the primitive as follows:

- **Successful:** Upon success, the provider confirms that the device is enabled with the CD_ENABLE_CON primitive.
- **Unsuccessful (non-fatal errors):** Upon failure, the provider acknowledges the receipt of the primitive with the CD_ERROR_ACK primitive indicating the error.

Reasons for Failure

Non-Fatal Errors: appropriate non-fatal errors are as follows:

CD_BADDIAL
: Dial information was invalid.

CD_BADDIALTYPE
: Invalid dial information type.

CD_BADPRIM
: Unrecognized primitive.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_INITFAILED
: Line initialization failed.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
UNIX system error.

4.2.2 Message CD_ENABLE_CON (cd_enable_con_t)

This provider originated primitive confirms that the previously issued CD_ENABLE_REQ primitive has been successful.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
} cd_enable_con_t;
```

Parameters

cd_primitive
: Indicates the primitive type.

cd_state
: Indicates the state of the CD provider at the time that the primitive was issued. *cd_state* can be one of the following values:

CD_UNATTACHED
: No PPA attached.

CD_UNUSABLE
: PPA cannot be used.

CD_DISABLED
: PPA attached.

CD_ENABLE_PENDING
: Waiting acknowledgement of enable request.

CD_ENABLED
: Awaiting use.

CD_READ_ACTIVE
: Input section enabled; disabled after data arrives.

CD_INPUT_ALLOWED
: Input section permanently enabled.

CD_DISABLE_PENDING
: Waiting acknowledgement of disable request.

CD_OUTPUT_ACTIVE
: Output section active only.

CD_XRAY
: X-raying another PPA.

State

This primitive is issued by the CD provider in the CD_ENABLE_PENDING state.

New State

After issuing this primitive, the CD provider enters the CD_ENABLED state, unless the CD_INFO_ACK returns the CD_AUTOALLOW flag in the *cd_features* field. In that case, the CD provider enters the CD_INPUT_ALLOWED state.

4.2.3 Message CD_DISABLE_REQ (cd_disable_req_t)

This user originated primitive requests that the communications device, previously enabled with a successful CD_ENABLE_REQ primitive, be disabled. In addition, ti specifies the disposition of unsent messages.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_disposal;
} cd_disable_req_t;
```

Parameters

cd_primitive
: Specifies the primitive type.

cd_disposal
: Specifies how unsent message are to be disposed. This field can be one of the following values:

  CD_FLUSH
  : Discard undeliverable data. All data that is unsent at the time that the CD_DISABLE_REQ primitive is received will be discarded. Any data awaiting transmission in the device's write queue will be flushed.

  CD_WAIT
  : Attempt to deliver unsent data. All data that is unsent, at the time that the CD_DISABLE_REQ primitive is received, the provider will attempt to send before confirming the primitive. The provider will not wait for acknowledgement of sent message.

  CD_DELIVER
  : Deliver unsent data. All data that is unsent, at the time that the CD_DISABLE_REQ primitive is received, the provider will deliver before confirming the primitive. The provider will wait for acknowledgement of sent messages.

State

This primitive is valid in state CD_ENABLED.

New State

The new state is CD_DISABLED.

Response

This primitive requires the provider to acknowledge receipt of the primitive as follows:

- **Successful:** When successful, the provider confirms the receipt of the primitive with a CD_DISABLE_CON primitive indicating the success of the operation. The new state is CD_UNATTACHED.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider acknowledges the receipt of the primitive with a CD_ERROR_ACK primitive indicating the error. The state is unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADDISPOSAL	Invalid disposal parameter.
CD_BADPRIM	Unrecognized primitive.
CD_EVENT	Protocol-specific event occurred.
CD_FATALERR	Device has become unusable.
CD_NOTSUPP	Primitive not supported by this device.
CD_OUTSTATE	Primitive was issued from an invalid state.
CD_PROTOSHORT	M_PROTO block too short.
CD_SYSERR	UNIX system error.

4.2.4 Message CD_DISABLE_CON (cd_disable_con_t)

This provider originated primitive confirms that the previous CD_DISABLE_REQ has been successful.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
} cd_disable_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Indicates the primitive type.

cd_state

CD_UNATTACHED
No PPA attached.

CD_UNUSABLE
PPA cannot be used.

CD_DISABLED
PPA attached.

CD_ENABLE_PENDING
Waiting acknowledgement of enable request.

CD_ENABLED
Awaiting use.

CD_READ_ACTIVE
Input section enabled; disabled after data arrives.

CD_INPUT_ALLOWED
Input section permanently enabled.

CD_DISABLE_PENDING
Waiting acknowledgement of disable request.

CD_OUTPUT_ACTIVE
Output section active only.

CD_XRAY
X-raying another PPA.

State

This primitive is issued in the CD_DISABLE_PENDING state.

New State

After issuing this primitive the provider enters the CD_DISABLED state.

4.3 Device Data Transfer Service Primitives

4.3.1 Message CD_ERROR_IND (cd_error_ind_t)

This provider originated primitive indicates that an asynchronous error has occurred and indicates the error number and new state of the CD provider.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_errno;
    cd_ulong cd_explanation;
} cd_error_ind_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Indicates the primitive type.

cd_state
: Indicates the state of the CD provider following the CD_ERROR_IND. *cd_state* can be one of the following values:

  CD_UNATTACHED
  : No PPA attached.

  CD_UNUSABLE
  : PPA cannot be used.

  CD_DISABLED
  : PPA attached.

  CD_ENABLE_PENDING
  : Waiting acknowledgement of enable request.

  CD_ENABLED
  : Awaiting use.

  CD_READ_ACTIVE
  : Input section enabled; disabled after data arrives.

  CD_INPUT_ALLOWED
  : Input section permanently enabled.

  CD_DISABLE_PENDING
  : Waiting acknowledgement of disable request.

  CD_OUTPUT_ACTIVE
  : Output section active only.

  CD_XRAY
  : X-raying another PPA.

cd_errno Indicates the reason for error. *cd_errno* can be one of the following values:

CD_BADFRAME
Defective SDU received.

CD_DISC Disconnected.

CD_EVENT
Protocol-specific event occurred.

CD_FATALERR
Device has become unusable.

CD_READTIMEOUT
Read request timed out before data arrived.

CD_SYSERR
UNIX system error.

CD_WRITEFAIL
Unit data request failed.

cd_explanation
Indicates a futher explanation of the error. When *cd_errno* is *CD_SYSERR*, this field contains the *UNIX* system error as described in `errno(3)`. Otherwise, *cd_explanation* may contain one of the following values:

CD_CRCERR
CRC or FCS error.

CD_DLE_EOT
DLE EOT detected.

CD_FORMAT
Format error detected.

CD_HDLC_ABORT
Aborted frame detected.

CD_OVERRUN
Input overrun.

CD_TOOSHORT
Frame too short.

CD_INCOMPLETE
Partial frame received.

CD_BUSY
Telephone was busy.

CD_NOANSWER
Connection went unanswered.

CD_CALLREJECT
Connection rejected.

CD_HDLC_IDLE	HDLC line went idle.
CD_HDLC_NOTIDLE	HDLC line no longer idle.
CD_QUIESCENT	Line being reassigned.
CD_RESUMED	Line has been reassigned.
CD_DSRTIMEOUT	Did not see DSR in time.
CD_LAN_COLLISIONS	LAN excessive collisions.
CD_LAN_REFUSED	LAN message refused.
CD_LAN_NOSTATION	LAN no such station.
CD_LOSTCTS	Lost Clear to Send signal.
CD_DEVERR	Start of device-specific codes.

In addition, when the explanation is *CD_DEVERR* or greater, the explanation may be a device-specific explanation code.

State

This primitive is valid in any state where data transmission is valid.

New State

The new state is indicated in the primitive.

4.3.2 Message CD_BAD_FRAME_IND (cd_bad_frame_ind_t)

This provider originated primitive indicates that a frame was received in error. The error is indicated along with any data that is retrievable from the frame received in error.

Message Format

This primitive consists of one M_PROTO message block followed by zero or more M_DATA message blocks. The M_PROTO message block is structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_error;
} cd_bad_frame_ind_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Indicates the primitive type.

cd_state
: Indicates the state of the provider following the issuing of the primitive. It can be one of the following values:

  CD_UNATTACHED
  : No PPA attached.

  CD_UNUSABLE
  : PPA cannot be used.

  CD_DISABLED
  : PPA attached.

  CD_ENABLE_PENDING
  : Waiting acknowledgement of enable request.

  CD_ENABLED
  : Awaiting use.

  CD_READ_ACTIVE
  : Input section enabled; disabled after data arrives.

  CD_INPUT_ALLOWED
  : Input section permanently enabled.

  CD_DISABLE_PENDING
  : Waiting acknowledgement of disable request.

  CD_OUTPUT_ACTIVE
  : Output section active only.

  CD_XRAY
  : X-raying another PPA.

cd_error Indicates the error encountered by the frame. Among other values defined for a particular device, this error can be one of the following values:

CD_FRMTOOLONG
: The frame was too long; it overflowed the receive buffer. The data that was successfully received is in the M_DATA message blocks associated with the primitive.

CD_FRMNONOCTET
: The frame was not octet-aligned. This is a residue error. The data that was successfully received (not including the residue error bits) is in the M_DATA message blocks associated with the primitive.

CD_EMPTY_BFR
: The receive buffer is empty. This error is not normally used. No M_DATA message blocks are included with this error.

CD_BAD_CRC
: There was a CRC error in an otherwise correctly received frame. The data that was successfully received, but which failed CRC calculation, is in the M_DATA message blocks associated with the primitive.

CD_FRM_ABORTED
: The frame was aborted. Any successfully received octets at the time of the abort are included in the M_DATA message blocks associated with the primitive.

CD_RCV_OVERRUN
: There was a receiver overrun during the reception of the frame. Any successfully received octets up to the point of the receiver overrun are included in the M_DATA message blocks associated with the primitive.

State

This primitive is valid in any state where the user is not expecting local acknowledgement.

New State

After issuing this primitive, the new state is indicated in the primitive.

4.3.3 Message CD_UNITDATA_IND (cd_unitdata_ind_t)

This provider originated primitive indicates that data has arrived for the specified source and destination addresses with the specified priority.

The M_PROTO message block is only necessary when the parameters included in the primitive are not implied by the communications device.

Message Format

This primitive consists of one M_PROTO message block followed by one or more M_DATA message blocks. The M_PROTO message block is optional. The M_PROTO message block is structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
    cd_ulong cd_src_addr_length;
    cd_ulong cd_src_addr_offset;
    cd_ulong cd_addr_type;
    cd_ulong cd_priority;
    cd_ulong cd_dest_addr_length;
    cd_ulong cd_dest_addr_offset;
} cd_unitdata_ind_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Indicates the primitive type.

cd_state Indicates the state of the CD provider following the indication primitive. *cd_state* can be one of the following values:

CD_UNATTACHED
No PPA attached.

CD_UNUSABLE
PPA cannot be used.

CD_DISABLED
PPA attached.

CD_ENABLE_PENDING
Waiting acknowledgement of enable request.

CD_ENABLED
Awaiting use.

CD_READ_ACTIVE
Input section enabled; disabled after data arrives.

CD_INPUT_ALLOWED
Input section permanently enabled.

> *CD_DISABLE_PENDING*
>
> > Waiting acknowledgement of disable request.
>
> *CD_OUTPUT_ACTIVE*
>
> > Output section active only.
>
> *CD_XRAY*
>
> > X-raying another PPA.

cd_src_addr_length

> Indicates the length of the source address associated with the received data. When the sending endpoint uses CDI, this address is the same as the *cd_dest_addr_length* of the corresponding CD_UNITDATA_REQ primitive. When no source address is provided, or the source address is implicit to the data, this field is coded zero (0).

cd_src_addr_offset

> Indicates the offset of the source address from the beginning of the M_PROTO message block. When *cd_src_addr_length* is zero (0), this field is also zero (0).

cd_addr_type

> *CD_SPECIFIC*
>
> > Indicates that an address is contained in the primitive. When *cd_addr_type* is set to CD_SPECIFIC, a destination address is indicated in the *cd_dest_addr_length* and *cd_dest_addr_offset* fields.
>
> *CD_BROADCAST*
>
> > Indicates that the data was sent to the implicit broadcast address and no specific address follows. When *cd_addr_type* is set to CD_BROADCAST, the fields *cd_dest_addr_length* and *cd_dest_addr_offset* are coded zero (0) and should be ignored by the CD user.
>
> *CD_IMPLICIT*
>
> > Indicates that an implicit address was used, or that the address is embedded in the data. When *cd_addr_type* is set to CD_IMPLICIT, the fields *cd_dest_addr_length* and *cd_dest_addr_offset* are coded zero (0) and should be ignored by the CD user.

cd_priority

> Indicates the priority of the received data. The priority is provider- and device-specific.

cd_dest_addr_length

> Indicates the length of the destination address. When this field is coded zero (0), it indicates that no destination address is included in the message.

cd_dest_addr_offset

> Indicates the offset of the destination address from the start of the M_PROTO message block. When *cd_dest_addr_length* is zero (0), this field is also coded zero (0) and should be ignored by the CD user.

State

This primitive is valid in any state when the device is allowed to receive data (i.e. CD_READ_ACTIVE and CD_INPUT_ALLOWED).

New State

The state remains unchanged.

4.3.4 Message CD_UNITDATA_REQ (cd_unitdata_req_t)

This user originated primitive requests that the specified data be sent to the specified destination address with the specified priority.

Message Format

This primitive consists of one M_PROTO message block followed by one or more M_DATA message blocks. The M_PROTO message block is structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_addr_type;
    cd_ulong cd_priority;
    cd_ulong cd_dest_addr_length;
    cd_ulong cd_dest_addr_offset;
} cd_unitdata_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

cd_addr_type
: Specifies the address type. The address type can be one of the following values:

  CD_SPECIFIC
  : Specifies that an address is contained in the primitive. When *cd_addr_type* is set to CD_SPECIFIC, a destination address must be specified in the *cd_dest_addr_length* and *cd_dest_addr_offset* fields.

  CD_BROADCAST
  : Specifies that the data is to be sent to the implicit broadcast address and no specific address follows. When *cd_addr_type* is set to CD_BROADCAST, the field *cd_dest_addr_length* and *cd_dest_addr_offset* should be coded zero (0) and are ignored by the CD provider.

  CD_IMPLICIT
  : Specifies that an implicit address is to be used, or that the address is embedded in the data. When *cd_addr_type* is set to CD_IMPLICIT, the fields *cd_dest_addr_length* and *cd_dest_addr_offset* should be coded zero (0) and are ignored by the CD provider. No address or embedded address.

cd_priority
: Specifies the priority of the data. Priorities are provider- and device-specific.

cd_dest_addr_length
When *cd_addr_type* is CD_SPECIFIC, this field specifies the length of the destination address to which to send the message. Otherwise, this field is coded zero (0) and ignored by the CD provider.

cd_dest_addr_offset
When *cd_addr_type* is CD_SPECIFIC, this field specifies the offset of the destination address from the start of the M_PROTO message block. Otherwise, this field is ignored by the CD provider.

State

This primitive is valid in states CD_ENABLED, CD_INPUT_ALLOWED, CD_OUTPUT_ACTIVE, CD_READ_ACTIVE.

New State

The state remains unchanged.

Response

This primitive requires an acknowledgement under the following conditions:

- **Successful:** When field *cd_output_style* in CD_INFO_ACK is set to CD_ACKEDOUTPUT, then the provider is required to acknowledge the CD_UNITDATA_REQ with a CD_UNITDATA_ACK. Otherwise, the primitive does not require an acknowledgement. In either case, the state remains unchanged.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider is required to acknowledge the primitive with a CD_ERROR_ACK primitive indicating the error.

Reaons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADADDRESS
Address was invalid.

CD_BADADDRTYPE
Invalid address type.

CD_BADPRIM
Unrecognized primitive.

CD_DISC Disconnected.

CD_EVENT
Protocol-specific event occurred.

CD_FATALERR
Device has become unusable.

CD_NOTSUPP
Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

CD_WRITEFAIL
: Unit data request failed.

4.3.5 Message CD_UNITDATA_ACK (cd_unitdata_ack_t)

This provider originated primitive acknowledges that the previous CD_UNITDATA_REQ primitive was acknowledged as sent.

CD_UNITDATA_ACK primitives are indicated or not depending on the output style as indicated in the *cd_output_style* field of the CD_INFO_ACK primitive as follows:

CD_UNACKEDOUTPUT
No CD_UNITDATA_ACK primitives will be indicated.

CD_ACKEDOUTPUT
CD_UNITDATA_ACK primitives will be issued for every outstanding CD_UNITDATA_REQ.

CD_PACEDOUTPUT
CD_UNITDATA_ACK primitives will only be issued as a timing clue for output.

When the CD_DISABLE_REQ primitive is requested, outstanding acknowledgements may be cancelled depending on the value contained in the *cd_disposal* field of this primitive. When the CD_ABORT_OUTPUT_REQ primitive is requested, outstanding acknowledgements are cancelled.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_state;
} cd_unitdata_ack_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Indicates the primitive type.

cd_state The state of the CD provider following the acknowledgement. *cd_state* can be one of the following values:

CD_UNATTACHED
No PPA attached.

CD_UNUSABLE
PPA cannot be used.

CD_DISABLED
PPA attached.

CD_ENABLE_PENDING
Waiting acknowledgement of enable request.

CD_ENABLED
Awaiting use.

CD_READ_ACTIVE
: Input section enabled; disabled after data arrives.

CD_INPUT_ALLOWED
: Input section permanently allowed.

CD_DISABLE_PENDING
: Waiting acknowledgement of disable request.

CD_OUTPUT_ACTIVE
: Output section active only.

CD_XRAY
: X-raying another PPA.

State

This primitive is valid in any state where a CD_UNITDATA_REQ is outstanding, or when a paced output request is necessary.

New State

The new state is unchanged.

4.4 Device Duplex Management Service Primitives

4.4.1 Message CD_READ_REQ (cd_read_req_t)

This user originated primitive requests that an enabled communications device temporarily allow the input section.

When a stream is enabled with CD_ENABLE_REQ, it can be used for transmission. If the stream returns CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK, the device automatically allows input and there is no need to call the CD_READ_REQ primitive for the device, unless CD_HALT_INPUT_REQ has been successfully called beforehand.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_msec;
} cd_read_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

cd_msec Specifies the interval of time for which to allow the input section, in units of milliseconds.

State

This primitive is valid in the CD_ENABLED state.

New State

When successful, the new state is CD_INPUT_ALLOWED. After the interval, *cd_msec*, has expired, the state will revert to CD_ENABLED.

Response

This primitive requires that the provider acknowledge the receipt of the primitive as follows:

- **Successful:** When successful, the provider acknowledges the receipt of the primitive with the CD_OK_ACK primitive. The new state is CD_INPUT_ALLOWED.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider acknowledges the receipt of the primitive with the CD_ERROR_ACK primitive indicating the error. The new state is unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows.

CD_BADFRAME
Defective SDU received.

CD_BADPRIM
: Unrecognized primitive.

CD_DISC
: Disconnected.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_READTIMEOUT
: Read request timed out before data arrived.

CD_SYSERR
: UNIX system error.

4.4.2 Message CD_ALLOW_INPUT_REQ (cd_allow_input_req_t)

This user originated primitive request that an enabled communications device permanently allow the input section.

When a stream is enabled with CD_ENABLE_REQ, it can be used for transmission. If the stream returns CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK, the device automatically allows input and there is no need to call the CD_ALLOW_INPUT_REQ primitive for the device, unless the CD_HALT_INPUT_REQ has been successfully called beforehand.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_allow_input_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

State

This primitive is valid in the CD_ENABLED state.

New State

When successful, the new stat is CD_INPUT_ALLOWED.

Response

This primitive requires that the provider acknowledge receipt of the primitive as follows:

- **Successful:** When successful, the provider acknowledges the receipt of the primitive with the CD_OK_ACK primitive. The new state is CD_INPUT_ALLOWED.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the provider acknowledges the receipt of the primitive with the CD_ERROR_ACK primitive. The reason for failure is provided in the error field of the primitive. The state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADPRIM
: Unrecognized primitive.

CD_DISC Disconnected.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

If the input section is already allowed and this primitive is issued in the CD_INPUT_ALLOWED state, the provider should ignore the primitive and not generate a non-fatal error.

4.4.3 Message CD_HALT_INPUT_REQ (cd_halt_input_req_t)

This user originated primitive requests that the input section be halted.

When a stream is enabled with CD_ENABLE_REQ, it can be used immediately for transmission. If the stream returns CD_AUTOALLOW in the *cd_features* field of the CD_INFO_ACK, the device automatically allows input and there is no need to call CD_ALLOW_INPUT_REQ for the device. However, CD_HALT_INPUT_REQ will halt input on such a device.

In addition, if the input section is temporarily enabled with CD_READ_REQ, on a half-duplex communications device, then CD_HALT_INPUT_REQ will abort the read operation.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_disposal;
} cd_halt_input_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

cd_disposal
: Specifies how unsent messages are to be disposed. This field can be one of the following values:

  CD_FLUSH
  : Discard undeliverable data. All data that is undelivered at the time that the CD_HALT_INPUT_REQ primitive is received will be discarded. Any data awaiting delivery in the device's read queue will be flushed.

  CD_WAIT
  : Attempt to deliver undelivered data. All data that is undelivered at the timer that the CD_HALT_INPUT_REQ primitive is received the provider will attempt to deliver before acknowledging the primitive. The provider will not wait for acknowledgement of sent messages.

  CD_DELIVER
  : Deliver undelivered data. All data that is undelivered at the time that the CD_HALT_INPUT_REQ primitive is received the provider will deliver before acknowledging the primitive. The provider will also wait for an deliver acknowledgement of sent messages.

State

This primitive is valid in state CD_ENABLED, CD_INPUT_ALLOWED or CD_READ_ACTIVE,

New State

The new state is CD_ENABLED.

Response

This primitive requires that the CD provider acknowledge receipt of the primitive as follows:

- **Successful:** Upon success, the provider will acknowledge receipt of the primitive with the CD_OK_ACK primitive. The new state is CD_ENABLED.
- **Unsuccessful (non-fatal errors):** Upon failure, the provider will acknowledge receipt of the primitive with the CD_ERROR_ACK primitive with the error indicated in the primitive. The new state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADDISPOSAL	Invalid disposal parameter.
CD_BADPRIM	Unrecognized primitive.
CD_DISC	Disconnected.
CD_EVENT	Protocol-specific event occurred.
CD_FATALERR	Device has become unusable.
CD_NOTSUPP	Primitive not supported by this device.
CD_OUTSTATE	Primitive was issued from an invalid state.
CD_PROTOSHORT	M_PROTO block too short.
CD_SYSERR	UNIX system error.

If the communications device is in the CD_ENABLED state and the input section is not active, the CD_HALT_INPUT_REQ primitive should be ignored and no non-fatal error generated.

4.4.4 Message CD_ABORT_OUTPUT_REQ (cd_abort_output_req_t)

This user originated primitive requests that any transmission operation currently in progress be aborted.

Message Format

This primitive consists of one M_PROTO or M_PCPROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_abort_output_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

State

This primitive is valid in any state in which the output section is enabled, but no local acknowledgement is pending.

New State

The new state remains unchanged unless the current state is CD_OUTPUT_ACTIVE, in which case the new state is CD_ENABLED or CD_INPUT_ALLOWED, depending on the state of the input section.

Response

This primitive requires the CD provider to acknowledge receipt of the primitive as follows:

- **Successful:** Upon success, the provider acknowledges the receipt of the primitive with the CD_OK_ACK primitive. The new state is unchanged.
- **Unsuccessful (non-fatal errors):** Upon failure, the provider acknowledges the receipt of the primitive with the CD_ERROR_ACK including the reason for failure. The new state remains unchanged.

  Note that if the output section is not active at the time that the CD_ABORT_OUTPUT_REQ is issued, but the communications device is enabled, the provider should discard the CD_ABORT_OUTPUT_REQ primitive and not issue any non-fatal error.

Reasons for Failure

CD_BADPRIM
: Unrecognized primitive.

CD_DISC
: Disconnected.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

4.4.5 Message CD_WRITE_READ_REQ (cd_write_read_req_t)

This user originated primitive requests that the provided data be transmitted and that the output section be disabled and the input section enabled immediately following the transmission.

Message Format

This primitive consists of one M_PROTO message block followed by one or more M_DATA message blocks. The M_PROTO message block is structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_unitdata_req_t cd_unitdata_req;
    cd_read_req_t cd_read_req;
} cd_write_read_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
: Specifies the primitive type.

cd_unitdata_req
: Specifies a CD_UNITDATA_REQ primitive. See [Message CD˙UNITDATA˙REQ (cd˙unitdata˙req˙t)], page 68 for formatting of this parameter.

cd_read_req
: Specifies a CD_READ_REQ primitive. See [Message CD˙READ˙REQ (cd˙read˙req˙t)], page 74 for formatting of this parameter.

State

This primitive is valid in any state where the CD_UNITDATA_REQ primitive and CD_ALLOW_INPUT_REQ primitive are permitted.

New State

The new state remains unchanged.

Response

– **Successful:** This primitive requires the same response from the CD provider as if a CD_UNITDATA_REQ primitive immediately followed by a CD_READ_REQ primitive were to be issued.
– **Unsuccessful (non-fatal errors):** When unsuccessful, this primitive requires an error acknowledgement using the CD_ERROR_ACK primitive with the error indicated.

Reasons for Failure

Non-Fatal Errors: appropriate non-fatal errors are as follows:

CD_BADADDRESS
Address was invalid.

CD_BADADDRTYPE
Invalid address type.

CD_BADPRIM
Unrecognized primitive.

CD_DISC Disconnected.

CD_EVENT
Protocol-specific event occurred.

CD_FATALERR
Device has become unusable.

CD_NOTSUPP
Primitive not supported by this device.

CD_OUTSTATE
Primitive was issued from an invalid state.

CD_PROTOSHORT
M_PROTO block too short.

CD_READTIMEOUT
Read request timed out before data arrived.

CD_SYSERR
UNIX system error.

CD_WRITEFAIL
Unit data request failed.

4.5 Lead and Signal Service Primitives

4.5.1 Message CD_MODEM_SIG_IND (cd_modem_sig_ind_t)

This provider originated primitive indicates the status of a number of modem lines and signals. This primitive is issued in response to a change in modem signals or in response to a CD_MODEM_SIG_POLL primitive.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_sigs;
} cd_modem_sig_ind_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Indicates the primitive type.

cd_sigs Indicates the state of specific modem lines and signals as a bitwise OR of any of the following flags (when the flag is set, the signal is asserted):

CD_DTR	Data terminal ready.
CD_RTS	Request to send.
CD_DSR	Data set ready.
CD_DCD	Data carrier detect.
CD_CTS	Clear to send.
CD_RI	Ring indicator.

State

This primitive can be issued by the CD provider in any state.

New State

The new state is unchanged.

4.5.2 Message CD_MODEM_SIG_POLL (cd_modem_sig_poll_t)

This user originated primitive request that the CD provider respond with a CD_MODEM_SIG_IND indicating the current state of modem lines and signals.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
} cd_modem_sig_poll_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

State

This primitive is valid in any state other than CD_UNUSABLE or CD_UNATTACHED, and where a local acknowledgement is not pending.

New State

The new state is unchanged.

Response

This primitive requires the CD provider to respond with an acknowledgement message as follows:

- **Successful:** When successful, the CD provider response with a CD_MODEM_SIG_IND indicating the state of modem leads and signals.
- **Unsuccessful (non-fatal errors):** When unsuccessful, the CD provider responds with a CD_ERROR_ACK primitive indicating the reason for failure.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADPRIM
Unrecognized primitive.

CD_DISC Disconnected.

CD_EVENT
Protocol-specific event occurred.

CD_FATALERR
Device has become unusable.

CD_NOTSUPP
Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

4.5.3 Message CD_MODEM_SIG_REQ (cd_modem_sig_req_t)

This user originated primitive request that the CD provider assert or de-assert the specified modem leads and signals.

Message Format

This primitive consists of one M_PROTO message block structured as follows:

```
typedef struct {
    cd_ulong cd_primitive;
    cd_ulong cd_sigs;
} cd_modem_sig_req_t;
```

Parameters

This primitive contains the following parameters:

cd_primitive
Specifies the primitive type.

cd_sigs Specifies the signals to assert or de-assert, and is a bitwise OR of the following flags:

CD_DTR	Data terminal ready.
CD_RTS	Request to send.
CD_DSR	Data set ready.
CD_DCD	Data carrier detect.
CD_CTS	Clear to send.
CD_RI	Ring indicator.

If the flag is set in *cd_sigs*, the corresponding lead will be asserted. If the flag is clear, the corresponding lead will be de-asserted. Flags that are not output leads and are input leads only (such as CD_DCD) are ignored.

State

This primitive is valid in any state other than CD_UNATTACHED or CD_UNUSABLE, and where a local acknowledgement is not pending.

New State

The state remains unchanged.

Response

This primitive requires the CD provider to acknowledge receipt of the primitive as follows:

- **Successful:** Upon success, the CD provider acknowledges receipt of the primitive with a CD_OK_ACK primitive. The state remains unchanged.
- **Unsuccessful (non-fatal errors):** Upon failure, the CD provider acknowledges receipt of the primitive with a CD_ERROR_ACK indicating the reason for failure in the error number. The state remains unchanged.

Reasons for Failure

Non-Fatal Errors: applicable non-fatal errors are as follows:

CD_BADPRIM
: Unrecognized primitive.

CD_EVENT
: Protocol-specific event occurred.

CD_FATALERR
: Device has become unusable.

CD_NOTSUPP
: Primitive not supported by this device.

CD_OUTSTATE
: Primitive was issued from an invalid state.

CD_PROTOSHORT
: M_PROTO block too short.

CD_SYSERR
: UNIX system error.

5 Allowable Sequence of CDI Primitives

6 Precedence of CDI Primitives

Appendix A Guidelines for Protocol Independent CDS Users

Appendix B Required Information for CDS Provider-Specific Addenda

Appendix C CDI Header File

```
#ifndef _SYS_CDI_H
#define _SYS_CDI_H

#ident "@(#) cdi.h 1.20 11/17/03"

/* This file can be processed by doxygen(1). */

/// @file
/// @brief This header file provides the Communications Device Interface (CDI).

/*
   cdi.h header for Communications Device Interface

   Copyright (c) 1989 NCR Comten

   This file distributed by Gcom, Inc with permission of NCR Comten
 */

/*
   Primitives for Local Management Services
 */
#define CD_INFO_REQ             0x00    /* Information request */
#define CD_INFO_ACK             0x01    /* Information acknowledgement */
#define CD_ATTACH_REQ           0x02    /* Attach a PPA */
#define CD_DETACH_REQ           0x03    /* Detach a PPA */
#define CD_ENABLE_REQ           0x04    /* Prepare a device */
#define CD_DISABLE_REQ          0x05    /* Disable a device */
#define CD_OK_ACK               0x06    /* Success acknowledgement */
#define CD_ERROR_ACK            0x07    /* Error acknowledgement */
#define CD_ENABLE_CON           0x08    /* Enable confirmation */
#define CD_DISABLE_CON          0x09    /* Disable confirmation */
#define CD_ERROR_IND            0x0a    /* Error indication */

/*
   Primitives used for Data Transfer
 */
#define CD_ALLOW_INPUT_REQ      0x0b    /* Allow input */
#define CD_READ_REQ             0x0c    /* Wait-for-input request */
#define CD_UNITDATA_REQ         0x0d    /* Data send request */
#define CD_WRITE_READ_REQ       0x0e    /* Write/read request */
#define CD_UNITDATA_ACK         0x0f    /* Data send acknowledgement */
#define CD_UNITDATA_IND         0x10    /* Data receive indication */
#define CD_HALT_INPUT_REQ       0x11    /* Halt input */
#define CD_ABORT_OUTPUT_REQ     0x12    /* Abort output */
#define CD_MUX_NAME_REQ         0x13    /* get mux name (Gcom) */
#define CD_BAD_FRAME_IND        0x14    /* frame w/error (Gcom extension) */
#define CD_MODEM_SIG_REQ        0x15    /* Assert modem signals (Gcom) */
#define CD_MODEM_SIG_IND        0x16    /* Report modem signal state (Gcom) */
#define CD_MODEM_SIG_POLL       0x17    /* requests a CD_MODEM_SIG_IND (Gcom) */

/*
   CDI device states
 */
#define CD_UNATTACHED           0x00    /* No PPA attached */
```

```
#define CD_UNUSABLE             0x01    /* PPA cannot be used */
#define CD_DISABLED             0x02    /* PPA attached */
#define CD_ENABLE_PENDING       0x03    /* Waiting ack of enable req */
#define CD_ENABLED              0x04    /* Awaiting use */
#define CD_READ_ACTIVE          0x05    /* Input section enabled; */
                                        /* disabled after data arrives */
#define CD_INPUT_ALLOWED        0x06    /* Input section permanently enabled */
#define CD_DISABLE_PENDING      0x07    /* Waiting ack of disable req */
#define CD_OUTPUT_ACTIVE        0x08    /* Output section active only */
#define CD_XRAY                 0x09    /* Xray-ing another ppa */
#define CD_NOT_AUTH             0x0A    /* Not authorized, unusable */

/*
   CD_ERROR_ACK and CD_ERROR_IND error return values
 */
#define CD_BADADDRESS           0x01    /* Address was invalid */
#define CD_BADADDRTYPE          0x02    /* Invalid address type */
#define CD_BADDIAL              0x03    /* Dial information was invalid */
#define CD_BADDIALTYPE          0x04    /* Invalid dial information type */
#define CD_BADDISPOSAL          0x05    /* Invalid disposal parameter */
#define CD_BADFRAME             0x06    /* Defective SDU received */
#define CD_BADPPA               0x07    /* Invalid PPA identifier */
#define CD_BADPRIM              0x08    /* Unrecognized primitive */
#define CD_DISC                 0x09    /* Disconnected */
#define CD_EVENT                0x0a    /* Protocol-specific event occurred */
#define CD_FATALERR             0x0b    /* Device has become unusable */
#define CD_INITFAILED           0x0c    /* Line initialization failed */
#define CD_NOTSUPP              0x0d    /* Primitive not supported by this device */
#define CD_OUTSTATE             0x0e    /* Primitive was issued from an invalid state */
#define CD_PROTOSHORT           0x0f    /* M_PROTO block too short */
#define CD_READTIMEOUT          0x10    /* Read request timed out before data arrived */
#define CD_SYSERR               0x11    /* UNIX system error */
#define CD_WRITEFAIL            0x12    /* Unitdata request failed */

/*
   Error explanations
 */
#define CD_CRCERR               0x01    /* CRC or FCS error */
#define CD_DLE_EOT              0x02    /* DLE EOT detected */
#define CD_FORMAT               0x03    /* Format error detected */
#define CD_HDLC_ABORT           0x04    /* Aborted frame detected */
#define CD_OVERRUN              0x05    /* Input overrun */
#define CD_TOOSHORT             0x06    /* Frame too short */
#define CD_INCOMPLETE           0x07    /* Partial frame received */
#define CD_BUSY                 0x08    /* Telephone was busy */
#define CD_NOANSWER             0x09    /* Connection went unanswered */
#define CD_CALLREJECT           0x0a    /* Connection rejected */
#define CD_HDLC_IDLE            0x0b    /* HDLC line went idle */
#define CD_HDLC_NOTIDLE         0x0c    /* HDLC line no longer idle */
#define CD_QUIESCENT            0x0d    /* Line being reassigned */
#define CD_RESUMED              0x0e    /* Line has been reassigned */
#define CD_DSRTIMEOUT           0x0f    /* Did not see DSR in time */
#define CD_LAN_COLLISIONS       0x10    /* LAN excessive collisions */
#define CD_LAN_REFUSED          0x11    /* LAN message refused */
#define CD_LAN_NOSTATION        0x12    /* LAN no such station */
#define CD_LOSTCTS              0x13    /* Lost Clear to Send signal */
```

```
#define CD_DEVERR               0x100   /* Start of device-specific codes */

/*
   CDI device classes
 */
#define CD_HDLC         0x00    /* Bit-synchronous */
#define CD_BISYNC       0x01    /* Character-synchronous */
#define CD_LAN          0x02    /* ISO 8802-3,4,5 local-area network MAC */
#define CD_NODEV        0x03    /* no device, ppa used for X-ray */

/*
   CDI duplex types
 */
#define CD_FULLDUPLEX   0x00    /* Full duplex; allow input supported */
#define CD_HALFDUPLEX   0x01    /* Half duplex; read and write/read supported */

/*
   CDI output styles
 */
#define CD_UNACKEDOUTPUT 0x00   /* No unitdata acknowledgements */
#define CD_ACKEDOUTPUT   0x01   /* Unitdata acknowledgements */
#define CD_PACEDOUTPUT   0x02   /* Unitdata acks as output timing hints */

/*
   CDI optional features
 */
#define CD_CANREAD      0x01    /* Read request supported on full duplex */
#define CD_CANDIAL      0x02    /* Dial information supported */
#define CD_AUTOALLOW    0x04    /* CD_INPUT_ALLOWED as soon as enabled */
#define CD_KEEPALIVE    0x08    /* Gcom: Don't send off at CD_DISABLE_REQ */

/*
   CDI provider style.

   The CDI provider style which determines whether a provider requires a
   CD_ATTACH_REQ to inform the provider which PPA user messages should be
   sent/received on.
 */
#define CD_STYLE1       0x00    /* PPA is implicitly bound by open(2) */
#define CD_STYLE2       0x01    /* PPA must be explicitly bound via CD_ATTACH_REQ */
#define CD_STYLE_1      CD_STYLE1       /* Gcom -- to match document */
#define CD_STYLE_2      CD_STYLE2       /* Gcom -- to match document */

/*
   Symbolic value for "no dialing information"
 */
#define CD_NODIAL       0x00

/*
   Actions to take with undelivered data in a CD_DISABLE_REQ or
 CD_HALT_INPUT_REQ
 */
#define CD_FLUSH        0x00    /* Discard undelivered data */
#define CD_WAIT         0x01    /* Attempt to deliver unsent data */
#define CD_DELIVER      0x02
```

```
/*
   Address types
 */
#define CD_SPECIFIC     0x00    /* Specific address follows */
#define CD_BROADCAST    0x01    /* Broadcast; no address follows */
#define CD_IMPLICIT     0x02    /* No address or embedded address */

/*
   Error types for CD_BAD_FRAME_IND
 */

#define CD_FRMTOOLONG   0xFFFF  /* frame overflowed rcv bfr */
#define CD_FRMNONOCTET  0xFFFE  /* frame not octet-aligned */
#define CD_EMPTY_BFR    0xFFFD  /* empty rcv buffer (not used) */
#define CD_BAD_CRC      0xFFFC  /* CRC error */
#define CD_FRM_ABORTED  0xFFFB  /* frame aborted */
#define CD_RCV_OVERRUN  0xFFFA  /* receive overrun */

/*
   Modem signal bits for modem signal related requests and indications
 */
#define CD_DTR          0x01
#define CD_RTS          0x02
#define CD_DSR          0x04
#define CD_DCD          0x08
#define CD_CTS          0x10
#define CD_RI           0x20

/*
   CDI interface primitive definitions.

   Each primitive is sent as a Stream message.  It is possible that the messages
   may be viewed as a sequence of bytes that have the following form without any
   padding. The structure definition of the following messages may have to
   change depending on the underlying hardware architecture and crossing of a
   hardware boundary with a different hardware architecture.

   Each message has the name defined followed by the Stream message type
   (M_PROTO, M_PCPROTO, M_DATA)
 */
#ifdef __LP64__
typedef int32_t cd_long;
typedef u_int32_t cd_ulong;
typedef u_int16_t cd_ushort;
#else                           /* __LP64__ */
typedef long cd_long;
typedef ulong cd_ulong;
typedef ushort cd_ushort;
#endif                          /* __LP64__ */

/*
        LOCAL MANAGEMENT PRIMITIVES
 */

/*
   CD_INFO_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
} cd_info_req_t;

/*
   CD_INFO_ACK, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_max_sdu;
        cd_ulong cd_min_sdu;
        cd_ulong cd_class;
        cd_ulong cd_duplex;
        cd_ulong cd_output_style;
        cd_ulong cd_features;
        cd_ulong cd_addr_length;
        cd_ulong cd_ppa_style;
} cd_info_ack_t;

/*
   CD_ATTACH_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_ppa;
} cd_attach_req_t;

/*
   CD_DETACH_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
} cd_detach_req_t;

/*
   CD_ENABLE_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_dial_type;
        cd_ulong cd_dial_length;
        cd_ulong cd_dial_offset;
} cd_enable_req_t;

/*
   CD_DISABLE_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_disposal;
} cd_disable_req_t;

/*
   CD_OK_ACK, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_correct_primitive;
} cd_ok_ack_t;

/*
   CD_ERROR_ACK, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_error_primitive;
        cd_ulong cd_errno;
        cd_ulong cd_explanation;
} cd_error_ack_t;

/*
   CD_ENABLE_CON, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
} cd_enable_con_t;

/*
   CD_DISABLE_CON, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
} cd_disable_con_t;

/*
   CD_ERROR_IND, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_errno;
        cd_ulong cd_explanation;
} cd_error_ind_t;

/*
        DATA TRANSFER PRIMITIVES
 */

/*
   CD_ALLOW_INPUT_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
} cd_allow_input_req_t;

/*
```

```
   CD_READ_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_msec;
} cd_read_req_t;

/*
   CD_UNITDATA_REQ, optional M_PROTO type, with M_DATA block(s)
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ushort cd_addr_type;
        cd_ushort cd_priority;
        cd_ulong cd_dest_addr_length;
        cd_ulong cd_dest_addr_offset;
} cd_unitdata_req_t;

/*
   CD_WRITE_READ_REQ, M_PROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_unitdata_req_t cd_unitdata_req;
        cd_read_req_t cd_read_req;
} cd_write_read_req_t;

/*
   CD_UNITDATA_ACK, M_PROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
} cd_unitdata_ack_t;

/*
   CD_UNITDATA_IND, optional M_PROTO type, with M_DATA block(s)
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_src_addr_length;
        cd_ulong cd_src_addr_offset;
        cd_ushort cd_addr_type;
        cd_ushort cd_priority;
        cd_ulong cd_dest_addr_length;
        cd_ulong cd_dest_addr_offset;
} cd_unitdata_ind_t;

/*
   CD_BAD_FRAME_IND, M_PROTO type, with M_DATA block(s)
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_state;
        cd_ulong cd_error;              /* what is wrong with the frame */
```

```
} cd_bad_frame_ind_t;

/*
   CD_MODEM_SIG_REQ, M_PROTO type

   Assert the modem signals with '1' bits in the cd_sigs mask and drop those
   signals with '0' bits.  Sensed modem signals such as DCD or CTS are ignored.
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_sigs;

} cd_modem_sig_req_t;

/*
   CD_MODEM_SIG_IND, M_PROTO type

   The cd_sigs field reports the current state of the modem signals.  This
   message is sent when modem signals change at the hardware interface.  Only
   changes in signals selected by the cd_modem_sig_enb_req_t cd_sigs mask will
   be evaluated for purposes of change detection.
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_sigs;

} cd_modem_sig_ind_t;

typedef struct {
        cd_ulong cd_primitive;

} cd_modem_sig_poll_t;

/*
   CD_HALT_INPUT_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
        cd_ulong cd_disposal;
} cd_halt_input_req_t;

/*
   CD_ABORT_OUTPUT_REQ, M_PROTO or M_PCPROTO type
 */
typedef struct {
        cd_ulong cd_primitive;
} cd_abort_output_req_t;

union CD_primitives {
        cd_ulong cd_primitive;
        cd_info_req_t info_req;
        cd_info_ack_t info_ack;
        cd_attach_req_t attach_req;
        cd_detach_req_t detach_req;
        cd_enable_req_t enable_req;
        cd_disable_req_t disable_req;
        cd_ok_ack_t ok_ack;
        cd_error_ack_t error_ack;
        cd_allow_input_req_t allow_input_req;
        cd_read_req_t read_req;
        cd_unitdata_req_t unitdata_req;
        cd_write_read_req_t write_read_req;
        cd_unitdata_ack_t unitdata_ack;
        cd_unitdata_ind_t unitdata_ind;
        cd_halt_input_req_t halt_input_req;
        cd_abort_output_req_t abort_output_req;
        cd_error_ind_t error_ind;
        cd_enable_con_t enable_con;
        cd_disable_con_t disable_con;
        cd_bad_frame_ind_t bad_frame_ind;
        cd_modem_sig_req_t modem_sig_req;
        cd_modem_sig_ind_t modem_sig_ind;
        cd_modem_sig_poll_t modem_sig_poll;
};

#define CD_INFO_REQ_SIZE                sizeof(cd_info_req_t)
#define CD_INFO_ACK_SIZE                sizeof(cd_info_ack_t)
#define CD_ATTACH_REQ_SIZE              sizeof(cd_attach_req_t)
#define CD_DETACH_REQ_SIZE              sizeof(cd_detach_req_t)
#define CD_ENABLE_REQ_SIZE              sizeof(cd_enable_req_t)
#define CD_DISABLE_REQ_SIZE             sizeof(cd_disable_req_t)
#define CD_OK_ACK_SIZE                  sizeof(cd_ok_ack_t)
#define CD_ERROR_ACK_SIZE               sizeof(cd_error_ack_t)
#define CD_ALLOW_INPUT_REQ_SIZE         sizeof(cd_allow_input_req_t)
#define CD_READ_REQ_SIZE                sizeof(cd_read_req_t)
#define CD_UNITDATA_REQ_SIZE            sizeof(cd_unitdata_req_t)
#define CD_WRITE_READ_REQ_SIZE          sizeof(cd_write_read_req_t)
#define CD_UNITDATA_ACK_SIZE            sizeof(cd_unitdata_ack_t)
#define CD_UNITDATA_IND_SIZE            sizeof(cd_unitdata_ind_t)
#define CD_HALT_INPUT_REQ_SIZE          sizeof(cd_halt_input_req_t)
#define CD_ABORT_OUTPUT_REQ_SIZE        sizeof(cd_abort_output_req_t)
#define CD_ERROR_IND_SIZE               sizeof(cd_error_ind_t)
#define CD_ENABLE_CON_SIZE              sizeof(cd_enable_con_t)
#define CD_DISABLE_CON_SIZE             sizeof(cd_disable_con_t)
#define CD_BAD_FRAME_IND_SIZE           sizeof(cd_bad_frame_ind_t)
#define CD_MODEM_SIG_REQ_SIZE           sizeof(cd_modem_sig_req_t)
#define CD_MODEM_SIG_IND_SIZE           sizeof(cd_modem_sig_ind_t)
#define CD_MODEM_SIG_POLL_SIZE          sizeof(cd_modem_sig_poll_t)

#endif                          /* _SYS_CDI_H */
```

Appendix D Glossary of CDI Terms and Acronyms

References

Index