AIXlink/X.25 Version 2.0 for AIX: Guide and Reference

SC23-2520-04

AIXlink/X.25 Version 2.0 for AIX: Guide and Reference

SC23-2520-04

Note

Before using this information and the product it supports, read the information in Appendix J, “Notices,” on page 331.

First Edition (December 2001)

This edition of *AIXlink/X.25 Version 2.0 for AIX: Guide and Reference* applies to AIX 5.2 and to all subsequent releases of these products until otherwise indicated in new editions.

Contents

About This Book

The *AIXlink/X.25 Version 2.0 for AIX: Guide and Reference* provides information on using, managing, and programming the AIXlink/X.25 Version 2.0 application for AIX Version 5.

Who Should Use This Book

This book is intended primarily for system administrators and application developers for X.25, but it also contains some end-user information.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-Sensitivity in AIX

Everything in the AIX operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type `LS`, the system responds that the command is ″not found.″ Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

The following books contain information about or related to AIXlink/X.25 2.0:

- *AIX 5L Version 5.2 Commands Reference*
- *RS/6000 X.25 Cookbook*
- *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*
- *AIX 5L Version 5.2 Communications Programming Concepts*
- *AIX 5L Version 5.2 General Programming Concepts: Writing and Debugging Programs*
- *AIX 5L Version 5.2 Files Reference*
- *AIX 5L Version 5.2 System Management Guide: Communications and Networks*
- *AIX 5L Version 5.2 System Management Guide: Operating System and Devices*

Chapter 1. X.25 Network Communications Overview

X.25 can be a cost-effective means of networking systems in a wide geographical area, compared to traditional dial-up (circuit switched) connections, or remote-bridged local area networks (LANs) connected by leased lines. It provides worldwide interconnection for international corporations.

X.25 provides the ability to transmit data between remote machines. It is a set of recommendations from the International Telegraph and Telephone Consultative Committee (CCITT). These recommendations define a standard network access protocol for attaching different types of computer equipment to a packet-switched data network (PSDN). A PSDN is an interconnecting set of switching nodes that enables subscribers to exchange data using a standard protocol and packet-switching technology. This protocol is particularly useful for communication between different types of computer systems and for accessing public databases.

Both public and private PSDNs can be based on the X.25 protocol. Public networks are usually provided nationally by the Post, Telegraph, and Telecommunications (PTT) authority. Private networks are operated by individual corporations. Many of the corporations using X.25 networks have a requirement for communication between themselves and other companies, such as dealers and agents. An example of such an X.25 network use is an airline reservation system.

Although some corporations have created private networks, most companies subscribe to a public PSDN. Such a network carries messages divided into packets over circuits that are shared by many network users. A single physical line into an office can handle many concurrent connections. A *packet* consists of a sequence of data and control elements in a format that is always transmitted as a whole. The network packet size is commonly 128 bytes, but can vary from 16 to 4096 bytes. In X.25, a byte is called an *octet*.

You can use X.25 communications to provide a network service for higher-level protocols, such as System Network Architecture (SNA) and Transmission Control Protocol/Internet Protocol (TCP/IP). Or you can use an X.25 network directly, either by using the **xtalk** command, or by using an application programming interface (API) to write your own applications.

X.25 Equipment Terminology

The terms DTE, DCE, and DSE are used with both the X.25 protocol and modems, with slightly different meanings. The CCITT has defined these terms as follows for X.25 protocol:

DTE (Data-Terminal Equipment.) A computer that uses a network for communications.
DCE (Data Circuit-Terminating Equipment.) A device at the point of access to a network.
DSE (Data-Switching Equipment.) A switching node in a packet-switched data network.

Notes:

1. Every DTE must have an associated DCE.
2. DTE and DCE are functional definitions; they need not correspond to specific items of equipment. For example, a single device may be a DSE and may also provide multiple DCE interfaces.

X.25 is not an end-to-end protocol. *CCITT Recommendation X.25* defines a standard protocol for information exchange in packet mode between a DTE and a DCE (that is, between an individual user's equipment and the network provider's equipment).

The network is composed of DCEs and DSEs that route the packets of data through the network to the intended destination. The path that a user's data takes might vary with every packet. In most cases the DTE is connected to a DCE in some form of network. In a few cases, two systems might be attached

more directly, without an intermediate network. When this is the case, one system has to act as a DCE at the Data Link layer. This usage is not common because it restricts the flexibility of the protocol and degrades performance.

Operation and maintenance of DCEs and DSEs are the responsibilities of the network provider. If a link between two DSEs goes down, the provider must reroute traffic. X.25 does not define the route through the network or the protocols employed within it.

The following diagram shows these elements of a packet-switched data network.

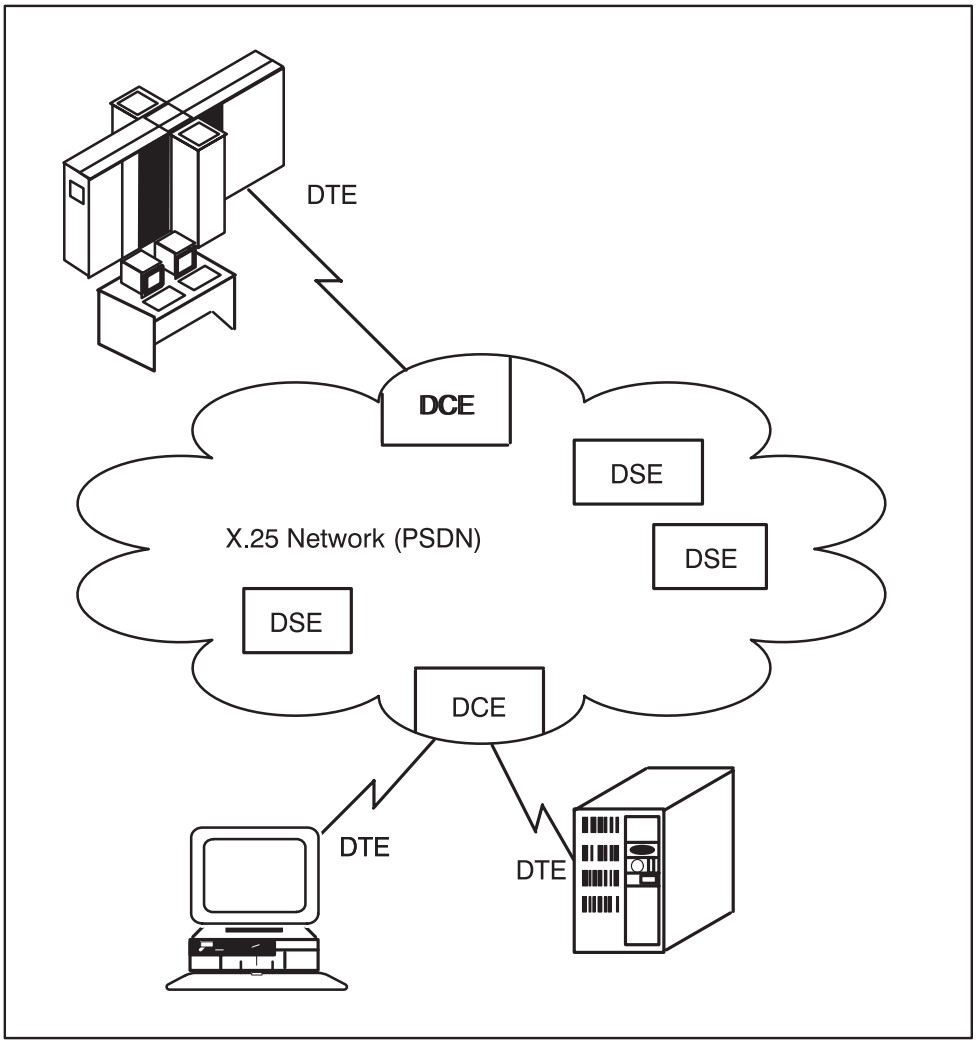


Figure 1. The X.25 Network

X.25 Levels

Several models have been used to specify how networks work. One of the most common of these conceptual models is the International Standards Organization's Open System Interconnect (OSI) Reference Model , also referred to as the OSI 7-layer model. The OSI model numbers the seven layers, or levels, beginning at the lowest (physical) level, as follows:

7 Application
6 Presentation

5 Session
4 Transport
3 Network
2 Data Link
1 Physical

Levels 1-3 are network-specific, and differ depending on the physical network used. Levels 4-7 are network-independent, higher-level functions. The X.25 protocol has three levels . These levels correspond to Levels 1, 2, and 3 of the OSI model as follows:

Physical level	OSI Level 1 (Physical)
Link level/Frame Level	OSI Level 2 (Data Link)
Packet level	OSI Level 3 (Network)

The X.25 protocol can be quite complicated to set up. If protocol information is required, refer to a communications textbook. Each layer relies on the lower layers to be functional for it to work. If a problem is encountered when setting up an X.25 connection, each layer should be checked to see if it is active.

Physical Level

The physical level activates, maintains, and deactivates the physical circuit between a DTE and a DCE. The physical level is implemented as a STREAMS driver and performs the following functions:

- Activate and deactivate physical circuits using electrical signals.
- Maintain line characteristics of the selected interface.
- Indicate faulty incoming HDLC frames, such as frames with the wrong length.
- Allow configuration of auto call units (ACU) for systems with dial-up X.25 connections.

This implementation supports three physical interfaces: V.24, V.35, and X.21bis. CCITT recommendations V.24 and V.35 are found in 1988 volume VIII.I. The CCITT X.21bis is found in volume VIII.

Link Level or Frame Level

The packet layer produces X.25 packets to establish calls and transfer data. All these packets are then passed to the frame layer for transmission to the local DCE. The frame layer uses a link-access procedure to ensure that data and control information are accurately exchanged over the physical circuit between the DTE and DCE. It provides recovery procedures and is based on a subset of the high-level data-link control (HDLC) protocol called LAP-B. It is synchronous and full-duplex. Once a link is started, either station can transfer information without waiting for permission from the other.

In HDLC all commands, responses, and data are transmitted in frames. Each frame has a header containing address and control information, and a trailer containing a frame-check sequence. Normally, none of this is seen when using X.25.

The three types of frames are:

I Information frames. These frames transfer user data and are numbered sequentially. All X.25 packets are transferred within **I** frames.

S Supervisory frames. These frames are numbered sequentially and supervise the link, performing such functions as:

- Acknowledging **I** frames.
- Requesting retransmission of **I** frames.
- Requesting temporary suspension of transmission of **I** frames.

U Unnumbered frames. These frames describe the mode of operation, such as the command Set Asynchronous Balanced Mode (SABM).

Packet Level

The packet-level protocol specifies how X.25 controls calls and data transfers between systems. There are many networks running X.25, and a number of these are interconnected. Each system connected to the network has an address to identify it, and this address is used when a connection from the local system to the remote is being requested.

When a system is installed with X.25 and a network subscription obtained, various pieces of configuration information are supplied by the network provider. This information is used to configure the X.25 software. One or more X.25 lines can be connected and for each line, an X.25 port will be configured.

The data transfer capacity of the X.25 line may be shared between a number of different sessions. The maximum number of subscriptions is based on the network subscription and the capabilities of the DTE hardware and software. Each session is called a *virtual circuit*. A virtual circuit is a data circuit between the local and remote systems, but a circuit that may have its route switched within the network. The element of the network subscription that limits the number of simultaneous virtual circuits in use is the number of *logical channels* subscribed to. Each virtual circuit takes up a logical channel for the period the circuit is active.

Logical Channels

Logical channels are the communications paths between a DTE and its data circuit-terminating equipment (DCE). For 15 simultaneous connections across a network, the network supplier must provide 15 logical channels. Valid logical channel numbers range from 1 to 4095 (logical channel number 0 is usually reserved for diagnostics). The network provider assigns the specific logical channel numbers, and each number must match between the DTE and its DCE. For example, a DTE configured to use logical channels 51-58 could not communicate with its DCE using logical channels 3002-3009. However, when a DTE communicates with another DTE across a packet-switched data network (PSDN), the DTE logical channel numbers do not have to match. The logical channel is not end-to-end in the network; it must only match between each DTE/DCE pair, as shown in the following diagram:

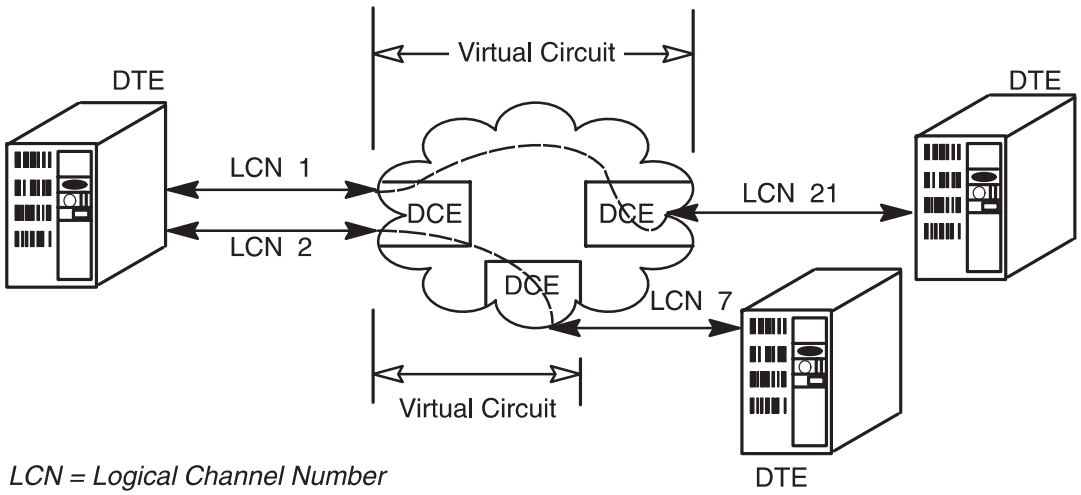


Figure 2. Logical Channels and Virtual Circuits

Virtual Circuits

When a user application begins a session with another DTE on the network, a virtual circuit is established from one DTE to the other, through the DCEs on the network. For an outgoing call, the system that originates the call automatically selects a free channel. When an incoming call is received, the system's local DCE selects the channel. Each application running between two hosts requires use of a virtual circuit. There are two kinds of virtual circuits, permanent (PVC) and switched (SVC), of which SVCs are the more commonly used. Normally when using an SVC, it is not necessary to know the logical channel number in use; the X.25 software ensures that the data from a given session goes over the appropriate virtual circuit. It is necessary to know the logical channel number a given session is on when using PVCs or when tracing the traffic on the X.25 line.

Some applications multiplex their own connections over one virtual circuit. For example, once a virtual circuit is established between two machines for TCP/IP, all TCP/IP traffic between those machines flows on that circuit.

It is important that the virtual circuit ranges used for configuration agree with those defined by the network provider.

Permanent Virtual Circuits:

Configure permanent virtual circuits (PVCs) to agree with the network provider's subscription. The subscription is permanently configured so that the PVC occupying a particular logical channel number (LCN) connects to a remote machine on a specified LCN on the remote machine. This allows call setup time to be saved, but dedicates the channel to that one remote system. This makes PVCs less flexible and less likely to be used.

Switched Virtual Circuits:

A switched virtual circuit (SVC) is a virtual circuit that exists only for the duration of the call, acting like a connection over a telephone network. There are three types of logical channels for SVCs:

Incoming	The DTE can only receive calls on this channel.
Outgoing	The DTE can only initiate calls on this channel.
Two-Way	The DTE can both receive and initiate calls on this channel.

These channel types are significant only during call initiation. Once a virtual circuit has been established, it is always for two-way communication. Typically, only two-way SVCs are used. However, if more than one type is used, the CCITT states that the logical channel numbers must be assigned within the following hierarchy, from the lowest logical channel numbers to the highest:

1. PVCs.
2. Incoming SVCs.
3. Two-way SVCs.
4. Outgoing SVCs.

If a system has only two-way SVCs, then it is possible that at any given time they could all be in use by incoming calls. If the ability to make an outgoing call must to be guaranteed, then an outgoing-only SVC would perform that function.

Network User Address

Each system on an X.25 network has an address to identify it. This address is supplied by the network provider. To ensure that the address is unique across different network providers and between different

countries, the X.121 specification defines an international numbering scheme that ensures a unique DTE address. This address is called the network user address (NUA) . For communication between systems, it is the remote system's NUA that must be known.

Most public networks use the X.121 addressing standard (defined in CCITT Volume VIII.3) to create NUAs. Under the X.121 addressing standard, an NUA consists of the following parts:

- Data Network Identification Code (DNIC)
- National Terminal Number (NTN)

Data Network Identification Code:

A data network identification code (DNIC) consists of 4 digits that include:

- Data Country Code (DCC). The first digit identifies a world geographic zone. The second and third digits identify a specific country.
- Public Data Network Code (PDN). The fourth digit in the DNIC identifies a specific PDN.

> **Note:** Because of the limitation of 1 digit to define only 10 PDNs within one single country, the United States obtained CCITT permission to use 1 digit for the DCC and 2 digits to specify a PDN.

National Terminal Number:

Following the DNIC are 10 digits assigned by the PDN. No rule determines how the national terminal numbers (NTNs) are made up. Most PDNs reserve the last 2 digits as an optional subaddress for the X.25 subscriber. It is the NTN, when communication is made within a given network, that is often given as the system's NUA. The optional subaddress is not processed by the PDN, but is available to identify a finer granularity of address on the remote system. The DNIC is usually used when the remote system is on a different network from the calling system.

The following shows the structure of the network user address (NUA), leaving two digits for the subaddress.

```
Data Network Identification   National Terminal Number   Optional Subaddress      1234     56789012     34
```

The X.121 addressing standard also defines a 1-digit optional prefix for international calls. If a call is beyond PDN boundaries, the user or application establishing the call can add a 0 or a 1 at the beginning of the NUA. If a call is within PDN boundaries, no prefix is necessary. The maximum length of an NUA is 15 digits.

X.25 Usage

Once all three layers are active, the DTE can set up calls and transfer data to other DTEs. To do this, the X.25 protocol uses different types of packets to make or accept the call, transfer data, and end calls. The X.25 communications software performs most of the tasks involved in creating the packets. You do not need to know the contents of each packet, you need only supply the information necessary to create it.

The way calls work on SVCs is different from the way calls work on PVCs. SVCs are more commonly used, and they initiate the DTE to DTE connection with a call-request packet.

The DTE to DTE connection shown in the following diagram illustrates how a call is set up, used, and cleared. The call uses a virtual circuit for the duration of the call. This circuit can be reused once the call is over.

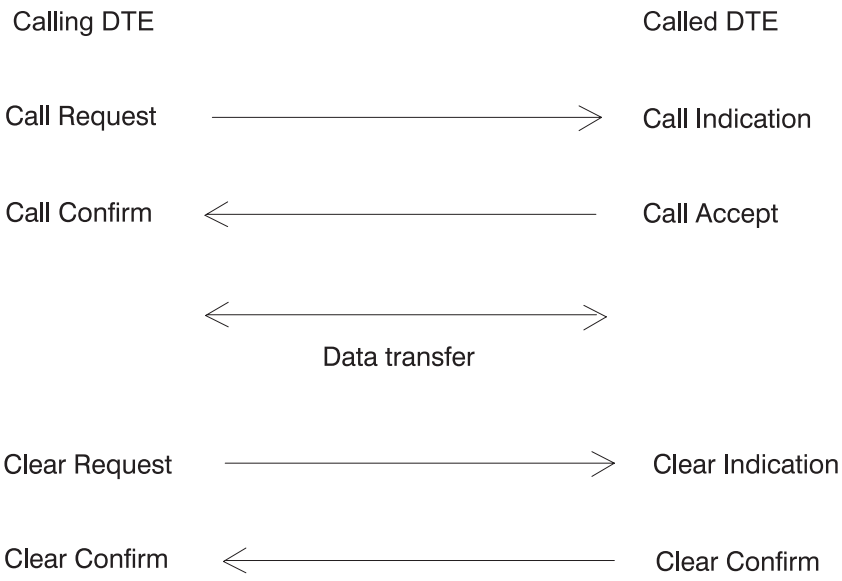


Figure 3. DTE to DTE Connection

Data sent during the call is divided up into units. The size of these units is the *packet size*. Packet size applies to the size of each data packet, not all packets. The default packet size is 128 bytes. When one unit of data needs to be sent that is greater than the packet size, a number of packets are sent. This sequence of packets is marked to indicate that it makes up one unit of data through use of a flag called the *more* or *M* bit. The packet size can be varied from 128 either through configuration or at call setup time. Allowing more data in each packet can have performance gains as there is a degree of overhead for every packet sent. The packet size is one of the call's characteristics that can be changed at call setup. Requests to make changes at this time are controlled by *facility requests*.

Usually the calling DTE is responsible for inserting all the facility requests it needs. Sometimes though, the network inserts the facility requests into the call packet to indicate to the remote DTE certain characteristics of the local DTE, one of which can be packet size. The contents of a packet, when it reaches the called DTE, may be different from the packet that left the calling DTE. Packet contents may change because some information differs for each DTE (for example, logical channel numbers) or only applies to one of the DTEs. The name given to a packet also varies to indicate if it is received or sent.

Packets are grouped into the following categories, according to type:

- Call establishment and clearing:
  - Call request
  - Incoming call
  - Call accepted
  - Call connected
  - Clear indication
  - Clear request
  - Clear confirmation
- Data and interrupt:
  - Data
  - Interrupt
  - Interrupt confirmation
- Flow control and reset:
  - Reset request

  - Reset indication
  - Reset confirmation
  - Receive ready
  - Receive not ready
- Restart:
  - Restart request
  - Restart indication
  - Restart confirmation
  - Diagnostics

Call Setup

The call-setup packets contain different types of information:

- Calling and called addresses.
- Requests related to the characteristics of the call, aimed at either the network (DCE) or remote DTE. These are referred to as *facilities*.
- Data provided by the calling DTE for use by the called DTE when it receives the call. This is referred to as call-user data (CUD).

Call-Time Facilities

A *facility* is a service provided by the X.25 network. Some facilities are offered as options by the network provider. Negotiating packet size, for example, is a standard facility on most networks, while reverse-charging acceptance is optional. Some optional facilities, such as reverse-charging acceptance, are valid for all virtual calls. Other facilities, such as reverse charging, must be specifically requested for the duration of a call. Certain facilities can be allowed or disallowed for a given X.25 port. If not disallowed, the requested facilities can be used during call setup.

The X.25 facilities and their coding are defined in the *CCITT Recommendation X.25*, Sections 6 and 7. The facilities you can use are defined in your X.25 subscription.

To use a valid facility during a virtual call, a facility request and the facility's corresponding parameters must be inserted in the call packet. The X.25 program on the data terminal equipment (DTE) can insert a facility request in the call request (calling DTE) or in the call accepted (called DTE) packet. Coding of the facilities is the responsibility of the X.25 application and not of the application programming interface (API), device driver, or X.25 microcode.

The network DCE can also notify the DTE of the use and parameters of a facility. The DCE inserts a facility indication, either in the incoming-call packet or in the call-connected packet. The following diagram illustrates the call-packet names.

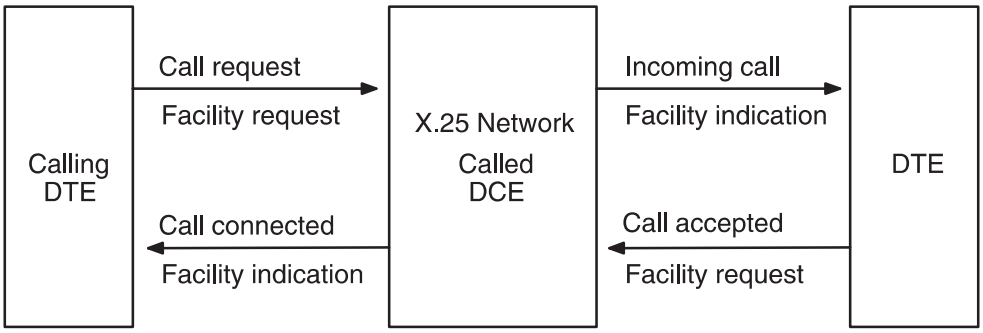


Figure 4. Facilities and Call Packets

The following table lists the main facilities that may be requested by the DTE or the DCE for the duration of a call.

X.25 Optional Facilities			
Facilities	**1980**	**Request**	**Indicator**
Throughput class selection	yes	yes	yes
Flow-control parameters selection	yes	yes	yes
CUG selection	*	yes	
Reverse charging	yes	yes	yes
Fast select	yes	yes	yes
Recognized private operating agency (RPOA) selection	yes	yes	
Network user identification (NUI)			yes
Call redirection notification			yes
Charging requesting service		yes	
Note: *Basic CUG only in CCITT 1980.			

The Throughput Class Selection facility allows you to change your default throughput class (measuring the transmission speed within the network) to a lower value. This action does not affect the DTE-to-DCE speed, only the speed at which a packet traverses the switching nodes in the network.

Flow-control Parameters Facility Consists of the window size selection and the packet size selection.

Closed User Group (CUG) Facility Enables a member of one subgroup to communicate only with other members of the same subgroup. One DTE may be a member of several CUGs. The CUG selection facility allows the DTE to specify what CUG it will be working with. A bilateral closed user group (BCUG) allows calls to be made only between two designated DTEs.

Reverse Charging Facility Allows a DTE to request that the cost of a call be charged to the called DTE.

Fast Select Facility Allows the user 128 bytes of call user data instead of the ordinary 16. Also, 128 bytes may be sent in the call-accepted packet. Fast select must be enabled at subscription time for that particular DTE. With the facility you can make applications that depend entirely on the virtual circuit protocol, clearing virtual circuits as soon as a call-connected indication comes in.

RPOA Selection Facility Selects one or more specific transit networks to carry your virtual circuit.

Network User Identification (NUI) Facility Allows the requesting DTE to provide billing, security, or management information on a per-call basis to the DCE.

Call Redirection Notification Facility Informs the caller that the call has been redirected to another DTE.

Charging Requesting Service Facility Specifies that charging information (such as segment-count, monetary-unit, or call-duration data) is required.

Coding and Decoding Facilities

The following diagram shows the structure of a call packet. Call requests, incoming calls, call-accepted packets, and call-connected packets all have the same structure.

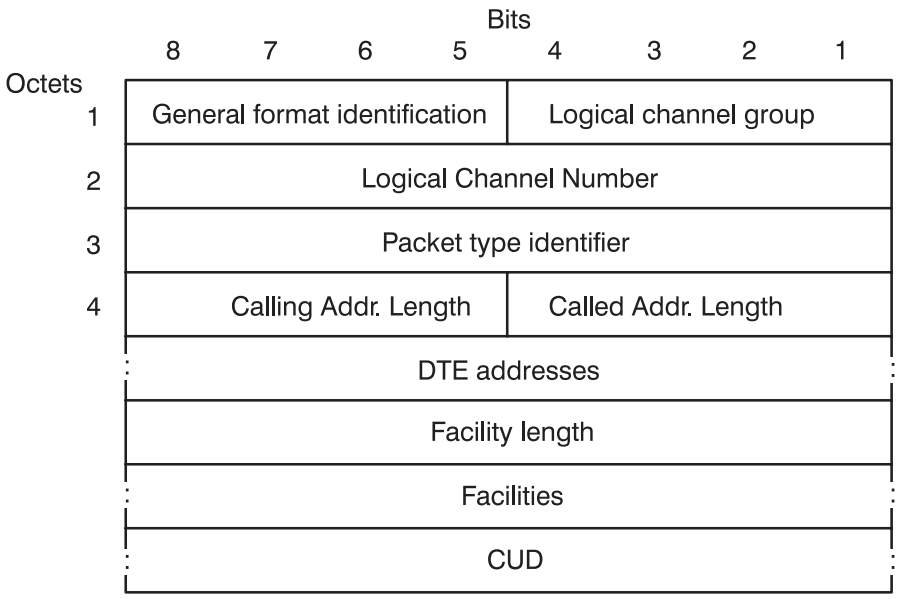


Figure 5. Call Packet Structure

Facility requests or facility indications are inserted between the address block and the call user data (CUD) and are prefixed by an octet containing the total length of the facilities. For information on coding and decoding facilities, see Supported Facilities for X.25 Communications.

Facilities Requested by the DTE

Your application is responsible for coding a facility you want to use and inserting it in the call-request or call-accepted packet. The **xtalk** command, Transmission Control Protocol/Internet Protocol (TCP/IP), and System Network Architecture (SNA) Services allow the user optionally to define a facility request to be inserted in the call packet.

X.25 Network Subscription

The network provider must supply some information about a connection before a user can connect to the network. The network provider assigns the DTE address (NUA), the logical channel numbers, and the types of virtual circuits. Additional attributes must also match between the DTE and the network's DCE.

The network supplier provides the X.25 attachment attributes that must be used to configure the DTE for a particular network subscription. Suppliers create subscriptions based on their network conventions, DCE hardware, and customers' requirements, such as performance, number of concurrent connections, and security needs. The network attachment speed and the DCE hardware determine the adapter and interface choice, so customers should check with the network provider before making these hardware choices.

To configure a DTE, the network provider should be told that the X.25 device driver requires the following attachment characteristics:

- Full-duplex, synchronous transmission.
- Leased-line attachment or dial-up capability through X.32.
- Network or modem-provided clocking.

The X.25 licensed program supports CCITT up to 1988.

Chapter 2. X.25 Licensed Program Functionality

This chapter discusses the functionality of the X.25 Licensed Program interface. The X.25 Licensed Program provides the following functionality:

- Support of the International Consultative Committee on Telegraph and Telephone (CCITT) 1988, 1984, 1980 X.25.
- Packet layer programming interface network provider interface (NPI). New programs written to use X.25 should be written to this interface.
- Frame layer programming interface data link provider interface (DLPI). Programs requiring a point-to-point LAP-B connection should use this interface.
- Compatibility application program interface (API) for applications written to the base Version 3 X.25 support. This API is not intended for new program development. For new program development, refer to sections on NPI or DLPI. Applications written to the base Version 3 X.25 support will need to be recompiled before they can be run on AIX 5.1 with the AIXlink/X.25 LPP COMIO emulation.
- Triple-X (X.3, X.28, X.29) Packet Assembler/Disassembler (PAD).
- Support for Transmission Control Protocol/Internet Protocol (TCP/IP) and Systems Network Architecture (SNA) higher layer protocols.
- Simple network management protocol (SNMP) support for data items from the Management Information Base (MIBs) for the packet and frame layers.

Each of these functional areas is covered in more detail in the relevant chapters.

Configuration Structure

With multi-port adapters, more than one X.25 port can be associated with a given adapter. When the AIXlink/X.25 LPP is configured, X.25 ports are set up to use the available ports on a given communications adapter. The following are supported: X.25 Interface Co-Processor/1 Adapter, X.25 Interface Co-Processor/2 Adapter, IBM ARTIC Portmaster Adapter, IBM 2-Port Multiprotocol Adapter, IBM ARTIC960 Adapter, and IBM ARTIC960Hx 4-Port Selectable PCI Adapter. All of these adapters support the V.24, V.35 and X.21 electrical interfaces. In addition, the ARTIC960 and ARTIC960Hx adapters support the V.36 interface. These adapters produce device instances of the form **ampx**, **ampx**, **apm**, **dpmpa**, **ricio**, and **riciop**, respectively.

For the X.25 Interface Co-Processor/1 , X.25 Interface Co-Processor/2, and the ARTIC Portmaster adapters, the device driver *twd* is associated with the adapter and there can only be one available per adapter. For the ARTIC960 and ARTIC960Hx adapters the *twd* driver is associated with the adapter's *ddricio* instance and there is only one per adapter. With the 2-Port Multiprotocol Adapter there is an *hdlc* device driver for each of the 2 ports and no *twd* device driver. See Appendix H for more information on this adapter.

Once a device driver is configured onto a given adapter, then as many ports supported by the adapter can be configured. The main device used is the X.25 port as shown in the following diagram.

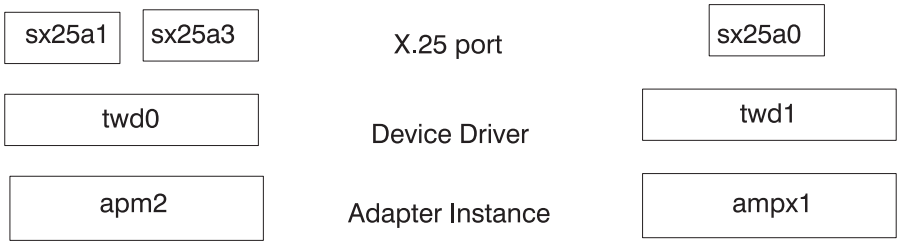


Figure 6. Sample Driver Configuration

When each device is added to the system configuration, it is given the next available instance number. As a result, the device driver's instance number might not match the adapter on which it is configured. The ports and which device driver or adapter is configured on each port can be viewed through **smit** or by entering the **lsx25** command.

Once an X.25 port is available, it can be used by a number of possible applications:

- x29d X.29 daemon
- xspad X.3/X.28 terminal session

  To enable the Triple-X PAD on the system, refer to Managing the Triple-X PAD .
- A COMIO emulator-based application

  To configure a COMIO emulator port on to an X.25 port, refer to Configuring X.25 Communications with SMIT
- TCP/IP

  To configure TCP/IP on an X.25 port, refer to Managing TCP/IP Configuration and Configuring TCP/IP in the *AIX 5L Version 5.2 System Management Guide: Communications and Networks*.
- An NPI based application
- A DLPI based application

AIXlink/X.25 Version 2.0 executes in a 32-bit AIX environment or a 64-bit AIX environment.

32-bit applications are supported in the 32-bit AIX or 64-bit AIX environment.

The 2-Port Multiprotocol PCI Adapter is supported in the 32-bit AIX environment and the 64-bit AIX environment. All other AIXlink/X.25 supported adapters are supported in the 32-bit AIX environment.

Triple-X PAD

A PAD allows remotely attached ASCII terminals to access applications running in X.25 based hosts. Refer to Packet Assembler/Disassembler (PAD) Overview for more details.

The PAD support can be enabled or disabled for the system, refer to Configuring X.25 Communications with SMIT . If enabled, one instance of X.29 supports all the configured X.25 ports, servicing all remote terminals using the PAD. Where the system is being used as a terminal PAD, each terminal has its own X.3/X.28 session.

COMIO

Each X.25 port can have COMIO emulation enabled. This emulation produces a programming interface compatible with that in the base AIX Version 3 X.25 support. Applications such as **xtalk** can use the emulation without recompiling. To allow for this emulation, a device entry is created with the same device name that would have been generated by the base AIX Version 3 X.25 support - `/dev/x25s`. Refer to Configuring X.25 Communications with SMIT for details on using this emulation.

AIX **Base Version 3 X.25 users**	Not all applications supplied with Version 3 are supported. The **xmanage** command is no longer needed as the line is continuously attempted to be brought up. The **xmonitor** trace utility has been replaced with **x25mon**. The **x25mon** trace utility has a ″control″ tracing option that provides information such as the state of the various layers and the contents of bad frames or packets.

TCP/IP

Transmission Control Protocol/Internet Protocol (TCP/IP) support can be added to any of the X.25 ports. The system allows IP addresses to be matched to an NUA. If an IP connection is required to a given remote X.25 DTE, then a virtual circuit is acquired and the IP data is sent. A typical TCP/IP port would be `xs0`. The instance number of the TCP/IP port is not necessarily the same as the X.25 port on which it is configured. Refer to Configuring X.25 Communications with SMIT and to Configuring TCP/IP in the *AIX 5L Version 5.2 System Management Guide: Communications and Networks* for details.

NPI

Network Provider Interface (NPI) provides a programming interface for the packet layer. Refer to NPI Overview for information on its use. Each instance of an NPI-based application accesses the NPI module as required. There are no separate configuration steps for NPI.

DLPI

Data link provider interface (DLPI) provides a programming interface to the frame, LAP-B, layer. Refer to DLPI Overview for information on its use. Though the access is to LAP-B rather than X.25, an X.25 port must be configured and set up to allow use of the DLPI interface. When DLPI is enabled, X.25 access to the port is disabled. Refer to “General Parameters” on page 32 for details.

> **Note:** See Appendix I, “Using AIXlink/X.25 over the IBM 2-Port Multiprotocol Adapter,” on page 327 for information on differences when using 2-Port Multiprotocol Adapters.

STREAMS

The X.25 Licensed Program works under the STREAMS environment. Refer to the section on STREAMS in the *AIX 5L Version 5.2 Communications Programming Concepts* for more details on how this works.

When an X.25 port is configured, the various modules of supporting code are loaded into the STREAMS environment as shown in the following diagram. Then the driver and modules are pushed to create the protocol stack.

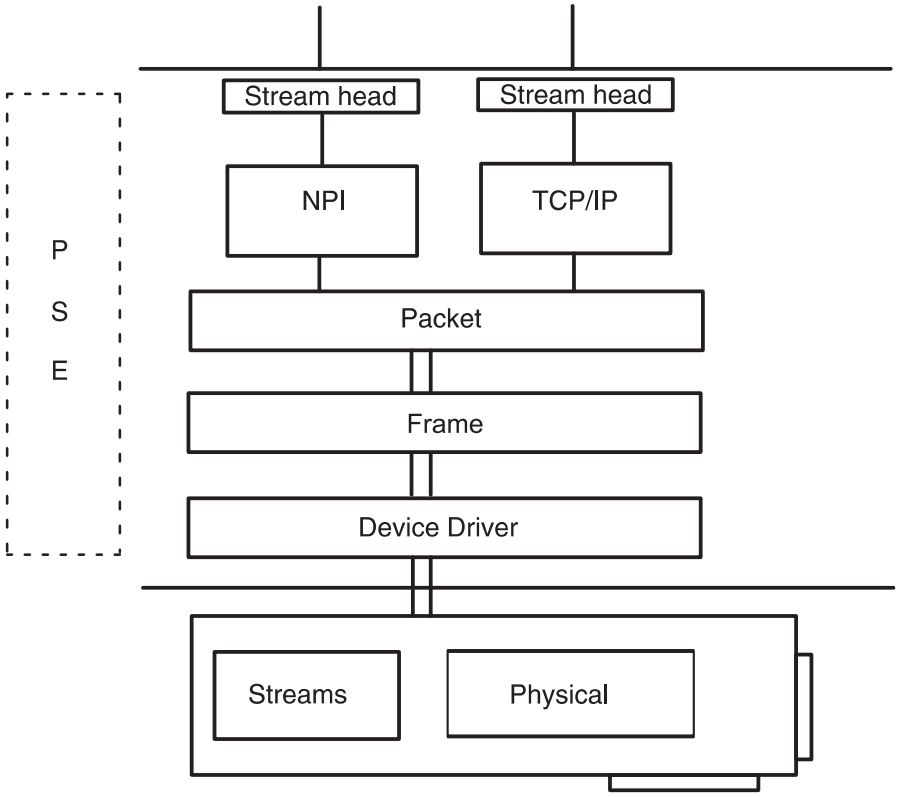


Figure 7. X.25 in the STREAMS Environment

Once the system is configured for use, the STREAMS setup happens automatically. Applications of NPI and the instances of the X.3/X.28-based PAD require that an instance of the module be pushed onto the stream. This is taken care of by the application.

> **Note:** See Appendix I, “Using AIXlink/X.25 over the IBM 2-Port Multiprotocol Adapter,” on page 327 for information on how the x.25 modules are loaded into STREAMS when using 2-Port Multiprotocol Adapters.

Chapter 3. X.25 Migration

This chapter describes the differences between X.25 Support on AIX Version 3, AIX Version 4, and AIX Version 5.

X.25 Functional Comparison

The following table shows the main characteristics and differences between X.25 support on AIX Version 3, AIX Version 4, and AIX Version 5.

	AIX Version 3 BOS	**AIXlink/X.25 V1.1**	**AIXlink/X.25 V2.0**
Operating System	AIX 3.2	AIX 4.1.5, AIX 4.2.1, and AIX 4.3.0	AIX 5L Version 5.1 or later
Packaging	Basic X.25 support in BOS	X.25 support through the Licensed Program. Selectable v.c. capacity: **Entry** Up to 4 **Basic** Up to 16 **Extended** Up to 64 **Advanced** Up to 256 **Full** Unrestricted	Same as V1.1 except entry level option is deleted
APPLICATIONS:			
TCP/IP	X (xt#), x25xlate	X (xs#), X25ip	Same as V1.1
PAD support	3rd Party	x	Same as V1.1
SNMP agent /MIB	-	X (x25smuxd) (subsets of RFCs 1381 and 1382)	Same as V1.1
NPI (packet layer NPI)	-	x	Same as V1.1
DLPI (frame layer API)	-	x [Note 1]	Same as V1.1
Application Compatibility	-	COMIO emulation provides X.25 access through /dev/x25s# for X.25 BOS applications and commands	Same as V1.1
SNA	x	- (Available in 4.1 and later platforms)	Same as V1.1
AIX Version 3 X.25 API	x	COMIO emulation	Same as V1.1
COMMANDS:			
xcomms	x	-	Same as V1.1
xroute	x	COMIO emulation	Same as V1.1
xtalk	x	COMIO emulation	Same as V1.1
xmanage	x	-	Same as V1.1
x25 line trace	**xmonitor** -packet -frame x25s#	**x25mon** -p -f -n sx25a#	**x25mon** -t -c -p -f -n sx25a#
INSTALLATION:			
	AIX Version 3 BOS	**AIXlink/X.25 V1.1**	**AIXlink/X.25 V2.0**

	AIX Version 3 BOS	AIXlink/X.25 V1.1	AIXlink/X.25 V2.0
SMIT Installation	All X.25 software is installed at once	Selectable Install: • **All** • **TCP/IP** Note 2 • **Runtime** Note 3 • **NPI** • **Triple-X PAD** • **COMIO**	Same as V1.1
Adapter microcode	Copy from Options disk	Installed with Licensed Program	Same as V1.1
MISCELLANEOUS:			
Driver names	x25s# (device driver)	twd# Note 4 (streams device driver)	Same as V1.1
X.25 port names	x25s#	sx25a#	Same as V1.1
CCITT Conformance	1980, 1984	1980, 1984, 1988	Same as V1.1
HARDWARE:			
X.25 Interface CoProcessor/2 (Micro Channel)	x25s# (until 3.2.3e) ampx# (from 3.2.3e)	X ampx#	Same as V1.1
ARTIC PortMaster/A: V.24 (8-port fanout) V.35 (6-port fanout) X.21 (6-port fanout)	-	X Note 5 amp#	Same as V1.1
ARTIC960: V.36 (6-port fanout) X.21 (8-port fanout) EIA-232E (8-port fanout)	-	ricio# Note 6	Same as V1.1
X.25 Interface CoProcessor (ISA bus)	-	X Note 7 ampx#	Same as V1.1
2-Port Multiprotocol PCI Adapter	-	X dpmp#	Same as V1.1
IBM ARTIC960Hx 4-Port Selectable PCI Adapter	-	riciop# Note 8	Same as V1.1
Maximum adapters per system	4 per bus Note 9	Limited to 8 adapters per system.	16/System except ARTIC960Hx 14; ARTIC960MCA 8
Maximum virtual circuits	64 virtual circuits	512 total Virtual Circuits per port, 1024 total Virtual Circuits per board (512 on a single port Note 10) and 4096 total Virtual Circuits per system.	Same as V1.1
Packets Per Second (pps) 128-octet packets Note 11	35 pps	100 pps per X.25 Interface adapter, 200 pps per PortMaster adapter, 1000 pps per ARTIC960 adapter, 650 pps per 2-Port Multiprotocol adapter, 1000 pps per ARTIC960Hx 4-Port Selectable PCI Adapter.	Same as V1.1
MODEM CABLE INTERFACE / MAXIMUM SPEEDS:			
X.21 (15-pin)	64 Kbps	64 Kbps Note 12	Same as V1.1

	AIX Version 3 BOS	AIXlink/X.25 V1.1	AIXlink/X.25 V2.0
V.24 (24-pin)	19.2 Kbps	19.2 Kbps	Same as V1.1
V.35 (34-pin)	64 Kbps Note 13	64 Kbps Notes 12,13	Same as V1.1
NETWORK ATTACHMENT(synchronous, full duplex, network/modem clocking):			
Dedicated leased line	x	x	Same as V1.1
Dial-up (X.32, V.25bis)	-	x	Same as V1.1

Notes:

1. A port cannot be used for running both frame and packet layer applications - DLPI interface is enabled via SMIT.
2. To run TCP/IP over X.25, the user must also install the TCP/IP software component.
3. The Runtime software component is the minimum required for X.25 - all other components require the runtime software.
4. The **twd** streams device driver is not used for X.25 ports configured over the 2-Port Multiprotocol PCI Adapter. Please refer to Appendix H. Using AIXlink/X.25 over the IBM 2-Port Multiprotocol Adapter for more information.
5. ARTIC PortMaster support is made of three components: a base PortMaster adapter with at least 1 MB memory, a V.24, V.35, or X.21 electrical interface board/daughter card (EIB), and the matching interface cable/fanout box.
6. ARTIC960 support is made of three components: a base ARTIC960 adapter with at least 4 MB memory, a V.24, V.36, or X.21 application interface board/daughter card (AIB), and the matching interface cable/fanout box.

   Note: The ARTIC960 adapter is supported on AIXlink/X.25 1.1.3 (and later).
7. The ISA and microchannel bus versions of the X.25 Interface Co-Processor adapters use the same modem cables (V.24, V.35, X.21).
8. The IBM ARTIC960Hx 4-Port Selectable PCI Adapter is supported on AIXlink/X.25 1.1.5 and later. The ARTIC960Hx support is made of three components: a base ARTIC960Hx adapter with at least 8 MB of memory, a 4-Port selectable daughter card, and a cable assembly with the desired electrical interface.

   Note: The IBM ARTIC960Hx 4-Port Selectable PCI Adapter does not support circuit 125 (ring indicate). Therefore, this adapter cannot receive incoming calls, using the V25bis direct mode (108.1).
9. Up to eight adapters are supported on systems with dual Micro Channel. Each microchannel supports 4 adapters.
10. 512 virtual circuits is the recommended maximum because the X.25 Interface Co-Processor adapters have only one port.
11. Packet per second (pps) values assume full 128-octet packets and were measured at the packet level.
12. The ARTIC960 and ARTIC960Hx Adapters support speeds up to 2MB on the V.35/V.36 and X.21 electrical interfaces. The 2-Port Multiprotocol adapter supports speeds up to 2 MB on the V.35 and V.36 electrical interfaces and up to 1.544 MB on X.21 electrical interfaces.
13. The CCITT V.35 specification defines 56 Kbps as the maximum line speed.

Differences Between X.25 Licensed Programs Version 2.0 and Version 1.1

If you are migrating from X.25 V1.1 to X.25 V2.0, read the following. This section explains what hardware, functionality, configuration and setup procedures have been changed with the new X.25 Licensed Program.

Hardware Differences

AIXlink/X.25 Version 2.0 continues to support the same adapters that were supported with Version 1.1:

- X.25 Co-Processor/2 adapter (FC 2960)
- X.25 Co-Processor ISA-bus adapter (FC 6753), only on a machine with AIX 4.1 or greater, an ISA-bus, and the AIXlink/X.25 Licensed Program.
- RIC Portmaster Adapter/A with 1MB (FC 7006)
- RIC Portmaster Adapter/A with 2MB (FC 7008)
- RIC Portmaster Adapters with 1Mb and 2Mb can now be used to connect to an X.25 network. They are multiport adapters which support V.24, V.35 and X.21 interfaces.
- ARTIC960 V.36 with 4MB (FC 2935)
- ARTIC960 X.21 with 4MB (FC 2938)
- ARTIC960 EIA-232E (V.24) with 4MB (FC 2929)
- 2-Port Multiprotocol PCI Adapter (FC 2962), only on AIX 4.1.5, AIX 4.2.1, AIX 4.3.0, or greater machines with PCI-bus and AIXlink/X.25 Licensed Program.
- IBM ARTIC960Hx 4-Port Selectable PCI Adapter (FC 2947) is only supported on machines with AIX 4.1.5, AIX 4.2.1, and AIX 4.3.1 or higher with the AIXlink/X.25 LPP version 1.1.5 (or higher).

X.25 Supported Adapters

Adapter	AIX Version 3Base X.25	X.25 Licensed Program V1.1 and V2.0
X.25 Co-Processor/2	supported	supported
X.25 Co-Processor ISA-bus	not supported	supported
RIC Portmaster Adapter/A 1MB	not supported	supported
RIC Portmaster Adapter/A 2MB	not supported	supported
ARTIC960 4MB	not supported	supported
2-Port Multiprotocol PCI Adapter	not supported	supported
IBM ARTIC960Hx 4-Port Selectable PCI Adapter	not supported	supported

Additions and Functionality Differences in V1.1.5

The X.25 Licensed Program continues to provide the following features that were provided in V1.1:

- Support of the International Telegraph and Telephone Consultative Committee (CCITT) 1988 X.25.
- Packet Layer Programming Interface Network Provider Interface (NPI).
- Frame Layer Programming Interface Data Link Provider Interface (DLPI).
- COMIO emulation provides X.25 access through /dev/x25s# for AIX Version 3 base X.25 applications. These applications must be recompiled before they can be run over the AIXlink/X.25 protocol stack.
- Triple-X (X.3, X.28, X.29) Packet Assembler/Disassembler (PAD).
- Simple Network Management Protocol (SNMP) support for data items from the Management Information Base (MIB) for the packet and frame layers.
- V25bis support (Not on ARTIC960Hx 4-port selectable PCI adapter).
- Support for up to 512 logical channels per line.
- Automatic DTE configuration.
- Support for an aggregate sustained rate of 200 128-bytes packets per second (measured at the packet layer API) for each adapter in the system.

Additions and Functionality Differences in Version 2.0

- Up to 16 adapters supported per system. ARTIC960Hx PCI has a limit of 14 adapters per system. ARTIC960 MCA has a limit of 8 adapters per system.

- AIXlink/X.25 Version 2.0 continues to be able to execute in a 32-bit environment. AIXlink/X.25 Version 2.0 has been enhanced to execute in a 64-bit environment. The 2-Port Multiprotocol PCI adapter is supported in the 64-bit AIX environment and continues to be supported in a 32-bit AIX environment.
- NPI has been enhanced to return CCITT cause and diagnostic codes to NPI applications.
- The entry level product has been eliminated.
- The frame layer code of AIXlink/X.25 no longer executes on the adapter for those adapters which use the **twd** driver. Instead, the frame layer code now executes in the kernel as the frame layer for the 2-Port Multiprotocol adapter has done.

Functionality differences

Functionality	**AIX Version 3 Base X.25 Support**	**X.25 Licensed Program V1.1 and V2.0**
CCITT Conformance	1984	1988
NPI	NO	YES
DLPI	NO	YES
API Library	YES	YES Note 1
PAD Support	NO Note 2	YES
SNA Support	YES	YES
TCP/IP Support	YES	YES
SNMP Support	NO	YES
Automatic DTE Configuration	NO	NO
Logical Channels per line	64	512
Packets per second (128-bytes packets)	35	200

Notes:

1. Applications based on the API library are only supported when the COMIO emulator is configured on the port.
2. Only with third-party software.

Packet Layer

The main differences between CCITT 1984 and 1988 X.25 recommendations at the packet level are:

Network User Identification (NUI)
Network User Identification (NUI) related facilities are divided into three facilities:
- NUI_subscription
- NUI_override
- NUI_selection

DTE/DTE Operation
DTE-to-DTE operation without an intervening network is defined. In this situation, one DTE must act as DCE. The DTE acting as DCE at packet layer may be acting as DTE at Data Link Layer and vice versa. This is an optional facility.

Circuit-switched Connection without Prior Agreement
A circuit-switched connection without prior agreement (such as electronic mail-order) is defined and default values specified for all applicable parameters. This is an optional capability.

Throughput Class of 64000 bits/s
A new throughput class of 64000 bits per second is defined. This is an optional capability.

Address Block Definition
A new address block is defined for call setup and clearing packets that allows addresses of 12 or 15 digits. This is an optional capability.

TOA/NPI Address Subscription
A new facility, TOA/NPI_Address_Subscription, is added to accommodate E.164 (ISDN) addresses of up to 17 digits in length. This addition results in a redefinition of the address block and the consequent definition of new formats for the CALL_REQUEST, CALL_ACCEPTED, CALL_CONNECTED, CLEAR_REQUEST, CLEAR_INDICATION and CLEAR_CONFIRMATION packets. This is an optional capability.

Call Deflection
Call_Deflection_selection facilities whereby the DTE forwards calls after receiving an INCOMING_CALL packet (unlike CALL_REDIRECTION that is handled in the network and the originally called DTE never receives an INCOMING_CALL packet) is added. There are three call deflection facilities:

- Call_Deflection_Subscription enables the DTE to request a Call_Deflection_Selection.
- Call_Deflection_Selection may be used on a per-virtual-call basis only if Call_Deflection_Subscription is subscribed to.
- Call_Deflection_Notification which informs the alternate DTE that the call is forwarded.

These are optional user facilities.

Priority Facility
A priority facility specifies the priority of data on a connection, and priority to keep a connection. This is an optional capability.

Maximum size of Called and Calling Address Extension
The maximum size of the called and calling address extension fields is extended from 32 to 40 digits, and an OSI/non-OSI indicator has been added. Support of the larger address is optional. However, a test of the OSI/non-OSI indicator is required to determine the size of the called and calling address.

Recognized Private Operating Agency RPOA related facilities are subdivided into:

- RPOA_Subscription applies to all virtual calls.
- RPOA_Selection applies to a given virtual call and does not require RPOA_Subscription.

These are optional capabilities.

Mandatory Address Length Fields in CALL_ACCEPTED packets
The use of the Address Length Fields in CALL_ACCEPTED packets is mandatory, even if they are set to zero.

Mandatory Facility Length Fields in CALL_ACCEPTED packets
The use of the Facility Length Fields in CALL_ACCEPTED packets is mandatory, even if they are set to zero.

Virtual Circuit Clearing/Resetting Failure
When a CLEAR_REQUEST packet is not confirmed within time-limit T23, the DTE will retry the call-clearing procedure up to R23 times, at T23 intervals, before notifying the higher layer (virtual circuit user) of the failure; leaving the logical channel in the DTE_CLEAR_REQUEST rate (p6) rather than placing the logical channel in an inoperative state as specified by early versions.

When a RESET_REQUEST packet is not confirmed within time-limit T22, the DTE will retry the resetting procedure up to R22 times, at T22 intervals, before notifying the higher layer (virtual circuit user) of the failure; leaving the logical channel in the DTE_RESET_REQUEST state (d2) rather than placing the logical channel in an inoperative state as specified by early versions.

Frame Layer

The main differences between CCITT 1984 and 1988 X.25 recommendations at the frame level are:

DTE/DTE Operation
Although not specified in CCITT Recommendation X.25, International Standard Organization ISO 7776 supports communication between two DTEs without an intervening network. Since there is no intervening network, link layer characteristics must be made by bilateral agreement rather than at subscription time. This is an optional capability but is required for communication using Open Systems-Interconnect (OSI).

Clearing a FRMR Condition at the DCE	After the DCE has transmitted a FRMR response, the frame rejection condition is cleared when the DCE receives a FRMR response (in addition to when a SABM/SABME, DISC or DM is sent or received).
Maximum Number of Outstanding I-Frames	American National Standards ANS X3.100 specifies that all networks must support k=7. K is the maximum number of outstanding I-frames.

Installation, Configuration, and Setup Differences

The X.25 entry level package of 4 SVCs or less (the X.25 Lite Package) is no longer available.

Since X.25 is a licensed program, the first step in your configuration and setup procedure is to install the X.25 code.

The next step is to configure the device driver and the X.25 port. As soon as the port is configured and available, the X.25 licensed program software continuously tries to bring up the connection to the X.25 network.

You'll probably need to change some attributes such as number of virtual circuits and throughput. If you are going to use SMIT fast paths, be aware that many of them have changed.

SMIT Fast Path Differences

Parameter	AIX V.3 BOS	V1.1 and V2.0
Change / Show X.25 General Parameters	x25csg	x25str_mp_csp_g_sel
Change / Show X.25 Frame Parameters	x25csf	x25str_mp_csp_f_sel
Change / Show X.25 Packet Parameters	x25csp	x25str_mp_csp_p_sel

The **xroute** command works only with X.25 ports that have COMIO emulation configured and must be used only when you have applications which use this emulation (such as xtalk and SNA).

The X.25 licensed program allows you to enable or disable only these two facilities:

- Fast Select.
- Reverse Charging.

Command Differences

Some commands that were used for management and configuration purposes have changed.

Differences Between xmonitor and x25mon Commands

	AIX V3 BOS xmonitor	V1.1 x25mon	V2.0 x25mon
Frame Layer	-frame	-f	-c, -t
Packet Layer	-packet	-p	
Port	x25s n	-n sx25a n	

Attributes Differences

The following table shows the differences in attribute names.

Attribute Names

AIX Version 3 X.25 Support	V1.1 and V2.0
num_in_out_svc	bi_vc_num
in_out_svc	bi_vc_start
ccitt_support	ccitt
connection_mode	connect_seq

Attribute Names

AIX Version 3 X.25 Support	V1.1 and V2.0
d_bit	d_bit_accept
pvc_d_bit	def_pvc_d_bit
def_rx_pkt_size	def_rx_size
def_rx_through	def_rx_th
def_rx_pkt_win	def_rx_win
def_tx_pkt_size	def_tx_size
def_tx_through	def_tx_th
def_tx_pkt_win	def_tx_win
frame_modulo	f_modulo
fast_select	fs_mode
num_in_svcs	in_vc_num
in_svc	in_vc_start
line_type	line_type
local_nua	local_nua
max_rx_pkt_size	max_rx_size
max_rx_pkt_win	max_rx_win
max_tx_pkt_size	max_tx_size
max_tx_pkt_win	max_tx_win
n2_counter	n2
network_id	network_id
num_out_svcs	out_vc_num
out_svc	out_vc_start
pkt_modulo	pkt_modulo
num_of_pvcs	pvc_num
pvc_channel	pvc_start
rev_charging	rev_charge
t1_timer	t1
t21_timer	t21
t22_timer	t22
t23_timer	t23
t24_timer	t24
t25_timer	t25
t26_timer	t26
t4_timer	t4
zero_address	zero_address

Note: All timers in the AIXlink/X.25 LPP are defined in seconds.

Default Values of Important Parameters

Some default values of port parameters have changed. The following table shows the differences among the AIX Version 3 Base X.25 support, V1.1, and V2.0.

Default Values of Important Parameters

Parameter	AIX Version 3 Base X.25 Defaults	X.25 Licensed Program Defaults V1.1	X.25 Licensed Program Defaults V2.0
Frame window size	7	7	7
Frame modulo	8	8	8
Packet Modulo	8	8	8
CCITT support	1980	1984	1988
Default receive packet size	128	128	128
Default transmit packet size	128	128	128
Default receive packet window	2	3	3
Default transmit packet window	2	3	3
Default receive throughput class	9600	64000	64000
Default transmit throughput class	9600	64000	64000

Use of the System Error Log

The AIXlink/X.25 LPP does not use the System Error Log to report problems. See X.25 Problem Determination for information on how to debug X.25 problems when using the AIXlink/X.25 LPP.

Chapter 4. X.25 Installation and Configuration

The following information explains what you need to install the X.25 program and how to configure your system for X.25 communications.

Before you install the X.25 program, ensure that your system meets the minimum software and hardware requirements.

Minimum Requirements

Software

You need the following software:

1. AIX 5L Release 5.1 with the 5100–01 Recommended Maintenance package or later, which is included on the 9/2001 or later AIX Update CD.
2. AIXlink/X.25 Version 2.0 licensed program or higher (feature 5765–E85).

Hardware

You need the following hardware:

1. A microchannel system with at least one of the following:
   a. ARTIC Portmaster Adapter/A 8-port V.24
   b. ARTIC Portmaster Adapter/A 6-port V.35
   c. ARTIC Portmaster Adapter/A 6-port X.21

      Note: The ARTIC Portmaster Adapter/A adapters are made up of a processor board, along with an add-on electrical interface board (EIB). The EIB requires you to select the type of line interface to be used. The adapters should have 1 or 2 MB of memory.

   d. ARTIC960 8-port EIA-232E (V.24)
   e. ARTIC960 6-port V.35/V.36
   f. ARTIC960 8-port X.21

      Note: The ARTIC960 adapters are made up of a processor board, along with an add-on interface board (AIB). The AIB type determines the line interface. The adapters must have 4MB of memory.

   g. X.25 Interface Co-Processor/2.
2. A POWER-based Personal Computer with the following:
   a. X.25 Interface Co-Processor/1.

      Notes:

      1) The POWER-based model must have ISA slots for this adapter to be used.
      2) The interrupt level of this adapter is set using switches on the adapter. The interrupt level should be unique for each adapter in the system. Refer to the system unit's documentation for more details.

   b. IBM 2-Port Multiprotocol PCI Adapter
   c. IBM ARTIC960Hx 4-Port Selectable PCI Adapter

Planning Your X.25 Installation

The best method for data transfer over a wide area network depends on a number of factors. Among the common factors are the amount of data traffic, the frequency that data is sent, response time required, and so forth. Along with these, the tariff available for the network providers can play an important part in protocol choice.

Once X.25 is decided upon, the specifics of the network subscription must be planned along with the network provider. Switched virtual circuits can be used by different applications, possibly to different remote destinations, at different times. Permanent virtual circuits are dedicated to a given remote destination. Depending on the type of X.25 traffic that is generated, the number and speed of X.25 lines must be decided, along with the number of virtual circuits, and so forth.

With the characteristics of the X.25 interfaces defined, X.25 adapters to support these interfaces must be added to the system. Choice of adapter and line interface depends on the characteristics of the network subscription.

The disk space needed to install the AIXlink/X.25 V2.0 packages is:

sx25	6.4 Megabytes for the package
sx25.adt	1.1 Megabytes for the package
sx25.html	7 Megabytes for the package

These numbers are current as of the product shipment in September 2001.

> **Note:** If your X.25 applications use the Data Link Provider Interface (DLPI), and you are migrating from AIXlink/X.25 Version 1.1.3 or a prior version of the product, you may need to make changes to your DLPI applications. The changes are necessary to allow your applications to make use of all adapters supported by the AIXlink/X.25 product. If your DLPI applications are not configured to use the IBM 2-Port Multiprotocol PCI adapter, then the changes are not strictly required. See Chapter 6, DLPI Overview, for more information.

Installation Procedure

The X.25 licensed program is delivered as an install-image. To install the X.25 program:

1. Read Planning Your X.25 Installation.

   The files making up the licensed program are divided into a number of sets that may be installed independently. Installation of the X.25 run time system is a prerequisite for installation of any of the other X.25 options. When installing the X.25 files, the language environment should be set to en_US. The following options are available:

   - All
   - COMIO Compatibility Support and Applications
   - NPI and DLPI Support
   - Runtime Environment
   - TCP/IP Support
   - Triple-X Packet Assembler/Disassembler (PAD)
   - Applications such as SNA and XTALK the COMIO Compatibility support.
2. Follow the instructions included with the program package.
3. Use the **installp** command or SMIT to install the program.

Hardware Installation

The adapters should be installed as described in the system unit's documentation.

- The **IBM ARTIC960Hx 4-Port Selectable PCI Adapter** attaches to a special breakout cable which converts the one connector on the adapter to the 4 ports supported. The physical interface used on all 4 ports is selected by the type of cable used. From the breakout cable, the ports are attached to the line interface equipment.

  Note: Prior to using the AIXlink/X.25 on the ARTIC960Hx adapter, install the IBM ARTIC960 Support Program for RISC System/6000 software (devices.artic960.rte). This software is shipped on a diskette packaged with the adapter and must be version 1.4.3 or later. When running X.25 on an ARTIC960Hx adapter, do not directly run any other ARTIC960Hx software on that adapter, including the programs in **/usr/lpp/devices.artic960/bin**.

- The **ARTIC960 Adapters** attach to a special breakout cable, which converts the one connector on the adapter to the 6 or 8 ports supported. This cable must be the same interface type as the AIB attached to the adapter. From the breakout cable, the ports are attached to the line interface equipment.

  Notes:

  1. Prior to using the AIXlink/X.25 on the ARTIC960 adapter, install the IBM ARTIC960 Support Program for RISC System/6000 software (devices.artic960.rte). This software is shipped on a diskette packaged with the adapter and must be version 1.3.4 or a later 1.3.x version. When running X.25 on an ARTIC960 adapter, do not directly run any other ARTIC960 software on that adapter, including the programs in **/usr/lpp/devices.artic960/bin**. The ARTIC960 adapter is not supported on 7006- and 7011-type systems.
  2. The ARTIC960 EIA-232E (V.24) AIB can use the same fanout boxes and cables as those used for the Portmaster Adapter/A. The ARTIC960 X.21 AIB cannot. If the ARTIC960 V.36 AIB is being connected to a V.35 network, the same fanout box and cables can be used as those for the Portmaster Adapter/A. If the ARTIC960 V.36 AIB is being connected to a V.36 network, the ARTIC960 6-port V.36 cable must be used.

- The **2-Port Multiprotocol PCI Adapters** provide two interface ports, the physical interface type being selected by the type of cable used. This adapter supports four different cabling options, corresponding to the following physical line interface types:
  - RS-232
  - V.35
  - V.36
  - X.21

  Please order the appropriate cable for your interface equipment.

  Note : Prior to using AIXlink/X.25 on the 2-Port Multiprotocol PCI Adapter, install devices.common.IBM.hdlc and devices.pci.331121b9.

- The **Portmaster/A Adapters** attach to a special breakout cable, which converts the one connector on the adapter to the 6 or 8 ports supported. This cable must be the same interface type as the EIB attached to the adapter. From the breakout cable, the ports are attached to the line interface equipment. Prior to using AIXlink/X.25 on the Portmaster/A Adapter, install devices.mca.8f70.
- The **X.25 Co-Processors** provide one interface port, the physical interface type being selected by the type of cable used.

  Note:

  - Prior to using AIXlink/X.25 on the Co-Processor/1, install devices.isa.c1x.
  - Prior to using AIXlink/X.25 on the Co-Processor/2, install devices.mca.eff0.

- With the **X.25 Co-Processor/1**, the adapter's interrupt level must be set so it will not conflict with other adapters in the system. Refer to the system's documentation for details of the interrupt setup.

Configuring X.25 Communications with SMIT

The easiest way to configure the X.25 software is through SMIT. When you are configuring the system, options are presented based on what adapters are available in the system.

> **Note:** It is best if all the adapters to be used are installed in the system before the X.25 configuration is begun. Adapters can be added or deleted at a later time, but for the instance numbers to be kept in a simple order, as much of the configuration as possible should be done at one time.

The main areas of configuration are:

- Adding and deleting adapter device drivers.
- Adding, deleting, and changing X.25 ports configured to an adapter.

From a system where the adapters are installed, the following steps outline those necessary to configure X.25 ports ready for use:

1. Configure the total number of virtual circuits allowed on the system. This is specified on the X.25 license.
2. Add device drivers to each adapter to be used. If you use the IBM 2-Port Multiprotocol PCI Adapter, you must add the **hdlc** device driver. Otherwise, you must add the **twd** device driver.
3. Add X.25 ports to each of the adapter's physical ports.
4. Change the configuration parameters on any ports where the settings must be different from the defaults.
5. Add COMIO emulation to any of the X.25 ports where this is required. Refer to Managing COMIO Emulation .
6. Add TCP/IP support to any of the X.25 ports where this is required. Refer to Managing TCP/IP Configuration .
7. Start Triple-X PAD if it is used on the system. Refer to Managing the Triple-X PAD .

Initial SMIT Path

The X.25 software is configured based on the adapter it is to run on. To reach the communication devices screen:

1. Enter: `smit`
2. Select **Devices**.
3. Select **Communication**.

   Under the communication devices list are entries for the adapters supported in the system. Select the adapter type required and a list of the available software packages for that adapter type is listed. Depending on the adapter, select one of the following:

   - **IBM ARTIC960 Adapter**.
   - **Portmaster Adapter/A**.
   - **X.25 Co-Processor/2 or Multiport/2 Adapter**.
   - **X.25 Co-Processor/1**.
   - **IBM 2-Port Multiprotocol PCI Adapter**. (See Note below.)
   - **IBM ARTIC960HX PCI Adapter**.
4. Select **Adapter**.
5. Select **Manage Device Drivers** for the adapter type selected.
6. Select **Manage X.25 Licensed Program Product Device Driver**

   > **Note:** After you select **IBM 2-Port Multiprotocol PCI Adapter** do the following:
   >
   > 1. Select **Manage HDLC Device Driver**

2. Select **Manage Additional Protocols / Emulators**
3. Select **Manage X.25 over HDLC Device Driver**

Managing Device Drivers

The Manage X.25 Licensed Program Product Device Driver allows the instances of the `twd` device driver to be managed. If a driver is in the defined state, selecting **Configure a Defined Device Driver** makes it available, assuming that no other driver is configured at the adapter. The device driver is added to an adapter before the X.25 ports on that adapter are added, and is removed after all the X.25 ports on the adapter are removed.

Managing Ports

Select **Manage X.25 Ports** to manage the X.25 ports on the system, from the Licensed Program's initial SMIT screen. This brings up the Manage X.25 Ports SMIT screen. From the Manage X.25 Ports SMIT screen, the following actions can be taken:

List All Defined Ports
List all the ports configured on the system, either in the defined or available state.

Add a Port
Add a new port definition. This option sets all defaults for the port depending on the country prefix you choose. If you do not choose a country prefix, port defaults are set to a universal setting.

Move a Port Definition
The definition of a given port can be moved to a different port, possibly on a different adapter.

1. Select the port to be moved.

   Note: For the IBM 2-Port Multiprotocol adapter, only those ports currently in the defined state may be moved.
2. Select the new parent adapter driver.
3. Enter the new port number.

   Note: For the IBM 2-Port Multiprotocol adapter, the port number is automatically filled in for you.

Change/Show Characteristics of a Port
The characteristics of the port are controlled by a number of attributes, divided up in SMIT into a number of screens. This section gives access to those attribute screens.

Remove a Port
The port is removed from the available state back to the defined state. When a port is in the defined state, the port's configuration information is kept in the ODM database. Therefore, the port can be reconfigured into the available state. An option is given to also remove the port's information from the ODM database. If this option is chosen, the port is completely removed from the system.

Configure a Defined Port
If a port is in the defined state, then this selection makes it available.

Add COMIO Interface to a Port
Use of the emulation of the base AIX Version 3 support is optional. It would be needed if applications written to the base AIX Version 3 APIs such as **xtalk** or **SNA** are used. If this is required on a given port, then COMIO should be added to produce a new device entry such as `/dev/x25s0`. The emulator can be added to any available port. No additional configuration information is required.

Remove COMIO Interface to a Port
The emulation of the base AIX Version 3 support for a given port can be removed.

Add TCP/IP Interface to a Port
TCP/IP can be enabled on any of the available ports. Details of the port's TCP/IP information is then provided in a separate screen.

Remove TCP/IP Interface to a Port
The TCP/IP support for a given port can be removed.

Adding a Port

To add a port, select **Add Port** and a list of the available adapters will be presented. When you select an adapter, a SMIT screen will allow entry of the port's NUA. Information on the attribute choices available is given in the help text. It is probable that the port will require further configuration before use to match its characteristics with those provided by the network provider. Use the Change/Show Characteristics of Port screen to alter the attributes.

The following attributes must be entered:

PORT number The physical port on the adapter's expansion cable that is to be used. With the Portmaster adapters, this can vary between 0 and 7 for the V.24/RS-232 interface and between 0 and 5 for its other interfaces. For the X.25 Co-Processor adapters, this must be 0. If the port is associated with an IBM 2-Port Multiprotocol adapter, the port number is automatically filled in, and does not require an entry.

Local network user address The Network User Address for the port. This is usually supplied by the network provider and must agree with the subscription obtained.

The following attributes are optional. See "General Parameters" for more details.

Network Identifier On networks that implement X.121 addressing, the country prefix uniquely identifies the country. In some cases, this can be used to identify default characteristics of the network. In some countries, there is more than one network type, and though they share the same country code, they have differing characteristics. If the identification for the local network is listed, then that should be selected. Otherwise, select other public if attached to a standard network. Select other private if the network's addressing scheme is different from X.121.

Country prefix The country code provided by X.121. This should be made available by the network provider.

VC ranges For each of the four types of virtual circuit, the initial channel number, and the number of channels must be given. This must match the configuration of the local DCE, and so should be based on the information provided by the network provider.

Moving a Port

The attributes defining a port can be transferred to a different port. When **Move Port Definition** is selected:

1. Choose existing port to be moved.
2. Select the new adapter to be used. The new adapter can be the same as the port's current adapter.
3. Enter the physical port number on the given adapter.

Configuring a Port

From the ″Manage X.25 Ports″ screen , select **Change/Show Characteristics of Port**. The attributes for the port are divided into a number of categories. They are listed in the SMIT screen in the order in which they will most likely be used. It is often not necessary to configure anything other than the port's general parameters.

General Parameters:

General parameters are those most likely to need configuring. For a given port, there are the following attributes:

NUA The network user address for the port. This is usually supplied by the network provider and must agree with the subscription obtained.

Calling address
On a few networks, the NUA of the machine originating the call cannot be put into the call it generates. Usually the calling address is allowed. This attribute is used only with COMIO connections.

Enable DLPI
If direct access to the frame layer is required on this port, then DLPI should be enabled. This disables packet layer (X.25) access. DLPI is usually disabled.

VC ranges
VC ranges are constructed according to the lowest logical channel number and the logical channels desired for each of the VC types. The lowest logical channel number indicates the lowest-numbered logical channel that may be used for a specific VC type. This number depends on the network subscription and must be in the range of 1 through 4096. The lowest logical channel may be 0 only when the VC type is not used. Logical channel number 0 may not be used for a VC channel because it is always used as the network protocol channel. Check with your network provider to learn if logical channel 0 is included in a VC range.

The number of logical channels indicates how many VCs to configure starting at the lowest logical number of that VC type. This information is supplied by the network provider.

The specified VC channel ranges must not overlap and must not exceed the total number of channels allowed. In addition, the logical channel numbers must be defined according to the following rule:

```
PVC < Incoming SVC < Two-way SVC < Out-going SVC
```

If the range of VCs is not needed, set the lowest logical channel number and the number of logical channels for the VC type to 0.

X.32
This set of attributes should be set if the network provider requires X.32 DTE identification. Both the X.32 XID identity and the X.32 XID signature are supplied by the network provider. For more information, refer to Configuring a Port for X.32 .

Dial-Up
These attributes must be set if a dial-up connection is used. For more information, refer to Dial-Up Parameter Descriptions

Packet Parameters:

Packet parameters give the characteristics of the packet layer.

CCITT
The level of CCITT recommendation that is implemented by the network. The most significant changes are between 1980 and 1984 where packets sizes greater than 1024 were introduced, along with modulo 128. It should be selected to reflect the implementation of the network.

Modulo
The packet layer modulo must agree with that of the network. Usually, it is 8. This governs the number of packets that can be sent out before the sender must wait for acknowledgement of the data. On some networks, modulo 128 can be used. Though this would not normally be of benefit, the network provider would be able to advise on its usefulness in a given situation.

Type of line
By default, the system being configured will run as a DTE. In exceptional cases where two systems are being run back to back, one can be set up as a DCE. This does not allow the system to run as a full DCE; rather it just allows back-to-back operation.

DTE/DCE alteration
In cases where two systems are being run back to back and the remote system is also running with this licensed program, the selection of DTE/DCE can be made automatically. In cases where one system is a DTE connected to a network, this parameter need not be used.

Registration
Some networks support registration, where the DCE configures some of the DTE's characteristics automatically. This is not normally the case, and so this should usually be set to **no**.

A bit
On most X.25 networks, the method by which the addresses are coded into the call packet is standard. However, on some networks this differs. By far the more common is one called non-TDA/NZI the section addressing, and this is standard for most networks. The other mode of addressing is TOA/NPI. For regular usage, set the A-bit to **off**, which provides the regular non-TDA/NZI addressing. Set this to **on** only if it is required by the network provider.

NIDU
The data passed up from the packet layer to applications such as TCP/IP is broken up into data units. Typically, the size of the data unit should be such as to wholly contain the typical application data packet. For example, if the MTU for the X.25 TCP/IP interface is 1500, set the NIDU parameter to I500.

Packet sizes
These attributes only apply to switched virtual circuits used on the port. The default packet sizes are used when no negotiation of packet sizes takes place between the DTE and the DCE. Therefore the default sizes depend on the network subscribed to and should match the networks requirements. The default packet size does not have to match the remote systems default packet size unless the systems are running back to back. The maximum packet size is used if negotiation does take place between the DTE and DCE. The maximum size should be set to a size sufficient to contain the average size of the data unit used.

Window sizes
The default and maximum packet window size that can be used. These attributes specify how many packets can be sent before acknowledgement is received. The window sizes should be set according to the recommendations of the network provider. Generally, for slow, reliable communication links, a larger window size is recommended. For unreliable communication links, a small window size is desirable. When two systems are running back to back, their window sizes must match.

Throughput class
Throughput is the DTE-to-DTE transfer rate in bits per second (bps). The throughput can be affected by the local system, remote system, and the network. The default throughput class should be set to the line rate.

T (timers)
The packet layer timers, T20 through T28, implement timeouts for different types of packets.

R (counters)
If the particular timer packet is not confirmed within the given period, it is re-sent, based on the value of the equivalent R counter. If still no acknowledgement is received, a recovery process is started. The nature of the process started depends on which type of packet was left unacknowledged.

Fast Select
An optional facility that is available from some network providers. In a fast-select call, the call request is always rejected, but the CUD in the call and clear constitute the information exchange. This saves having to have the call established if only a small amount of data needs to be sent.

D bit
If the applications being run require DTE-to-DTE data acknowledgement, they will make use of the D bit. On some networks, this is a separate option, and so may not be subscribed to. If this is the case, then D bits should not be allowed, and this parameter should be set to **forbid**. Usually, they should be allowed.

Reverse Charging
An optional facility that allows a call to specify that the receiving party pays for the call. In all cases, **allow** allows it, while **allow if not billed** allows it only in outgoing calls.

Facility size
This gives the maximum number of bytes of facility that can be in a call.

Interrupt size
This gives the maximum number of bytes of data that can be held in an interrupt packet.

Frame Parameters:

The frame parameters give the characteristics of the frame layer.

T (timers)	The T1 timer gives the timeout by which an acknowledgement of the data sent should have been received. On slow lines, especially with large data packets, the value of T1 should not be too low. If too low, the frame will not be fully transmitted before the timer expires, and so a high number of T1 expirations would be seen. N2 specifies, if T1 does timeout, how many times the frame should be retransmitted. T2 is less likely to be varied, specifying to the software the required turn around time for a frame. T3 and T4 ensure that any frame layer failure is recognized.
Frame modulo	Specifies whether the counters that check frame-layer activity between the DTE and the DCE work in modulo 8 or 128. Usually modulo8 is used. This value will be based on the network.
Frame window	This gives the number of frames that can be sent to the DCE before the DTE must wait for acknowledgement. The value, typically, does not affect performance greatly and should be based on the network providers' recommendation.
DTE/DCE	Selection on whether the system operates as DTE or DCE can be made at the packet or frame layers. This selection can be automatic (based on the ISO standard ISO8208) or fixed. However, this does not allow the system to act as a full DCE.
Connection mode	Once the physical layer is connected, the frame layer is established. The different networks require different modes of startup at the frame layer. One of the most common is for the DTE to start sending out SABM frames to indicate its frame layer is ready. The network provider will provide additional information in this area.

Permanent Virtual Circuit (PVC) Parameters:

The characteristics of a PVC cannot be changed at the time of usage. Unlike SVCs, they have no call request/confirm exchange in which to negotiate packet size. The X.25 Default PVC Parameters SMIT screen allows the defaults that are to be assumed for this port to be set. If a specific PVC does not match these defaults, then this should be given as a non-default PVC. The Manage Non-Default PVC screen is provided for this. As identification of a given PVC is based on its logical channel number, this must be specified when dealing with non-default PVCs.

Packet sizes	The packet sizes to be used.
Window sizes	The packet window sizes to be used.
D bit	If the applications being run require DTE-to-DTE data acknowledgement, they make use of the D bit. On some networks, this is a separate option, and so may not be subscribed to. If this is the case, then D bits should not be allowed. Usually, D bits should be allowed.

Configuring a Port for V.25bis Dialing

The CCITT V.25bis Recommendation provides a protocol to connect automatic calling and/or answering equipment to a telephone network. This recommendation contains two types of connections. The connection type is distinguished by the DTR signal used. If the DTR signal is 108.1, the connection type is called **direct**. A **direct** V.25bis connection uses only V.24 signals to establish connections. If the DTR signal is 108.2, the connection type is called **addressed**. An **addressed** connection uses V.25bis commands and V.24 signals to establish connections. To use the **addressed** mode, your modem or ACU must support the V.25bis command set.

The V.25bis functionality provided in the Streams X.25 LPP configures the port for both incoming and outgoing calls. Therefore, your port can both originate and answer calls without having to be reconfigured.

Use the following method to configure a port for V.25bis dialing.

Using SMIT, enter:

```
smit
```

Select the following SMIT menus:

1. Select **Manage X.25 Ports.**
2. Select **Change/Show Characteristics of Port.**
3. Select **Change/Show X.25 General Parameters.**
4. On the **Change/Show X.25 General Parameters** screen, there is a Dial-Up Configuration menu that displays certain options for setting up calls.

The setup in the SMIT menus must match the setup of the actual modem or autocall unit (ACU) you are using. Therefore, it is important that you are familiar with the modem setup before you attempt to configure the port for V.25bis.

Here is a sample setup for using V25bis addressed mode. A phone number must be specified if the port is to originate any calls. When originating a call, the X.25 adapter sends the dial string included in the Phone Number or Address to Call field to the modem or ACU using V25bis commands.

```
Dial-Up Configuration
**********************
Connection type                          [V25bis]        +
V25bis Call Establishment Method         [Addressed]     +
Phone Number or Address to Call          [5551234]
Maximum Connection Delay                 [30]            #
Enable/Disable DSR Polling               [enable]        +
DSR polling timeout                      [30]
```

Here is a sample setup for using V25bis direct mode:

```
Dial-Up Configuration
**********************
Connection type                          [V25bis]         +
V25bis Call Establishment Method         [Direct]         +
Phone Number or Address to Call          []
Maximum Connection Delay                 [30]             #
Enable/Disable DSR Polling               [enable]         +
DSR polling timeout                      [30]
```

Dial-Up Parameter Descriptions

Connection Type There are three types of connections:

Direct/Leased
Default connection type.

V25bis Represents all dial-up connection types with the exception of manual dial.

Manual-dial
Connections where the user does the dialing instead of the modem.

V25bis Call Establishment Methods

There are two V25bis call establishment methods:

ADDRESSED

Uses V25bis commands to communicate with the modem or ACU. For originated calls, a dial string must be specified. This string is sent to the modem when the DTE requests the call. When answering calls, the modem uses V25bis commands to notify the DTE of an incoming call.

DIRECT

Uses V.24 signal control to establish connections. When originating a call, the modem must already know the dial string to call. When answering calls, the modem or ACU uses the Calling Indicator (circuit 125) to indicate an incoming call to the DTE.

> **Note:** The IBM ARTIC960Hx 4-Port Selectable PCI Adapter does not support the Calling Indicator (circuit 125). Therefore, direct mode can not be used to receive incoming V25bis calls with this adapter.

Phone Number or Address to Call

Specifies the phone number the modem should call to establish a connection. The phone number may contain the digits 0 through 9 inclusive and any of the following dial modifiers:

:	Wait tone
<	Pause
=	Separator 3
>	Separator 4
P	Dialing to be continued in pulse mode
T	Dialing to be continued in DTMF mode
&	Flash

These dial modifiers are only valid if using V25bis addressed mode.

Since the port may not be used to originate calls, the phone number is not a required field. However, if you do not specify a phone number and you try to send a call using addressed mode, the call will fail.

Maximum Connection Delay

Maximum time in seconds to allow for connection establishment. Must be set long enough for modems to perform handshaking or training operations. When originating calls, this value should be smaller than the T3 idle timer frame-layer parameter.

Used for all dial-up connections. For leased connections, this value remains unchanged.

Enable/Disable DSR Polling

DSR polling must be enabled if using a V25bis or manual-dial dial-up connection. If using a leased connection, DSR polling must be disabled.

DSR Polling Timeout

After call establishment, time in seconds to wait after DSR has dropped before ending the connection. Used only if DSR polling is enabled.

Configuring a Port for X.32

Use the following method to configure a port to use X.32 XID authentication.

Using SMIT, enter:

```
smit
```

Select the following SMIT menus:

1. Select **Change/Show Characteristics of Port.**
2. Select **Change/Show X.25 General Parameters.**
3. On the Change/Show General Parameters screen, the settings for the X.32 attributes must be obtained from your network provider.

X.32 Parameter Descriptions

Use X.32 XID Exchange	Indicates whether or not to use X.32 XID authentication procedures when the link is established.
X.32 XID Identity	Represents the X.32 identity of your port. This attribute must be provided by the DCE you are calling. The identity should be a string of 64 hexadecimal characters.
X.32 Signature	Identifies your password. This attribute must be provided by the DCE you are calling. The signature should be a string of 64 hexadecimal characters.

Managing COMIO Emulation

From the ″Manage X.25 Ports″ SMIT screen, emulation of the AIX Version 3 base X.25 driver support can be added to an X.25 port. This screen also allows it to be removed. For details on this screen, see “Managing Ports” on page 31.

When **Add COMIO Interface to Port** is selected, a choice of available ports is presented. Only one emulator can be added to any one port, and if it has already been added to the port chosen, the operation will fail.

When **Remove COMIO Interface from Port** is selected, a choice of the emulators configured on the system is presented.

Managing TCP/IP Configuration

TCP/IP support can be added or removed from a given port. For more details of the TCP/IP support, refer to the section on TCP/IP in the *AIX 5L Version 5.2 System Management Guide: Communications and Networks*.

Add a TCP/IP Interface

To add TCP/IP support, select **Add TCP/IP Interface to Port** from the Manage X.25 Ports and select the correct adapter.

Internet Address	The IP address to be given to this interface, for example, 192.35.231.224. There should be a host name for this address in **/etc/hosts**.
Network Mask	The netmask of the network depends on the network’s implementation. If the IP connection is simply from this port to another, then the default, which is obtained by leaving the field blank, is usually acceptable. If many X.25/IP connections are being used to make up a network, the netmask should be consistent across the systems.
Licensed Program Product Port	The X.25 port selected.
Activate	Choose whether to immediately activate the interface or not. If there is still configuration work to be done on the port, then it should be activated later with the **ifconfig** command.

Add Routes for Remote Systems

In order to communicate with a remote network, routes must be added.

The static route panel in smitty has changed. To add a route, change the following parameters:

1. Set **Destination TYPE** to ″host″
2. Set **DESTINATION Address** to the remote X.25 hostname
3. Set **Default GATEWAY Address** to the local X.25 hostname
4. Set **Network Interface** to ″xs0″

5. Set **Is this a Local (Interface) Route?** to ″yes″

To access the configuration menu using the SMIT fast path, enter:

```
smitty route
```

The following shows the SMIT menu with the entry fields containing sample field values:

```
                              Add Static Route

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  Destination TYPE                                     host
  DESTINATION Address                                 [remotex25hostname]
  (dotted decimal or symbolic name)
  Default GATEWAY Address                             [localx25hostname]
  (dotted decimal or symbolic name)
  COST                                                [0]
  Network MASK (hexadecimal or dotted decimal)        []
  Network Interface                                   [xs0]
  (interface to associate route with)
  Enable Active Dead Gateway Detection?                no
  Is this a Local (Interface) Route?                   yes
```

Remove a TCP/IP Interface

To remove a TCP/IP interface from a port, select **Remove TCP/IP Interface from Port** and then choose the interface to be deleted.

Map IP Addresses to X.25 NUAs

AIXlink/X.25 conforms to RFC 1236 ″IP to X.121 Address Mapping for DDN″ which defines an algorithm that automatically determines the NUA based on the Internet address.

Attachment to the Defense Data Network (DDN): Choosing the DDN network identifier invokes this algorithm in place of the x25ip tables. The RFC defines this support for Class A Internet addresses only (Class B and C addresses are also supported).

For All Other Network Identifiers: For all other types of network, the x25ip table maps TCP/IP Internet addressing to the X.121 Network User Address (NUA). This table is maintained via the SMIT IP/X.25 Host Entry menu.

Create an entry in this table for each remote host using either of the following methods:

- Using the SMIT fast path, enter:

  ```
  smit mkx25s
  ```

- Using SMIT, enter:

  ```
  smit
  ```

  Choose the following SMIT menus:

  1. Select **Communications, Applications, and Services.**
  2. Select **TCP/IP.**
  3. Select **Further Configuration.**
  4. Select **Network Interfaces.**
  5. Select **Network Interface Selection.**
  6. Select **X.25 Licensed Program Product IP Host Configuration.**
  7. Select **Add an X.25 Licensed Program Product IP Host Entry.**
  8. Select **Add a Switched Virtual Circuit (SVC) X.25 Licensed Program Product IP Host Entry.**

Both methods display the Add an X.25 Licensed Program Product IP Host Entry screen. This screen contains fields in which you type or select the desired values.

Notes:

1. A virtual call is made when the first session with a remote system is established.
2. Several IP sessions can share the same virtual circuit. Once an X.25 communication has been established, all IP traffic between two systems shares the same virtual circuit. Only one SVC or PVC is needed for communications between two systems.
3. The X.25 virtual circuit is left connected for a period of time after the last IP session has been closed. This is so the cost of call establishment is not incurred if the remote system is contacted again in the near future (which is reasonable). This time delay, which is approximately 20 minutes, is modifiable with the Network Options (no) command using the `arpt_killc` option. The **no** command is available to modify some network parameters for the pending IPL. If you want your new values to remain unchanged after each system boot, add the command at the end of the **/etc/rc.tcpip** file. For example, to clear the virtual circuit after five minutes of inactivity:

   ```
   no -o arpt_killc=5
   ```

   Note: This applies to SVCs and PVCs that are removed using the **arp -d** command.
4. To clear a TCP/IP X.25 virtual connection, use the **arp -d** command. For example to clear an existing virtual connection between `kilix` and `filix`, type on filix:

   ```
   arp -d filix
   ```

Managing the Triple-X PAD

The functionality of the PAD can be divided into two types of PAD, host and terminal PADs. The host PAD allows remote ASCII terminals to access applications running on it, with their access across an X.25 network. A terminal PAD allows the ASCII terminals to directly attach to the system over asynchronous links, and for them to use the X.28 protocol to establish calls with the remote X.25 attached system.

To enable host PAD support, the Manage Triple-X PAD option should be selected from the Manage X.25 LPP Device Driver menu. This option brings up a menu from which the host PAD support can be added or stopped - the Start/Stop the Triple-X PAD X.29 Daemon menu. Select the Add and Configure/Stop the X.29 Daemon menu to start the X.29 daemon. The daemon is only started for that instance and will be stopped automatically if the machine is rebooted. To avoid having to restart the daemon each time, add the following to any start-up script:

```
/usr/lib/drivers/pse/x29d
```

The terminal support does not need to be configured and is available once the X.25 and PAD software is installed, and the X.25 ports configured.

Configuration Commands

The X.25 Licensed Program can be configured directly from the command line if this is desired, though for most users the SMIT interface is preferable. A set of general system and specific X.25 commands allow the X.25 system to be configured.

Managing Device Driver

Instances of the adapter device driver `twd` are added and deleted through the system commands **mkdev** and **rmdev**. The port and adapter numbers used in the following examples would have to be replaced by those needed on a particular system.

Add a driver

```
mkdev -c driver -s artic -t star -w0 -p 'apm1'
```

This assumes a Portmaster adapter, but could equally be a coprocessor `ampx`. For more details, see the **mkdev** command.

Remove a driver

```
rmdev -l twd1
```

Moves the driver `twd1` to the defined state. Using the **-d** option also removes the configured data from the database. For more details, see the **rmdev** command.

List drivers

```
lsdev -C -t star or lsx25
```

For more details, see the **lsdev** or **lsx25** commands.

Managing X.25 Ports

To manage the X.25 ports, the X.25 commands listed should be used:

Add a port

The appropriate command to use depends on which base device driver (**twd** or **hdlc**) is used by the port.

Note: The following is only an example.

If the port uses the **hdlc** device driver:

```
mksx25 -c port -s star -t stx25 -a nddname='hdlc2' -a local_nua='54663' -a network_id='5'
```

If the port uses the **twd** device driver:

```
mksx25 -c port -s star -t stx25 -p 'twd1' -w '4' -a local_nua='54663' -a network_id='5'
```

Change a port

```
chsx25 -l sx25a0 -a...
```

Changes the attributes listed for the port given. This allows such things as VC ranges and packet sizes to be modified. For more details, see the **chsx25** command.

Remove a port

```
rmsx25 -l sx25a0
```

Moves the port `sx25a0` to the defined state. Using the **-d** option also removes the configured data from the database. For more details, see the **rmsx25** command.

List ports

```
lsdev -C -t stx25
```

Add a non-default PVC

```
mkpvc -U -l sx25a0 -n 4 -s 256
```

Adds a non-default PVC on port `sx25a0`'s logical channel number 4, itsnon-default characteristic being a transmit packet size of 256. For more details, see the **mkpvc** command.

List PVCs

```
lspvc -l sx25a0
```

For more details, see the **lspvc** command.

Chapter 5. Network Provider Interface Programming Reference

This chapter discusses how the network provider interface (NPI) is used with X.25. It includes the primitives necessary to run NPI on an X.25 network and illustrates how programs can use NPI.

The network provider interface (NPI) provides a connection-oriented programming interface based on the *UNIX International Network Provider Interface Specification, Version 2.0.0*. A connection-oriented subset of the standard has been implemented with enhancements to better suit it to an X.25 environment.

NPI consists of a set of primitives defined as STREAMS messages that provide access to the network layer services. The primitives are transferred between the network service (NS) user entity and the NS provider. An NPI primitive can be one of the following types:

User-originated These primitives make requests to the NS provider or respond to an event of the NS provider.

Provider-originated These primitives are either confirmations of a request or are indications to the NS user that the event has occurred.

The following diagram illustrates the NPI:

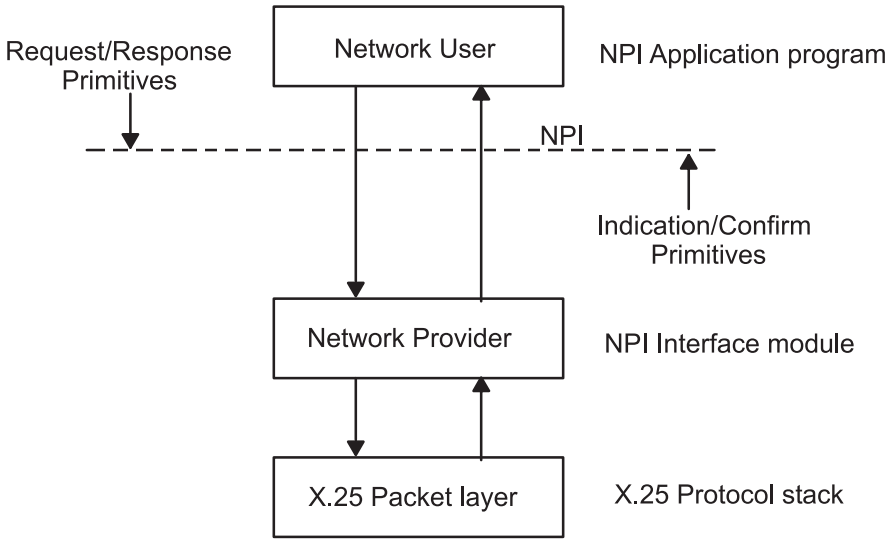


Figure 8. Model of the NPI

Multiple NPI applications can access the X.25 protocol code.

The main features of the connection-mode communication are:

- Virtual circuits.
- Data transfer by a pre-established path.
- Data transfer reliability.

There are three phases to each instance of communication: connection establishment, data transfer, and connection termination. Units of data arrive at their destination in the same order as they departed their source, and the data is protected against duplication or loss. Along with the NPI primitives required to use the X.25 communications, there are a number of local management primitives that act locally to the NPI module.

For an application to use NPI, it would typically follow these steps:

1. Open a stream to the packet driver.
2. Push the NPI module.
3. Bind the application to the NPI module.
4. Submit a connection request to establish a connection between the local and remote X.25 systems.
5. Once the connection becomes established, transfer the required data.
6. Disconnect the two systems.

NPI Enhancements for AIXlink/X.25 Version 2.0.0

AIXlink/X.25 Version 2.0.0 enhances Network Provider Interface (NPI) support by supplying CCITT cause and diagnostic codes for X.25 to the user's NPI applications.

Existing applications compiled with AIXlink/X.25 Version 1.1 can still continue to run on AIXlink/X.25 Version 2.0.0 without any modification. Customers who wish to receive the cause and diagnostic codes must recompile their X.25 applications to take advantage of the modified functions.

The following NPI structures contained in **/usr/include/sys/npi_20.h** have been changed to allow users to use the CCITT cause and diagnostic codes:

N_reset_req_t
N_reset_ind_t
N_discon_req_t
N_discon_ind_t

Structure Changes for 64–bit Mode

All structures using the **ulong** definition have been changed to use **ulong32int64_t** instead of **ulong**. This is for use with the 64–bit kernel. However, AIXlink applications do not need to change definitions from **ulong** to **ulong32int64_t** since they run in 32-bit mode. In 32-bit mode **ulong32int64_t** and **ulong** definitions take up 4 bytes of space.

Sample Program

The use of NPI is demonstrated in a set of sample programs, which is located in the following directory:

/usr/samples/sx25/npi

Local Management Primitives

Local management primitives are called to open, close, manage, and report information on a stream connected to the NPI module. They also manage options supported by it and report information on its supported parameter values.

The following local management primitives are supported:

N_INFO_REQ Indicates a network information request.
N_INFO_ACK Indicates a network information acknowledgment.
N_BIND_REQ Indicates a bind protocol address request.
N_BIND_ACK Indicates a bind protocol address acknowledgment.
N_UNBIND_REQ Indicates an unbind protocol address request.
N_ERROR_ACK Provides error acknowledgment.

N_OK_ACK Provides successful receipt acknowledgment.

Connection-Mode Primitive Formats and Rules

A network connection (NC), once established, is like a pair of queues linking two network addresses, or X.25 hosts. There is one queue for each direction of information flow, and each queue provides a flow control function for its information flow. The pair of queues is considered to be available for each potential NC. As with the X.25 protocol, NPI is a state machine, what action the application should take at a given time is sometimes due to the state that the protocol layer is in. For example, a call cannot be established until the application binds to the NPI module.

The following rules and guidelines apply to connection-mode service:

- Primitives can be put onto a queue by the application, subject to the control of the NPI module.
- The NPI module can add data to the queue.
- The receiving application controls the removal of objects from the queue. The objects are normally removed in the order they were received, except when:
  - The object is of a type defined to be able to advance ahead of the preceding object.

    Note: No object can advance ahead of another object of the same type.
  - The following object is defined to be destructive with respect to the preceding object on the queue. If necessary, the last object on the queue would be deleted to allow a destructive object to be put on the queue. For example, disconnect objects are defined to be destructive with respect to all other objects. Reset objects are defined to be destructive with respect to all other objects except connect, disconnect, and other reset objects.

The CONS primitives can be grouped as follows:

- Connection establishment primitives
- Normal data transfer primitives
- Receipt confirmation service primitives
- Reset service primitives
- Network connection release primitives

The following sections describe the format and rules of CONS primitives.

Connection Establishment Primitives

The following network service primitives pertain to establishing a connection. To make a connection, the application must previously have bound to the NPI module.

N_CONN_REQ Indicates a network connection request.
N_CONN_IND Provides a network connection indication.
N_CONN_RES Indicates a network connection response.
N_CONN_CON Provides a network connection confirmation.

The connection sequence is as follows:

1. Network service (NS) user, user A, sends a connection request primitive (N_CONN_REQ) to NS user, user B.
2. User B receives the corresponding connection indication primitive (N_CONN_IND).
3. User B accepts the connection by sending a connection accept primitive (N_CONN_RES) to User A.
4. User A receives the corresponding connection confirm primitive (N_CONN_CON).

Now, the connection is established.

A pair of queues is associated with the connection between the two systems once the connection sequence completes. The queues can now be added to the NS user's read queue. The queues remain associated with the connection until the NS user sends a disconnect request primitive (N_DISCON_REQ) or receives a disconnect indication primitive (N_DISCON_IND).

Normal Data Transfer Primitives

The following NPI primitives provide normal data transfer service:

N_DATA_REQ Indicates a normal data transfer request.
N_DATA_IND Provides a normal data transfer indication.

These primitives provide a way to exchange data between the two systems. NPI refers to the units of data passed as network service data units (NSDUs). The exchange can occur in one direction or in both directions simultaneously. The X.25 layer preserves the sequence and the boundaries of these data units. If an **N_DATA_REQ** for an NSDU greater than the NIDU packet parameter is sent to NPI, the receiving application gets several **N_DATA_IND** primitives. It is the responsibility of the receiving application to check the **N_MORE_DATA_FLAG** in each data indication until the entire NSDU is received.

Receipt Confirmation Service Primitives

The following NPI primitives provide receipt confirmation service:

N_DATACK_REQ Indicates a data acknowledgment request.
N_DATACK_IND Provides a data acknowledgment indication.

The **N_RC_FLAG** flag of the **N_DATA_REQ** primitive enables the receipt confirmation service. For each data unit received with the confirmation request parameter set, the receiving user should return an **N_DATACK_REQ** primitive. These acknowledgments are issued in the same order as the corresponding **N_DATA_IND** primitives are received. The NS provider responds to the primitives such that each acknowledgment is distinct from previous or subsequent acknowledgments. The application would have to correlate the acknowledgments with the original requests. When an NSDU has been segmented into more than one network interface data unit (NIDU), only the last NIDU can request receipt confirmation.

N_DATACK_REQ primitives are not subject to the flow control that affects **N_DATA_REQ** primitives at the same NC endpoint. **N_DATACK_IND** primitives are not subject to the flow control affecting **N_DATA_IND** primitives at the same NC endpoint.

To use receipt-confirmation, the two applications must agree upon its use at connection time, assuming it is supported by the underlying X.25 network. The **REC_CONF_OPT** flag of the **N_CONN** primitives requests receipt-confirmation service. As it is based on the X.25 D bit, the X.25 subscription being used must have D bit support.

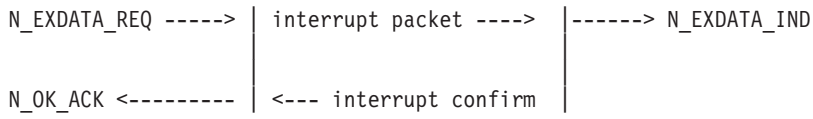
Expedited Data Transfer Service

The following NPI primitives provide an expedited data transfer service:

N_EXDATA_REQ Indicates an expedited data transfer request.
N_EXDATA_IND Provides an expedited data transfer indication.

The expedited data transfer primitives correspond to X.25 interrupt packets. When the network services (NS) provider receives an **N_EXDATA_REQ** from the user, the provider generates an X.25 interrupt

packet. When the NS provider receives an X.25 interrupt packet from the network, the provider automatically sends an X.25 interrupt confirm packet to the network and sends an **N_EXDATA_IND** primitive to the user. When the NS provider on the side that sent the **N_EXDATA_REQ** receives an X.25 interrupt confirm packet from the network, the provider sends an **N_OK_ACK** primitive to the user to acknowledge that the interrupt sequence completed successfully. The NS provider sends the **N_OK_ACK** primitive to the user, so that the application knows that the expedited data transfer sequence completed successfully. The expedited data transfer sequence must complete before another expedited data request can be sent from the same user. The expedited data transfer sequence is illustrated in the following example:



Note: The expedited data transfer service is always allowed with NPI, during the data transfer state. Therefore, even though an NS user may choose not to ever send an **N_EXDATA_REQ** primitive, the user must be able to handle receiving **N_EXDATA_IND** primitives from the NS provider, during the data transfer mode.

Reset Service Primitives

The following NPI primitives provide a reset service:

N_RESET_REQ	Indicates a reset request.
N_RESET_IND	Provides a reset indication.
N_RESET_RES	Indicates a reset response.
N_RESET_CON	Provides a reset confirmation.

An application would use the reset service to resynchronize the connection. The underlying X.25 layer uses the service to report a detected loss of unrecoverable data within the network service. The reset causes the NS provider to discard all data, expedited data, and receipt confirmations associated with this connection.

Network Connection Release Primitives

The following NPI primitives provide for network connection release:

N_DISCON_REQ	Indicates a disconnect request.
N_DISCON_IND	Provides a disconnect indication.

The connection can be released in the following ways:

- Either or both of the NS users can issue a **N_DISCON_REQ** to release an established connection.
- The destination application can reject an **N_CONN_IND** primitive.
- The NS provider, or underlying network, can release the connection. (All failures by the network to maintain a connection are indicated in this manner.)
- The NS provider can fail to establish a requested NC.

A connection can be released at any time, regardless of its current state. A disconnection request cannot be rejected, so once invoked, the connection will be released. Once the release phase is entered, the network service does not guarantee delivery of any data.

NPI STREAMS Programming

The NPI interface works in the STREAMS environment. The following provides an overview of how to program to the STREAMS interface.

Initially the ″NPI″ module is not available for use by an application, and the application itself must push it onto a stream.

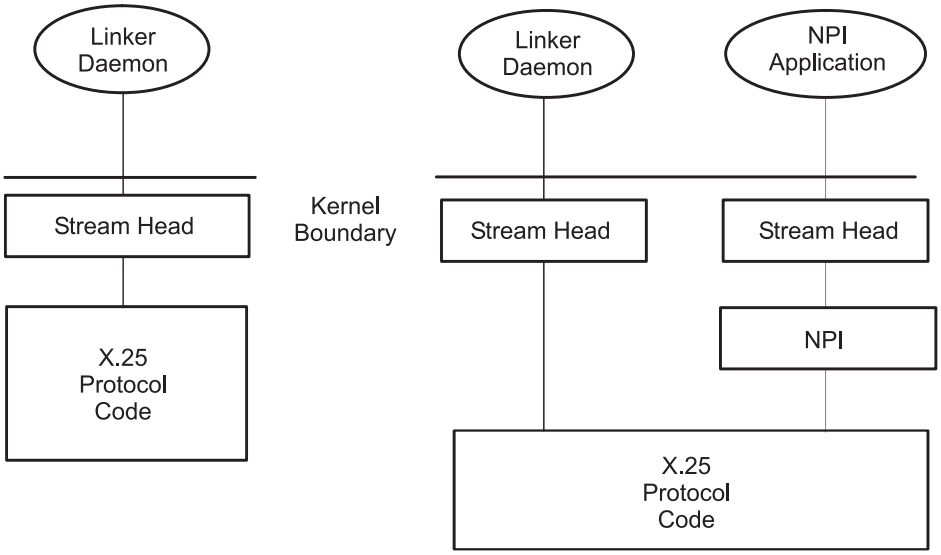


Figure 9. Initial X-25 Stream State with NPI

For an application to work with NPI in the STREAMS environment, the NPI module must be available to STREAMS applications. Once AIXlink/X.25 Version 2.0 is installed on a system, the NPI module is made available. Then applications that link to the STREAMS library are able to access the NPI module. The way that an application ″pushes″ NPI is shown in the following sample of pseudo code:

```
dev=open("/dev/x25pkt", O_RDWR);
/* assumes good return value */
ioctl(dev, I_PUSH, "npi");          /* uses the streams library
                                       push ioctl */
```

Once pushed, the application may then bind to NPI. See the **bind** primitive for more details. The application uses the functions **putmsg** and **getmsg** to communicate with NPI. Detailed under the STREAMS documentation, examples of their use with NPI are given under some of the primitives. An example of programming to NPI can be seen in the licensed program's sample programs.

Handling Calls

The N_CONN primitives are used to generate outgoing calls and accept incoming calls. If incoming calls are expected, the application must specify their characteristics to identify them. The X.25 call user data field is used to identify incoming calls to NPI, and how the listen is specified is described under the N_BIND_REQ primitive. The N_DISCON primitives are used to terminate the calls.

N_BIND_REQ Primitive

Purpose

Requests that NPI bind a user stream to the X.25 protocol stack. For connections using SVCs, the bind request gives details about how the user stream is to be used. If the user stream is to be used to receive

incoming calls, then the bind request gives details on what incoming calls should be routed to that stream. For connections using PVCs, the bind request gives details of what logical channel number to use.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong ADDR_length;
   ulong ADDR_offset;
   ulong CONIND_number;
   ulong BIND_flags;
   ulong PROTOID_length;
   ulong PROTOID_offset;
} N_bind_req_t;
```

Description

The **N_BIND_REQ** primitive requests that a network address be associated with a stream. It gives the maximum number of connect indications that can be outstanding for the particular application through the use of the CONIND_number field. It also indicates if the stream is a listening stream or a non-listening stream. A listening stream only receives connect indications (**N_CONN_IND**), and a non-listening stream either sends connect responses (**N_CONN_RES**) or connect requests (**N_CONN_REQ**). Bind requests for listening streams should use the **PROTOID** field to specify the details of incoming calls that should be listened for. In addition, bind requests for listening streams must use the CONIND_number field to specify the number of connection indications that can be outstanding at any given time.

If it is desired that a stream bound as a non-listening stream issue both **N_CONN_REQ** and **N_CONN_RES** primitives, it is necessary to do two binds. One bind allows **N_CONN_REQ** primitives to be sent; then an **N_UNBIND_REQ** primitive followed by an **N_BIND_REQ** primitive that binds the stream for sending the **N_CONN_RES** primitive is issued.

A sample of NPI code is included with the X.25 product.

Parameters

PRIM_type Indicates the **N_BIND_REQ** primitive.

ADDR_length Indicates the length, in bytes, of the network address to be bound to the stream.

ADDR_offset Specifies where the network address begins. The parameter value is an offset from the beginning of the **M_PROTO** block.

The following table defines specific address formats to be used at bind time. Note that this primitive is also used for attaching PVCs.

N_BIND_REQ *SVC Address Formats*

Byte #	Represents	Value	Format
1	line number	0-255	Binary
2	address prefix	0 for X.121	ASCII
3 - on	address itself	X.121 address	ASCII

N_BIND_REQ *PVC Address Formats*

Byte #	Represents	Value	Format
1	line number	0-255	Binary
2	address prefix	P	ASCII
3 - on	logical channel #	0-4095	ASCII

Note: At a given time, not all 4096 LCNs are available. The range must be that which is configured for this X.25 port.

The line number can be obtained several ways. The **lsx25** command lists the logical port number for each port, which is the line number. Another way is to use the **lsattr** command as follows:

```
lsattr -E -l portname -a port_num
```

The value to use for the line number is the *port_num* field. A third option is to use the **odm_get_obj** subroutine. An example of this code can be found in the sample code directory for NPI.

Note: The *port_num* value is set by the X.25 licensed program when the port is defined. The X.25 software uses this field to understand which physical port the application is referencing. Therefore, the value of *port_num* may not be changed by anyone other than the X.25 licensed program.

CONIND_number — Indicates the requested number of outstanding connect indications for the specified protocol address. An outstanding connection indication is one that has not been accepted by the application through the use of a connection response primitive (**N_CONN_RES**). This number should be set to zero if a non-listening stream is desired or greater than zero if a listening stream is desired. A listening stream can also be a **DEFAULT_LISTENER**, which means that it will accept connect indications (**N_CONN_IND**) for any network address on any port with any call user data.

BIND_flags Specifies one or more of the following:

DEFAULT_LISTENER
Indicates that the current stream is the default listener. This means that the stream accepts connect indications (**N_CONN_IND**) for any network address on any port with any call user data. There can only be one default listener per system. If this flag is set, the CONIND_number should be set to a value greater than zero.

TOKEN_REQUEST
Indicates that a token be assigned to the stream. This flag should only be set if the stream is to be a non-listener stream used to send a connection response primitive (N_CONN_RES). When this flag is set in the bind request, the N_BIND_ACK primitive returns a token value to the user. The user uses the token in a subsequent N_CONN_RES primitive to identify the stream on which the connection is to be established.

Note: This flag should not be set if the non-listener stream is to be used to send a connection request primitive (N_CONN_REQ).

TRANSPAC_OPT
On most networks, an outgoing originated call contains both the calling and called addresses. On some networks, however, the calling address must be absent from the call packet since the network itself insets this address. When connected to such networks, this flag must be set. To set it, use a logical OR to ″OR″ in the value of the **TRANSPAC_OPT** flag with the **BIND_flags** field.

Note: In cases where the **TRANSPAC_OPT** flag is used, the following rules apply:

- If sub-addressing is desired, the address in the N_BIND_REQ should be formatted as follows:
  - BYTE1 = line number
  - BYTE2 = 0 (X.121 address prefix)
  - BYTE 3 and on = sub-address.
- Otherwise, the address in the N_BIND_REQ should be formatted as follows:
  - BYTE1 = line number
  - BYTE2 = 0 (X.121 address prefix).

PROTOID_length Specifies the length, in bytes, of the protocol IDs to be bound to the stream.

PROTOID_offset Specifies where the protocol ID begins. The parameter value is an offset from the beginning of the **M_PROTO** block.

The **PROTO_id** field gives a pattern to match to the call user data (CUD) of incoming X.25 packets. It must be a null-terminated ASCII string where only digits (0-9), hex digits (A-F), ″?″ and ″*″ are allowed. Note that the length of this string, including the null character, cannot exceed 34 bytes. Following are some examples of how the matching for the **PROTO_id** works.

1234 Only matches for the specific CUD 1234.

1234* Matches for any CUD which starts with the digits 1234. This includes the CUD 1234 itself since a * (asterisk) matches any number of characters including none.

12*34 Matches for any CUD, including 1234, which starts with 12 and ends with 34.

12?? Matches for any CUD consisting of exactly four digits and starting with the characters 12.

12??* Matches for any CUD consisting of at least four digits and starting with the characters 12.

???? Matches for any CUD consisting of any four digits.

????* Matches for any CUD consisting of at least four digits.

1 Matches for any CUD containing the digit ″1″.

***** Matches for any CUD. This includes calls that have no CUD.

Acknowledgments

The NS provider generates one of the following acknowledgments upon receipt of the **N_BIND_REQ** primitive:

Successful	The NS provider sends the **N_BIND_ACK** primitive.
Unsuccessful	Non-fatal errors are indicated by the **N_ERROR_ACK** primitive.

Error Codes

The applicable non-fatal errors are as follows:

- To a network address with the *CONIND_number* parameter set to a nonzero value.
- With the **DEFAULT_LISTENER** flag value set to nonzero.

NNOADDR	Indicates that no local address was supplied in the **N_BIND_REQ**.
NOUTSTATE	Indicates the primitive was issued from an invalid state.
NSYSERR	Indicates a system error. The error is indicated in the **N_ERROR_ACK** primitive.

Implementation Specifics

The **N_BIND_REQ** primitive is part of X.25 licensed program.

N_BIND_ACK Primitive

Purpose

Acknowledges that the NPI application has been bound to a network address.

Structure

This primitive consists of one **M_PCPROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong ADDR_length;
   ulong ADDR_offset;
   ulong CONIND_number;
   ulong TOKEN_value;
   ulong PROTOID_length;
   ulong PROTOID_offset;
} N_bind_ack_t;
```

Description

The **N_BIND_ACK** primitive indicates to the application that the specified network user entity has been bound to the requested network address. The primitive also indicates the number of outstanding connect indications that can be queued.

Parameters

PRIM_type Specifies the **N_BIND_ACK** primitive.

ADDR_length Specifies the length, in bytes, of the bound network address.

ADDR_offset Specifies where the network address begins. The value of this parameter is the offset from the beginning of the **M_PCPROTO** block.

CONIND_number Specifies the accepted number of outstanding NS provider connect indications allowed to be outstanding by the NS provider for the specified network address. The *CONIND_number* parameter will have one of the following values:

- **0** Indicates the stream cannot accept **N_CONN_IND** messages.
- **>0** Indicates the NS user can accept up to the specified number of **N_CONN_IND** messages before responding with an **N_CONN_RES** or **N_DISCON_REQ** message.

TOKEN_value Specifies the value of the token assigned to this stream. The value can be used by the NS user in an **N_CONN_RES** primitive to accept a NC on this stream. The value will be nonzero and will be unique to all streams bound to the NS provider.

PROTOID_length Specifies the length of the bound protocol IDs.

PROTOID_offset Specifies the offset of the bound protocol IDs.

Implementation Specifics

The **N_BIND_ACK** primitive is part of X.25 Licensed Program.

N_UNBIND_REQ Primitive

Purpose

Requests that the NPI application be unbound from the network address.

Syntax

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
}N_unbind_req_t;
```

Description

The **N_UNBIND_REQ** primitive requests that the NPI unbinds the application that was previously bound to the network address.

Parameters

PRIM_type Specifies the **N_UNBIND_REQ** primitive.

Acknowledgments

NPI can generate the following acknowledgments upon receipt of the primitive:

Successful	The **N_OK_ACK** primitive acknowledges that the **N_UNBIND_REQ** primitive successfully completed.
Unsuccessful	The **N_ERROR_ACK** primitive indicates non-fatal errors.

Error Codes

The applicable non-fatal errors are as follows:

NOUTSTATE	Indicates the primitive was issued from an invalid state.
NSYSERR	Indicates a system error. The error is indicated in the **N_ERROR_ACK** primitive.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

N_OK_ACK Primitive

Purpose

Indicates the network provider received the previous user-originated primitive.

Syntax

This primitive consists of one **M_PCPROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong CORRECT_prim;
} N_ok_ack_t;
```

Description

The **N_OK_ACK** primitive indicates to the application that the network provider received the previously submitted primitive. The **N_OK_ACK** does not indicate any network protocol action taken due to the issuance of the last primitive. The **N_OK_ACK** primitive can only be initiated as an acknowledgment for user-originated primitives that have no other means of confirmation. These primitives include the **N_UNBIND_REQ**, **N_RESET_RES**, **N_CONN_RES**, and **N_DISCON_REQ** primitives.

Parameters

PRIM_type	Specifies the **N_OK_ACK** primitive.
CORRECT_prim	Identifies the successfully received primitive.

Implementation Specifics

The **N_OK_ACK** primitive is part of X.25 Licensed Program.

N_ERROR_ACK Primitive

Purpose

Provides notification of an error.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong ERROR_prim;
   ulong NPI_error;
   ulong UNIX_error;
} N_error_ack_t;
```

Description

The **N_ERROR_ACK** primitive informs the application that a non-fatal error occurred in the previously issued primitive. This primitive can only be initiated as an acknowledgment for those primitives that require one. The **N_ERROR_ACK** primitive does not perform any action on the primitive that caused the error.

Parameters

PRIM_type	Specifies the **N_ERROR_ACK** primitive.
ERROR_prim	Identifies the primitive that caused the error.
NPI_error	Contains the network provider interface (NPI) error code.
UNIX_error	Contains the system error code. This parameter can only be nonzero if the value of the *NPI_error* parameter is **NSYSERR**.

Error Codes

The following error codes can be returned:

NBADADDR	Indicates the specified network address was in an incorrect format, or the address contained illegal information.
NBADOPT	Indicates the specified options values were in an incorrect format or contained illegal information.
NNOADDR	Indicates the NS provider could not allocate an address.
NOUTSTATE	Indicates the primitive was issued from an invalid state.
NBADSEQ	Indicates the specified sequence number was incorrect or illegal.
NBADDATA	Indicates the specified amount of user data was outside the range supported by the NS provider.
NSYSERR	Indicates a system error. The error is indicated in the primitive.
NNOTSUPPORT	Indicates the specified primitive type is not known to the NS provider.
NODDCUD	Indicates an odd-length call user data string.

Implementation Specifics

The **N_ERROR_ACK** primitive is part of X.25 Licensed Program.

N_INFO_REQ Primitive

Purpose

Requests network information from the network service (NS) provider.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_info_req_t;
```

Description

The **N_INFO_REQ** primitive requests the NS provider to return the values of all supported protocol parameters and the current state of the NS provider. The **N_INFO_REQ** primitive does not affect the state of the network. The information returned is detailed under the **N_INFO_ACK** primitive.

Parameters

PRIM_type Indicates the **N_INFO_REQ** primitive.

Acknowledgments

The NS provider generates one of the following acknowledgments upon receipt of the primitive:

Successful	The **N_INFO_ACK** primitive acknowledges the **N_INFO_REQ** primitive.
Unsuccessful	There are no non-fatal errors associated with issuing this primitive.

Implementation Specifics

The **N_INFO_REQ** primitive is part of X.25 Licensed Program.

N_INFO_ACK Primitive

Purpose

Acknowledges a request for network information.

Structure

This primitive consists of one **M_PROTO** message block with the following structure and values. Where a value is not supported, QOS_UNKNOWN is returned:

```
typedef struct {
ulong PRIM_type;        /* always N_INFO_ACK */
ulong NSDU_size;        /* max NSDU size */
ulong ENSDU_size;       /* max ENSDU size */
ulong CDATA_size;       /* connect data size */
ulong DDATA_size;       /* disconnect data size */
ulong ADDR_size;        /* address size */
ulong ADDR_length;      /* address length */
ulong ADDR_offset;      /* address offset */
ulong QOS_length;       /* length of the default QOS values */
ulong QOS_offset;       /* offset of the default QOS values from the
                           beginning of the block */
ulong QOS_range_length; /* length of the range of QOS values */
ulong QOS_range_offset; /* offset of the range of the QOS values from
                           the beginning of the block */
```

```
ulong OPTIONS_flags;    /* bit masking for options supported */
ulong NIDU_size;        /* network interface data unit size */
long SERV_type;        /* service type */
ulong CURRENT_state;    /* current state */
ulong PROVIDER_type;    /* type of provider */
ulong NODU_size;        /* optimal NSDU size */
ulong PROTOID_length;   /* length of bound protocol ids */
ulong PROTOID_length;   /* offset of bound protocol ids */
ulong NPI_version;      /* version number of npi that is supported */
} N_info_ack_t;
```

Description

The **N_INFO_ACK** primitive acknowledges a request for network information. The primitive indicates to the network services (NS) user any relevant protocol-dependent parameters. The **N_INFO_ACK** primitive serves as a response to the **N_INFO_REQ** primitive. The data transmission sizes are based on those of the underlying X.25 network.

Parameters

PRIM_type	Indicates the primitive type.
NSDU_size	Specifies the maximum size, in octets, of a network service data unit (NSDU) which is based on the X.25 packet size.
ENSDU_size	Specifies the maximum size, in octets, of an expedited network service data unit (ENSDU), which is based on the X.25 network's interrupt packet size.
CDATA_size	Specifies the maximum number of octets of data that can be associated with connection establishment primitives, which is based on the X.25 network's call user data limit.
DDATA_size	Specifies the maximum number of octets of data that can be associated with the disconnect primitives, which is based on the X.25 network's clear user data limit.
ADDR_size	Specifies the maximum size, in decimal digits, of a network address. The value must be between 1 and 40.
ADDR_length	not supported, returns **QOS_UNKNOWN**
ADDR_offset	not supported, returns **QOS_UNKNOWN**
QOS_length	not supported, returns **QOS_UNKNOWN**
QOS_offset	not supported, returns **QOS_UNKNOWN**
QOS_range_length	not supported, returns **QOS_UNKNOWN**
QOS_range_offset	not supported, returns **QOS_UNKNOWN**
OPTIONS_flags	not supported, returns **QOS_UNKNOWN**
NIDU_size	Indicates the amount of user data that can be present in an **N_DATA** primitive. The value of the *NIDU_size* parameter should not be larger than the value of the *NSDU_size* parameter.
SERV_type	Specifies the service type supported by the NS provider. The only possible value is N_CONS which indicates connection-mode service.
CURRENT_state	not supported, returns **QOS_UNKNOWN**
PROVIDER_type	not supported, returns **QOS_UNKNOWN**
NODU_size	not supported, returns **QOS_UNKNOWN**
PROTOID_length	Specifies the length of the protocol identifiers that were bound with the **N_BIND_REQ** primitive. This value should be -1, which indicates that there is no obtainable information that corresponds to the parameter's definitions.
PROTOID_offset	Specifies the offset of the protocol identifiers to be bound from the beginning of the **M_PCPROTO** message block. This value should be -1, which indicates that there is no obtainable information that corresponds to the parameter's definitions.
NPI_version	not supported, returns **QOS_UNKNOWN**

Implementation Specifics

This primitive is part of X.25 Licensed Program.

N_CONN_REQ Primitive

Purpose

Requests a network connection.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong DEST_length;
   ulong DEST_offset;
   ulong CONN_flags;
   ulong QOS_length
   ulong QOS_offset;
 } N_conn_req_t;
```

Note: *QOS_length* and *QOS_offset* are not supported and should be set to 0 (zero).

Description

The **N_CONN_REQ** primitive requests that the network service (NS) provider make a network connection to a specified destination. This will generate an X.25 call request to the remote X.25 host.

The format of the message is one **M_PROTO** message block followed by one or more **M_DATA** blocks for the NS user data transfer. Specifying user data is optional. The NS user can send any integral number of octets of data within the range supported by the NS provider. (For more information, see the **N_INFO_ACK** primitive.)

The following table defines the specific address format to be used for SVCs:

N_CONN_REQ_NPI SVC Address Format			
Byte # (from 1)	**Represents**	**Value**	**Format**
1	address prefix	0 for X.121	ASCII
2 - on	address itself	X.121 address	ASCII

Note: User data is expected to be an even-length, null-terminated ASCII string. Facilities should be included in the M_DATA, and are expected to be coded as they would appear in a call packet. The sample code gives an example of facilities and CUD.

For PVCs, the N_CONN primitives are not used. The connection is established through the N_BIND_REQ.

Parameters

PRIM_type Specifies the **N_CONN_REQ** primitive.

DEST_length Specifies the length of the destination address parameter. The destination address parameter conveys an address identifying the NS user to which the network connection (NC) is to be established. The *DEST_length* parameter accommodates variable length addresses within a range supported by the NS provider.

DEST_offset Specifies the offset of the destination address from the beginning of the **M_PROTO** message block.

CONN_flags This parameter can have the following values:

REC_CONF_OPT
Indicates the use and/or availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

Note: This flag is automatically set by NPI unless the X.25 port packet layer D-bit attribute is set to forbid. Therefore, NPI ignores whether or not this flag is set in this primitive.

EX_DATA_OPT
Indicates the use of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Note: The expedited data transfer service is always allowed with NPI during the data transfer state. Therefore, NPI ignores whether or not the **EX_DATA_OPT** flag is set in this primitive.

QOS_length This should be set to 0 (zero) since QOS parameters are supported.
QOS_offset This should be set to 0 (zero) since QOS parameters are supported.

Acknowledgments

The following acknowledgments are valid for the **N_CONN_REQ** primitive:

Successful The **N_CONN_CON** primitive indicates the NC has been established.
Unsuccessful The **N_DISCON_IND** primitive indicates the NC was not established. A connection may be rejected because either the called NS user cannot be reached, or the NS provider and the called NS user did not agree with the specified facilities.

The **N_ERROR_ACK** primitive indicates non-fatal errors.

Error Codes

The applicable non-fatal errors are defined as follows:

NBADADDR Indicates the network address was in an incorrect length or of zero length. This error code is not intended to indicate NC errors, such as an unreachable destination. These errors types are indicated with the **N_DISCON_IND** primitive.
NBADDATA Indicates the amount of user data specified was outside the range supported by the NS provider.
NBADOPT Indicates the options were either in an incorrect format or contained illegal information.
NOUTSTATE Indicates the primitive was issued from an invalid state. Usually, this error is returned when the connection request is sent on a stream bound as a listener stream. See the N_BIND_REQ primitive for more information.
NSYSERR Indicates a system error. The error is indicated in the **N_ERROR_ACK** primitive.
NODDCUD Indicates an odd-length call user data string.

Implementation Specifics

The **N_CONN_REQ** primitive is part of X.25 Licensed Program.

N_CONN_IND Primitive

Purpose

Indicates a network request connection has been made.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong DEST_length;
   ulong DEST_offset;
   ulong SRC_length;
   ulong SRC_offset;
   ulong SEQ_number;
   ulong CONN_flags;
   ulong QOS_length;
   ulong QOS_offset;
 } N_conn_ind_t;
```

Note: *QOS_length* and *QOS_offset* are 0 (zero) since they are not supported.

Description

The **N_CONN_IND** primitive indicates to an application that it has received an incoming call or connection.

The format of this message is one **M_PROTO** message block normally followed by one or more **M_DATA** blocks for NS user data. The calling application could have sent any integral number of octets of data within the range supported by the NS provider. The NS user data exists only if the corresponding **N_CONN_REQ** primitive specified user data. The data in the **N_CONN_IND** and **N_CONN_REQ** primitives is identical.

Parameters

PRIM_type	Specifies the **N_CONN_IND** primitive.
DEST_length	Specifies the length of the destination address parameter. The destination address parameter conveys an address identifying the NS user to which the NC is to be established.
DEST_offset	Specifies the offset of the destination address from the beginning of the **M_PROTO** message block.
SRC_length	Specifies the source address length. The source address parameter conveys the network address of the NS user from which the NC has been requested. The semantics of the value in the **N_CONN_IND** primitive are identical to the value associated with the stream on which the **N_CONN_REQ** primitive was issued.
SRC_offset	Specifies the offset of the destination address from the beginning of the **M_PROTO** message block.
SEQ_number	Identifies the sequence number that can be used by the NS user to associate this message with a subsequent **N_CONN_RES** or **N_DISCON_REQ** primitive. This value must be unique among the outstanding **N_CONN_IND** primitives. The parameter allows the NS user to issue the **N_CONN_RES** or **N_DISCON_REQ** primitives in any order.

CONN_flags This parameter can have the following values:

REC_CONF_OPT
Indicates the use and/or availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

Note: This flag is automatically set in this primitive by NPI unless the X.25 port packet layer D-bit attribute is set to forbid.

EX_DATA_OPT
Indicates the use of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Note: The expedited data transfer service is always allowed with NPI during the data transfer state. Therefore, this flag is always automatically set by NPI in this primitive.

QOS_length Set to 0.
QOS_offset Set to 0.

Implementation Specifics

The **N_CONN_IND** primitive is part of X.25 Licensed Program.

N_CONN_RES Primitive

Purpose

Allows the destination NPI application to accept an incoming connection request.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong TOKEN_value;
   ulong RES_length;
   ulong RES_offset;
   ulong SEQ_number;
   ulong CONN_flags;
   ulong QOS_length;
   ulong QOS_offset;
} N_conn_res_t;
```

Note: *QOS_length QOS_length* should be 0 (zero) since QOS parameters are not supported.

Description

The format of this message is one **M_PROTO** message block followed by one or more **M_DATA** blocks. The **M_DATA** blocks contain user data which is optional. Any integral number of octets, up to the limit imposed by the underlying network, can be passed. Note that an odd-length CUD string is not allowed.

Parameters

PRIM_type Specifies the **N_CONN_RES** primitive.

TOKEN_value Identifies the stream on which the NS user wants to establish the network connection (NC). The value of this parameter can be one of the following:

>0 Indicates the NS user wants to establish the NC on a stream other than the stream on which the **N_CONN_IND** primitive arrived.

0 Indicates the NS user wants to establish the NC on the same stream on which the **N_CONN_IND** primitive arrived.

The NS user determines the stream's value by issuing an **N_BIND_REQ** primitive with the **TOKEN_REQUEST** flag set. The **N_BIND_ACK** primitive returns the token value.

RES_length Specifies the length of the responding address parameter. The responding address parameter conveys the network address of the NS user to which the NC has been established. Under certain circumstances, such as call redirection or generic addressing, the parameter value may be different from the destination address parameter specified in the corresponding **N_CONN_REQ** primitive.

RES_offset Indicates the offset of the responding address from the beginning of the **M_PROTO** message block.

SEQ_number Indicates the sequence number of the **N_CONN_RES** primitive. The NS provider uses this number to associate the **N_CONN_RES** with an outstanding **N_CONN_IND** message. An invalid sequence number results in an **NBADSEQ** error.

CONN_flags This parameter can have the following values:

REC_CONF_OPT
Indicates the use and/or availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

Note: This flag is automatically set by NPI unless the X.25 port packet layer D-bit attribute is set to forbid. Therefore, NPI ignores whether or not this flag is set in this primitive.

EX_DATA_OPT
Indicates the use of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Note: The expedited data transfer service is always allowed with NPI during the data transfer state. Therefore, NPI ignores whether or not the **EX_DATA_OPT** flag is set in this primitive.

QOS_length This should be set 0 (zero) since QOS parameters are not supported.

QOS_offset This should be set 0 (zero) since QOS parameters are not supported.

Acknowledgments

The NS provider generates one of the following acknowledgments upon receipt of the **N_CONN_RES** primitive:

Successful The **N_OK_ACK** primitive indicates the **N_CONN_RES** primitive succeeded.

Unsuccessful The **N_ERROR_ACK** primitive indicates that a non-fatal error occurred.

Error Codes

The applicable non-fatal errors are defined as follows:

NBADDATA Indicates the amount of user data specified was outside the range supported by the NS provider.

NBADOPT Indicates the options were either in an incorrect format or contained illegal information.

NBADSEQ Indicates the sequence number specified in the primitive was incorrect or illegal.

NOUTSTATE Indicates the primitive was issued from an invalid state.

NSYSERR Indicates a system error. The error is indicated in the primitive.

Implementation Specifics

The **N_CONN_RES** primitive is part of X.25 Licensed Program.

N_CONN_CON Primitive

Purpose

Confirms a network connection.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong RES_length;
   ulong RES_offset;
   ulong CONN_flags;
   ulong QOS_length;
   ulong QOS_offset;
} N_conn_con_t;
```

Note: *QOS_length* and *QOS_offset* are 0 (zero) since they are not supported.

Description

The **N_CONN_CON** primitive informs an NPI application that issued a **N_CONN_REQ** that the remote application accepted the connection.

The format of this primitive is one **M_PROTO** message block followed by one or more **M_DATA** blocks that contain NS user data. In the remote application's response it can send an integral number of octets of user data - within the range supported by the network. User data will be a part of the confirmation only if used in the remote's **N_CONN_RES** primitive. The data in the **N_CONN_CON** and **N_CONN_RES** primitives are always identical.

Parameters

PRIM_type Specifies the **N_CONN_CON** primitive.

RES_length Indicates the length of the responding address parameter. This parameter conveys the network address of the NS user entity to which the network connection (NC) has been established. Under certain circumstances, such as call redirection or generic addressing, the parameter value may be different from the destination address parameter specified in the corresponding **N_CONN_REQ** primitive.

RES_offset Indicates the offset of the responding address from the beginning of the **M_PROTO** message block.

CONN_flags This parameter can have the following values:

REC_CONF_OPT
Indicates the use and/or availability of the receipt confirmation service on the NC. The receipt confirmation service must be provided in the network service to be used on the NC.

Note: This flag is automatically set in this primitive by NPI unless the X.25 port packet layer D-bit attribute is set to forbid.

EX_DATA_OPT
Indicates the use of the expedited data transfer service on the NC. The expedited data transfer service must be provided by the NS provider for it to be used on the NC.

Note: The expedited data transfer service is always allowed with NPI during the data transfer state. Therefore, this flag is always automatically set by NPI in this primitive.

QOS_length 0 (zero) since QOS parameters are not supported.
QOS_offset 0 (zero) since QOS parameters are not supported.

Implementation Specifics

The **N_CONN_CON** primitive is part of X.25 Licensed Program.

N_DATA_REQ Primitive

Purpose

Sends data to the remote application

Structure

The structure of the **M_PROTO** message block, if present, is as follows:

```
typedef struct {
   ulong PRIM_type;
   ulong DATA_xfer_flags;
} N_data_req_t;
```

Description

The **N_DATA_REQ** primitive indicates that the message contains data. This primitive is user-originated and allows transfers of data between applications.

The NS user must send any integral number of octets of data greater than 0. If the size of the NSDU exceeds the network interface data unit (NIDU), the NSDU may be broken up into multiple NIDUs. If an NSDU is segmented into multiple NIDUs, the **N_DATA_REQ** primitive must have the **N_MORE_DATA_FLAG** flag set for each NIDU except the last one. The **N_RC_FLAG** flag can only be set in the **N_DATA_REQ** containing the last NIDU. This is the standard method of working for the X.25 network.

The format of the message is one or more **M_DATA** blocks. Using a **M_PROTO** message block is optional, but it is used for two reasons:

- To indicate the NSDU is broken into multiple NIDUs, and the data contained in the subsequent **M_DATA** message block constitutes one NIDU.
- To indicate that active receipt confirmation is required for this data through use of the X.25 D bit.

Parameters

PRIM_type Specifies the **N_DATA_REQ** primitive.

DATA_xfer_flags Specifies one of the following values:

N_MORE_DATA_FLAG
Indicates that the next **N_DATA_REQ** message (NIDU) is also part of this NSDU. This uses the X.25 M bit. This flag cannot be used with the **N_RC_FLAG** flag.

N_RC_FLAG
Allows the originating NS user to request confirmation of receipt of the **N_DATA** primitive. The **N_DATACK** primitives provide confirmation. The parameter can only be present if the NS users and provider agreed to use receipt confirmation during connection establishment. This uses the X.25 D bit. This flag cannot be used with the **N_MORE_DATA_FLAG** flag.

N_Q_FLAG
Indicates that the Qualifier bit (Q-bit) should be set by NPI in the X.25 data packet. This is used by applications that wish to indicate a difference between data and the applications internal command data.

Acknowledgments

This primitive does not require any acknowledgments. However, the primitive may generate a fatal error. The STREAMS **M_ERROR** message type notifies the application of an **EPROTO** error. As a result of the error, all system calls on that stream will fail.

Error Codes

EPROTO Indicates one of the following unrecoverable protocol conditions:

- The network interface was in an incorrect state.
- The amount of NS user data associated with the primitive is outside the range supported by the NS provider. The *NIDU_size* parameter of the **N_INFO_ACK** primitive determines the range.
- The requested option is either not supported by the NS provider or was not specified with the **N_CONN_REQ** primitive.
- The **M_PROTO** message block was not followed by one or more **M_DATA** message blocks.
- The **N_RC_FLAG** and **N_MORE_DATA_FLAG** flags were both set in the primitive, or the flags parameter contained an unknown value.

Implementation Specifics

The **N_DATA_REQ** primitive is part of X.25 Licensed Program.

N_DATA_IND Primitive

Purpose

Indicates that the current message contains application data.

Structure

The structure of the **M_PROTO** message block, if present, is as follows:

```
typedef struct {
   ulong PRIM_type;
   ulong DATA_xfer_flags;
} N_data_ind_t;
```

Otherwise, the message consists of one or more **M_DATA** message blocks.

Description

The **N_DATA_IND** primitive indicates to the NS user that this message contains NS user data. This primitive originates from the network provider and is a response to the **N_DATA_REQ** primitive. The network service data unit (NSDU) can be segmented into more than one network interface data units (NIDUs). The **MORE_DATA_FLAG** associates NIDUs with the NSDU. The **RC_FLAG** flag can be set only on the last NIDU.

The format of the message is one or more **M_DATA** message blocks. The value of the NS user data field is always the same as that supplied in the corresponding **N_DATA_REQ** primitive at the peer service access point. **M_PROTO** message blocks are optional.

Parameters

PRIM_type Specifies the **N_DATA_IND** primitive.

DATA_xfer_flags Specifies one of the following flags:

N_MORE_DATA_FLAG
Indicates that the next **N_DATA_IND** primitive (NIDU) is part of the current NSDU. The NSDU is what was considered by the remote application to a whole unit of data.

N_RC_FLAG
Indicates whether confirmation is requested. The *DATA_xfer_flags* parameter can have this value only if both applications and the network agreed its use while the network connection (NC) was being established. This parameter value is always identical to that supplied in the corresponding **N_DATA_REQ** primitive.

N_Q_FLAG
Indicates that the qualifier bit (Q-bit) was set by the remote application.

Implementation Specifics

The **N_DATA_IND** primitive is part of X.25 Licensed Program.

N_DATACK_REQ Primitive

Purpose

Acknowledges the receipt of data which had the **N_DATA_ACK** flag set.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_datack_req_t;
```

Description

The **N_DATACK_REQ** primitive acknowledges receipt of an **N_DATA_IND** primitive which had the receipt confirmation parameter set. The NPI application should send a **N_DATACK_REQ** on receipt of such data.

Parameters

PRIM_type Specifies the **N_DATACK_REQ** primitive.

Acknowledgments

This primitive does not require any acknowledgments. However, the primitive may generate a fatal error. The STREAMS **M_ERROR** message type notifies the NS user of an **EPROTO** error. As a result of the error, all system calls on that stream will fail.

Error Codes

NSYSERR Indicates a system error. The error is indicated in the **N_ERROR_ACK** primitive.

Implementation Specifics

The **N_DATACK_REQ** primitive is part of X.25 Licensed Program.

N_DATACK_IND Primitive

Purpose

Indicates the remote NPI application has acknowledged data sent to it with the receipt acknowledge flag set.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_datack_ind_t;
```

Description

The **N_DATACK_IND** primitive indicates to the local NPI application that the remote has acknowledged the data that had previously been sent with the receipt confirmation set. The **N_DATACK_IND** primitive is a received based on the remote application submitting a **N_DATACK_REQ** primitive.

Parameters

PRIM_type Specifies the **N_DATACK_IND** primitive.

Implementation Specifics

The **N_DATACK_IND** primitive is part of X.25 Licensed Program.

N_EXDATA_REQ Primitive

Purpose

Requests an expedited data transfer.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_exdata_req_t;
```

Description

The **N_EXDATA_REQ** primitive requests that the network service (NS) provider send an X.25 interrupt packet. The primitive consists of an **M_PROTO** message block followed by one or more **M_DATA** blocks. The **M_PROTO** portion should contain the **N_exdata_req_t** structure, and the **M_DATA** block should contain the data to be sent in the corresponding X.25 interrupt packet. The length of the data can be 1-32 bytes, depending on the X.25 CCITT level set in the X.25 port packet parameters. Note that the 1980 CCITT level only allows 1 byte of X.25 interrupt data, but 1984 and 1988 CCITT levels allow up to 32 bytes of data.

> **Note:** The expedited data transfer service is always allowed with NPI during the data transfer state. Therefore, NPI ignores whether or not the user set the **EX_DATA_OPT** flag in the **N_CONN_REQ** or **N_CONN_RES** primitives.

Parameters

PRIM_type Specifies the **N_EXDATA_REQ** primitive.

Acknowledgments

Successful The **N_OK_ACK** primitive indicates that the X.25 interrupt sequence completed successfully and the application may now send another **N_EXDATA_REQ** primitive.

Unsuccessful An **N_ERROR_ACK** primitive indicates non-fatal errors.

An **M_ERROR** STREAMS message indicates a fatal/unrecoverable error. This error results in the failure of all system calls on that stream.

Error Codes

The applicable non-fatal errors are defined as follows:

NNOINTCF Indicates that the NS user sent the **N_EXDATA_REQ** primitive to the NS provider before the user's previous **N_EXDATA_REQ** primitive completed. Each expedited data sequence must complete before another one may be requested by the same user.

The applicable fatal/unrecoverable errors are defined as follows:

EPROTO This indicates one of the following unrecoverable protocol conditions:

- The network interface was found to be in an incorrect state. Expedited data transfer requests are only valid during the data transfer state.
- The **N_EXDATA_REQ** primitive was received without an **M_DATA** block.

Implementation Specifics

The **N_EXDATA_REQ** primitive is part of the X.25 licensed program.

N_EXDATA_IND Primitive

Purpose

Indicates the receipt of expedited data.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_exdata_ind_t;
```

Description

This primitive consists of one **M_PROTO** message block followed by one or more **M_DATA** blocks. The value of the data in the **M_DATA** blocks is identical to that supplied with the corresponding **N_EXDATA_REQ** primitive. The **N_EXDATA_IND** primitive indicates that the network service (NS) provider received an interrupt packet from the network.

> **Note:** The expedited data transfer service is always allowed with NPI during the data transfer state. NPI ignores whether or not the user set the **EX_DATA_OPT** flag in the **N_CONN_REQ** or **N_CONN_RES** primitives. Therefore, all users must be able to handle receiving **N_EXDATA_IND** primitives from NPI.

Parameters

PRIM_type Specifies the **N_EXDATA_IND** primitive.

Implementation Specifics

The **N_EXDATA_IND** primitive is part of the X.25 licensed program.

N_RESET_REQ Primitive

Purpose

Requests that the network connection be reset.

Syntax

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong RESET_reason;
   uchar cause;
   uchar diagnostic;
} N_reset_req_t;
```

Description

The **N_RESET_REQ** primitive requests that the network service (NS) provider reset the network connection. The NS user initiates this primitive.

Parameters

PRIM_type	Specifies the **N_RESET_REQ** primitive.
RESET_reason	Indicates the cause of the reset. The value of this parameter is always **N_USER_RESYNC**.
cause	CCITT cause value of the reset. This field can be optionally assigned.
diagnostic	CCITT diagnostic value of the reset. This field can be optionally assigned.

Acknowledgments

Successful	This primitive does not require an immediate acknowledgment. However, when the reset completes, the issuer of this primitive receives an **N_RESET_CON** primitive.

Unsuccessful — The **N_ERROR_ACK** primitive acknowledges non-fatal errors. The resulting state remains unchanged.

Error Codes

The following non-fatal error codes are valid:

NOUTSTATE — Indicates the primitive was issued from an invalid state.
NSYSERR — Indicates a system error. The error is indicated by the **N_ERROR_ACK** primitive.

Implementation Specifics

The **N_RESET_REQ** primitive is part of X.25 Licensed Program.

N_RESET_IND Primitive

Purpose

Indicates the connection has been reset.

Syntax

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong RESET_orig;
   ulong RESET_reason;
   uchar cause;
   uchar diagnostic;
} N_reset_ind_t;
```

Description

The **N_RESET_IND** primitive indicates to the network service user that the network connection has been reset. The network provider originates this primitive.

Parameters

PRIM_type — Specifies the **N_RESET_IND** primitive.
RESET_orig — Indicates the source of the reset. Its value can be one of the following:

N_PROVIDER
Indicates the NS provider or network originated the reset.

N_USER
Indicates the remote application originated the reset.

N_UNDEFINED
Indicates the reset originator was undefined.

RESET_reason Indicates the cause of the reset.

- If the value of the *RESET_orig* parameter is **N_PROVIDER**, the value is one of the following:

  N_CONGESTION
  Indicates a reset due to congestion.

  N_RESET_UNSPECIFIED
  Indicates a reset due to an unspecified reason.

  N_NET_LINK_DOWN
  Indicates that the network or link is down.

  N_NET_LINK_UP
  Indicates that the network or link is up.

  Note: Once a **N_NET_LINK_DOWN** reset indication is received, a **N_RESET_RES** should be sent in response. After the **N_RESET_RES**, no other primitives should be sent until a subsequent **N_NET_LINK_UP** reset indication is received. Other primitives issued will receive a **N_NET_LINK_DOWN** reset in response.
- If the value of the *RESET_orig* parameter is **N_USER**, the value is:

  N_USER_RESYNC
  Indicates a user resynchronization.
- If the value of the *RESET_orig* parameter is **N_UNDEFINED**, the value is:

  N_REASON_UNDEFINED
  Indicates a reset due to an undefined reason.

cause CCITT cause value of the reset. This field can be optionally checked.
diagnostic CCITT diagnostic value of the reset. This field can be optionally checked.

Implementation Specifics

The **N_RESET_IND** primitive is part of X.25 Licensed Program.

N_RESET_RES Primitive

Purpose

Acknowledges receipt of a reset indication.

Syntax

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_reset_res_t;
```

Description

The **N_RESET_RES** primitive indicates that the NPI application has accepted a reset request. This primitive must be issued before use of the connection can be restored after a reset has been indicated.

Parameters

PRIM_type Specifies the **N_RESET_RES** primitive.

Acknowledgments

Successful	The **N_OK_ACK** primitive indicates the successful completion of this primitive. This results in the data transfer state.
Unsuccessful	The **N_ERROR_ACK** primitive indicates this primitive was unsuccessful. The resulting state remains the same.

Error Codes

The following non-fatal error codes are valid:

NOUTSTATE	Indicates the primitive was issued from an invalid state.
NSYSERR	Indicates a system error. The error is indicated in the **N_ERROR_ACK** primitive.

Implementation Specifics

The **N_RESET_RES** primitive is part of X.25 Licensed Program.

N_RESET_CON Primitive

Purpose

Provides confirmation from the network that it received the **N_RESET_RES** primitive.

Syntax

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
} N_reset_con_t;
```

Description

The **N_RESET_CON** primitive informs the NPI application that the reset previously issued with the **N_RESET_RES** primitive has completed. The NS provider can issue the **N_RESET_CON** primitive to the application before receiving an **N_RESET_RES** primitive from the remote.

Parameters

PRIM_type	Specifies the **N_RESET_CON** primitive.

Implementation Specifics

The **N_RESET_CON** primitive is part of X.25 Licensed Program.

N_DISCON_REQ Primitive

Purpose

Requests that an existing connection be disconnected.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong DISCON_reason;
```

```
   ulong RES_length;
   ulong RES_offset;
   ulong SEQ_number;
   uchar cause;
   uchar diagnostic;
} N_discon_req_t;
```

(Optionally followed by clear user data.)

Description

The **N_DISCON_REQ** primitive requests that the network service (NS) provider either disconnect an existing connection or deny a request for a network connection. This primitive is user-originated.

The format of the message is one **M_PROTO** message block, followed by one or more **M_DATA** message blocks that indicate user data. Any integral number of octets of data, up to the network's clear user data limit, can be sent provided the length is even.

Parameters

PRIM_type Specifies the **N_DISCON_REQ** primitive.

DISCON_reason This value is ignored by NPI. The corresponding X.25 CLEAR packet that is generated from the **N_DISCON_REQ** will always have a diagnostic of 0x00.

RES_length Indicates the length of the responding address parameter. The responding address parameter is optional and is present if the primitive indicates rejection of an attempt by an NS user to establish a network connection. The responding address parameter conveys the network address of the NS user entity from which the **N_DISCON_REQ** primitive was issued. Under certain circumstances, such as call redirection and generic addressing, the value of the responding address parameter may be different from the destination address in the corresponding **N_CONN_REQ** primitive.

RES_offset Specifies where the responding address begins. The value of this parameter is the offset from the beginning of the **M_PROTO** message block.

SEQ_number Identifies the sequence number of an **N_CONN_IND** primitive. This parameter can have two types of values:

- **>0** Identifies the sequence number of the rejected **N_CONN_IND** message. NPI uses this number to associate the **N_DISCON_REQ** primitive with an unacknowledged **N_CONN_IND** primitive that is to be rejected.
- **0** Indicates the **N_DISCON_REQ** primitive is rejecting an established network connection (NC).

cause CCITT cause value of the reset. This field can be optionally assigned.

diagnostic CCITT diagnostic value of the reset. This field can be optionally assigned.

Acknowledgments

The NS provider should generate one of the following acknowledgments upon receipt of this primitive:

Successful Indicated by the **N_OK_ACK** primitive.

Unsuccessful Non-fatal errors are indicated by the **N_ERROR_ACK** primitive.

Error Codes

The applicable non-fatal errors are as follows:

NOUTSTATE Indicates the primitive was issued from an invalid state.

NBADDATA Indicates the amount of user data specified was outside the range supported by the NS provider.

NSYSERR Indicates a system error. The error is indicated in the primitive.

NBADSEQ Indicates one of the following conditions:

- The specified sequence number referred to an invalid **N_CONN_IND** primitive.
- The primitive attempted to reject an established NC, but the value of the sequence number was not 0.
- The primitive attempted to reject an unconfirmed **N_CONN_REQ** primitive, but the value of the sequence number was not 0.

Implementation Specifics

The **N_DISCON_REQ** primitive is part of X.25 Licensed Program.

N_DISCON_IND Primitive

Purpose

Indicates a disconnected or denied connection.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong PRIM_type;
   ulong DISCON_orig;
   ulong DISCON_reason;
   ulong RES_length;
   ulong RES_offset;
   ulong SEQ_number;
   uchar cause;
   uchar diagnostic;
} N_discon_ind_t;
```

Description

The **N_DISCON_IND** primitive indicates to the NPI application that either an existing connection has been disconnected or a request for connection has been denied. The network provider generates this primitive.

The format of the message is one **M_PROTO** message block, followed by one or more **M_DATA** blocks. The value of the user data parameter is identical to the user data in the corresponding **N_DISCON_REQ** primitive sent by the remote application. If the remote application did not initiate the disconnect, then the **N_DISCON_REQ** primitive will not contain user data.

Parameters

PRIM_type Specifies the **N_DISCON_IND** primitive.

DISCON_orig Indicates the source of the network connection (NC) release and represents one of the following values:

N_PROVIDER
Indicates an NS provider-originated disconnect.

N_USER
Indicates an NS user-originated disconnect.

N_UNDEFINED
Indicates an undefined disconnect originator. This value is not permitted when an NS user or the NS provider issues an **N_DISCON_IND** primitive to reject an NC establishment attempt.

DISCON_reason All calls are cleared with a diagnostic of 0 (zero).

RES_length Indicates the length of the responding address parameter. This parameter is optional and is present only if the primitive indicates the rejection of an NS user's attempt to establish a network connection. If the responding address parameter is not present, the value of this parameter is 0. Otherwise, the value of the disconnect address parameter is identical to that supplied with the corresponding **N_DISCON_REQ** primitive.

RES_offset Specifies the offset from the beginning of the **M_PROTO** message block where the network address begins.

SEQ_number Identifies the sequence number of an **N_CONN_IND** primitive. This parameter can have two types of values:

>0 Identifies the sequence number associated with the **N_CONN_IND** primitive that is being terminated.

0 Indicates either a previously issued **N_CONN_REQ** primitive has been rejected, or an established NC has been released.

When the value of this parameter is nonzero and matches the sequence number assigned to an unacknowledged **N_CONN_IND** primitive, the value of the *SEQ_number* parameter indicates that the NS provider is canceling the unacknowledged **N_CONN_IND** primitive.

cause CCITT cause value of the reset. This field can be optionally checked.

diagnostic CCITT diagnostic value of the reset. This field can be optionally checked.

Implementation Specifics

The **N_DISCON_IND** primitive is part of X.25 Licensed Program.

Chapter 6. Data Link Provider Interface Programming Reference

This chapter discusses the Data Link Provider Interface (DLPI) which provides access to the LAP-B frame layer, including the primitives necessary to run DLPI.

The Data Link Provider Interface (DLPI) enables a data link service user to access and use the frame or LAP-B layer of the protocol stack. To access the frame layer directly on a port, the port's configuration must be set to allow it. With the frame layer access active, regular X.25 applications cannot use the port.

The interface provides point-to-point data communications and is designed for two systems connected back to back over a communications link. This is provided through a connection-oriented subset of the full DLPI specification. Other implementations of DLPI exist, and shipped with the operating system's portable STREAMS environment is a connectionless-oriented subset. The X.25 Licensed Program only supports the connection-oriented subset of the specification that is shipped with the product.

On AIXlink v1.1, with X.25 ports that use the **twd** device driver, the frame layer resided on the adapter. As a result, it was necessary to encode DLPI primitives in commands to the hardware device driver, **twd**. The application's DLPI primitives were passed to the frame layer through the **twd** driver. In AIXlink v2.0, the frame layer resides in the kernel for **twd** and **hdlc** device drivers. Therefore it is no longer necessary to encode DLPI primitives in commands to the hardware device driver, **twd**; DLPI primitives are issued directly to the frame layer.

For X.25 ports that use the **hdlc** device driver, the frame layer resides in the kernel. In this case it is not necessary (or appropriate) to encode the DLPI primitives. They are issued directly to the frame layer.

See the sample programs (referenced below) for the subroutine called **dlpi_open**. This subroutine provides a single open interface that isolates these base device driver differences form DLPI applications.

For additional information on the **twd** device driver, see X.25 Licensed Program Functionality .

Structure Changes for 64-bit Mode

All structures using the **ulong** definition have been changed to use **ulong32int64_t** instead of **ulong**. This is for use with the 64-bit kernel. However, AIXlink applications do not need to change definitions from **ulong** to **ulong32int64_t** since they run in 32-bit mode. In 32-bit mode **ulong32int64_t** and **ulong** definitions take up 4 bytes of space.

The Data Link Layer

The data link layer (layer 2 in the OSI Reference Model) is responsible for the transmission and error-free delivery of information over a physical communications medium. This layer is also referred to as the frame or LAP-B layers .

The model of the data link layer is presented here to describe concepts that are used throughout the specification of DLPI. It is described in terms of an interface architecture, as well as addressing concepts needed to identify different components of that architecture.

Model of the Service Interface

Each layer of the OSI Reference Model has two standards:

- One that defines the services provided by the layer.
- One that defines the protocol through which layer services are provided.

DLPI is an implementation of the first type of standard. It specifies an interface to the services of the data link layer. The following diagram depicts an overview of DLPI.

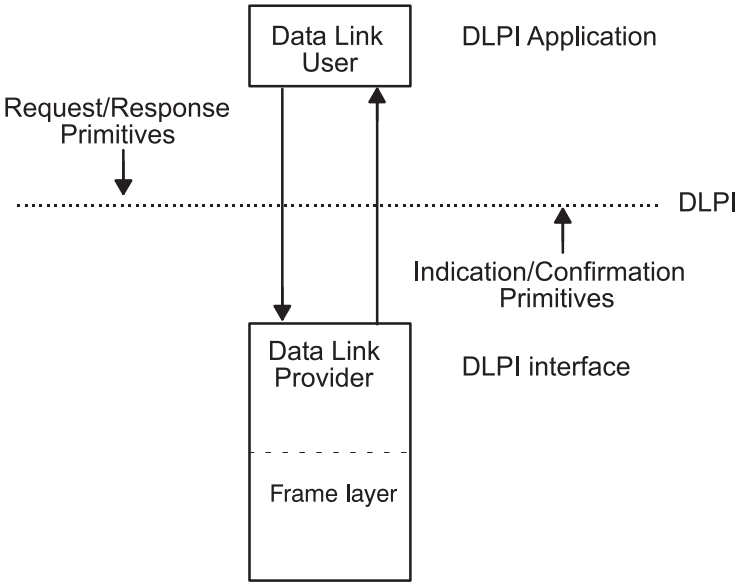


Figure 10. Overview of DLPI

The data link interface is the boundary between the network and data link layers of the OSI Reference Model. The network layer entity, which is usually the X.25 layer , is the user of the services of the data link interface. This user is DLPI's user application and is sometimes referred to as the Data Link Service (DLS) user . The DLPI layer which provides the programming interface is referred to as the DLS provider. This interface consists of a set of primitives that provide access to the data link layer services.

The service primitives that make up kernel-level interfaces are defined as STREAMS messages that are transferred between the user and provider of the service. DLPI is targeted for STREAMS protocol modules that either use or provide data link services. In addition, user programs that wish to access a STREAMS-based data link provider directly may do so using the **putmsg** and **getmsg** subroutines.

The use of DLPI is demonstrated in a set of sample programs located in the **/usr/samples/sx25/npi** directory for AIXlink/X.25 Version 2.0.

DLPI Primitives

Before DLPI primitives can be used by the application, the user must establish a stream to DLPI. It does this by binding to the frame layer that has been made available in the streams environment through the X.25 Licensed Program's installation. Once bound to a stream, the application can send primitives to DLPI and connect and transfer data. When an application wishes to terminate it disconnects the link and unbinds from its stream. The primitives are split into two groups:

- Local management service primitives which control peripheral setup requirements such as the bind
- Connection-mode service primitives associated with connecting, disconnecting and data transfer

Local Management Service Primitives

The following primitives support the information reporting, attach, and bind services of DLPI. Once a stream has been opened by a DLS user, these primitives initialize the stream and prepare it for use.

DL_INFO_ACK	Returns information about the stream.
DL_BIND_REQ	Requests the DLS provider to bind a DLSAP to a stream.
DL_BIND_ACK	Reports a DLSAP has been bound.
DL_UNBIND_REQ	Requests the DLS provider unbind a DLSAP.
DL_OK_ACK	Indicates a request primitive was received.
DL_ERROR_ACK	Indicates a previously issued request or response was invalid.

Connection-Mode Service Primitives

This section describes the service primitives that support the connection-mode service of the data link layer. These primitives support:

- Connection establishment
- Data transfer
- Connection release
- Reset

Connection Establishment Primitives

The following DLPI primitives establish data link connections:

DL_CONNECT_REQ	Requests a data link connection with a remote user.
DL_CONNECT_CON	Indicates a data link connection has been established.

The calling application initiates the connection by issuing a **DL_CONNECT_REQ** primitive. The **DL_CONNECT_CON** primitive informs the calling application that the connection has been established.

Once the connection is established, data is exchanged using the STREAMS **putmsg** and **getmsg** subroutines.

The data transfer service provides for the exchange of user data in either direction or in both directions simultaneously between the applications. Data is transmitted in local groups called DLSDUs. The DLS provider preserves both the sequence and boundaries of DLSDUs as they are transmitted.

Normal data transfer is neither acknowledged nor confirmed. It is up to the DLS users, if they so choose, to implement a confirmation protocol. Refer to the DLPI sample programs for an example of coding data frames.

Connection Release Primitives

The following DLPI primitives provide data link connection release service:

DL_DISCONNECT_IND	Requests the DLS provider to disconnect a connection.
DL_DISCONNECT_REQ	Confirms the link has been disconnected.

The connection release service allows DLS users or the DLS provider to initiate the connection release. Any data in that has not been delivered to the DLS user when the connection release is requested can be discarded. Normally, one DLS user requests disconnection and the DLS provider issues an indication of the ensuing release to the other DLS user.

Reset Primitives

The following DLPI primitives provide reset service:

DL_RESET_REQ	Requests the DLS provider to resynchronize a data link connection.
DL_RESET_IND	Informs the DLS user a remote DLS user is resynchronizing the connection.
DL_RESET_RES	Directs the DLS provider to complete resynchronizing the connection.
DL_RESET_CON	Confirms the reset has been completed.

The reset service may be used by the DLS user to resynchronize the use of a data link connection, or by the DLS provider to report detected loss of data unrecoverable within the data link service.

If the data link connection is congested, running the reset service unblocks the flow of DLSDUs. DLSDUs may be discarded by the DLS provider. The DLS user or users that did not invoke the reset are notified that a reset has occurred. A reset may require a recovery procedure to be performed by the DLS users.

DL_BIND_ACK Primitive for X.25

Purpose

Reports the successful bind of a data link service access point (DLSAP) to a stream.

Structure

This primitive consists a message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_sap;
   ulong dl_addr_length;
   ulong dl_addr_offset;
   ulong dl_max_conind;
   ulong dl_xidtest_flg;
} dl_bind_ack_t;
```

Description

The **DL_BIND_ACK** primitive reports the successful bind of an application to a stream and returns the bound DLSAP address to the DLS user. This primitive is generated in response to a **DL_BIND_REQ** primitive.

Parameters

dl_primitive	Specifies the **DL_BIND_ACK** primitive.
dl_sap	Specifies the DLSAP address information associated with the bound DLSAP. It corresponds to the *dl_sap* parameter of the associated **DL_BIND_REQ** primitive, which contains all or part of the DLSAP address. For the portion of the DLSAP address specified in the **DL_BIND_REQ** primitive, this parameter contains the corresponding portion of the address for the bound DLSAP.
dl_addr_length	For point to point addressing is not supported, so address length and offset are both set to 0.
dl_addr_offset	Set to 0 on return.
dl_max_conind	Set to 0 on return.
dl_xidtest_flg	Set to 0 on return.

Implementation Specifics

The **DL_BIND_ACK** primitive is part of X.25 Licensed Program.

DL_BIND_REQ Primitive for X.25

Purpose

The application requests a stream to be bound for the application to the frame layer, thus producing a data link service access point (DLSAP) to the stream.

Structure

This primitive consists of a message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_sap;
   ulong dl_max_conind;
   ushort dl_service_mode;
   ushort dl_conn_mgmt;
   ulong dl_xidtest_flg;
} dl_bind_req_t;
```

Description

The **DL_BIND_REQ** primitive requests the DLS provider to bind a DLSAP to a stream. The DLS user must identify the address of the DLSAP to be bound as well as indicate whether it will accept incoming connect requests on the stream. The request directs the DLS provider to activate the stream associated with the DLSAP.

A stream is active when the DLS provider can transmit and receive protocol data units destined to or originating from the stream. The physical point of attachment (PPA) associated with each stream must be initialized when the **DL_BIND_REQ** primitive has been processed. The PPA is initialized when the **DL_BIND_ACK** primitive is received. If the PPA cannot be initialized, the **DL_BIND_REQ** primitive fails.

A stream can be bound as a connection management stream, which receives all connect requests that arrive through a given PPA. In this case, the *dl_sap* parameter is ignored.

Parameters

dl_primitive Specifies the **DL_BIND_REQ** primitive.

dl_sap The address format used by the frame layer is:

byte 1 Master station (1) or Slave station (0)

byte 2 Logical Line number

byte 3 Must be set to 0

byte 4 Not used, set to 0

where byte 1 is the most significant byte of the ulong, and byte 4 is the least significant.

dl_max_conind Should be set to 0.

dl_service_mode Should be set to 0.

dl_conn_mgmt Should be set to 0.

dl_xidtest_flg Should be set to 0.

States

Valid The message is valid in the **DL_UNBOUND** state.

New The resulting state is **DL_BIND_PENDING**.

Acknowledgments

Successful	The **DL_BIND_ACK** primitive is sent to the DLS user. The resulting state is the **DL_IDLE** state.
Unsuccessful	The **DL_ERROR_ACK** primitive is returned. The resulting state is unchanged.

Error Codes

DL_BADPPA	Line number passed is too large
DL_INITFAILED	Line not associated with a physical line
DL_BUSY	Line already bound
DL_BADSAP	Byte 1 of *dl_sap* not valid master or client SAP value
DL_BADADDR	Bytes 3 and 4 not zero

Implementation Specifics

The **DL_BIND_REQ** primitive is part of X.25 Licensed Program.

DL_CONNECT_CON Primitive for X.25

Purpose

Informs the local data link service (DLS) user that the requested data link connection has been established.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_resp_addr_length;
   ulong dl_resp_addr_offset;
   ulong dl_qos_length;
   ulong dl_qos_offset;
   ulong dl_growth;
} dl_connect_con_t;
```

Description

The **DL_CONNECT_CON** primitive informs the local DLS user that the requested data link connection has been established. The primitive contains the data link service access point (DLSAP) address of the responding DLS user as well as the quality of service (QOS) parameters selected by the responding DLS user.

Parameters

dl_primitive	Specifies the **DL_CONNECT_CON** primitive.

Addressing and quality of service parameters are not supported; therefore, each of the following parameters will be set to 0.

- *dl_resp_addr_length*
- *dl_resp_addr_offset*
- *dl_qos_length*
- *dl_qos_offset*

dl_growth Defines a growth field for future enhancements to this primitive. Its value must be set to 0.

States

Valid The message is valid in the **DL_OUTCON_PENDING** state.
New The resulting state is **DL_DATAXFER**.

Implementation Specifics

The **DL_CONNECT_CON** primitive is part of X.25 Licensed Program.

DL_CONNECT_REQ Primitive for X.25

Purpose

Requests data link connection between the data link service (DLS) provider and a remote DLS user.

Structure

This primitive consists of a message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_dest_addr_length;
   ulong dl_dest_addr_offset;
   ulong dl_qos_length;
   ulong dl_qos_offset;
   ulong dl_growth;
} dl_connect_req_t;
```

Description

The **DL_CONNECT_REQ** primitive requests that a DLS provider establish a data link connection with a remote DLS user.

Parameters

dl_primitive Specifies the **DL_CONNECT_REQ** primitive.

Addressing and quality of service parameters are not supported; therefore, each of the following parameters will be set to 0.

- *dl_resp_addr_length*
- *dl_resp_addr_offset*
- *dl_qos_length*
- *dl_qos_offset*

dl_growth Defines a growth field for future enhancements to this primitive. Its value must be set to 0.

States

Valid The message is valid in the **DL_IDLE** state.
New The resulting state is **DL_OUTCON_PENDING**.

Acknowledgments

The **DL_CONNECT_REQ** primitive has no immediate response. However, if the connect request is accepted by the called DLS user, the **DL_CONNECT_CON** primitive is sent to the calling DLS user, resulting in the **DL_DATAXFER** state.

If the connect request is rejected by the called DLS user, the called DLS user cannot be reached, or the DLS provider or called DLS user do not agree on the specified quality of service, a **DL_DISCONNECT_IND** is sent to the calling DLS user, resulting in the **DL_IDLE** state.

If the request is erroneous, the **DL_ERROR_ACK** primitive is returned and the resulting state is unchanged.

DL_DISCONNECT_IND Primitive for X.25

Purpose

Informs the data link service (DLS) user that the data link connection on the current stream has been disconnected.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_originator;
   ulong dl_reason;
   ulong dl_correlation;
} dl_disconnect_ind_t;
```

Description

The **DL_DISCONNECT_IND** primitive informs the DLS user of one of the following conditions:

- The data link connection on the current stream has been disconnected.
- A pending connection from either the **DL_CONNECT_REQ** or **DL_CONNECT_IND** primitive has been cancelled.

The primitive indicates the origin and the cause of the disconnect.

Parameters

dl_primitive Specifies the **DL_DISCONNECT_IND** primitive.

dl_originator Specifies one of the following:

DL_USER
Indicates whether the disconnect originated from a DLS user.

DL_PROVIDER
Indicates whether the disconnect originated from a DLS provider.

dl_reason Specifies the reason for the disconnect. Reasons for disconnect are:

DL_DISC_PERMANENT_CONDITION
Indicates the connection was released due to a permanent condition.

DL_DISC_TRANSIENT_CONDITION
Indicates the connection was released due to a temporary condition.

DL_CONREJ_DEST_UNKNOWN
Indicates the connect request has an unknown destination.

DL_CONREJ_DEST_UNREACH_PERMANENT
Indicates the connection was released because the destination for the connect request could not be reached. This is a permanent condition.

DL_CONREJ_DEST_UNREACH_TRANSIENT
Indicates the connection was released because the destination for the connect request could not be reached. This is a temporary condition.

DL_CONREJ_QOS_UNAVAIL_PERMANENT
Indicates the requested quality of service parameters became permanently unavailable while establishing a connection.

DL_CONREJ_QOS_UNAVAIL_TRANSIENT
Indicates the requested quality of service parameters became temporarily unavailable while establishing a connection.

DL_DISC_UNSPECIFIED
Indicates the connection was closed due to an unspecified reason.

dl_correlation If the value is nonzero, specifies the correlation number contained in the **DL_CONNECT_IND** primitive being cancelled. This value permits the DLS user to associate the message with the proper **DL_CONNECT_IND** primitive. If the disconnect request indicates the release of a connection that is already established, or is indicating the rejection of a previously sent **DL_CONNECT_REQ** primitive, the value of the *dl_correlation* parameter is 0.

States

Valid The message is valid in any of the following states:

- **DL_DATAXFER**
- **DL_INCON_PENDING**
- **DL_OUTCON_PENDING**
- **DL_PROV_RESET_PENDING**
- **DL_USER_RESET_PENDING**

New The resulting state is **DL_IDLE**.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_DISCONNECT_REQ Primitive for X.25

Purpose

Requests that an active data link be disconnected.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_reason;
   ulong dl_correlation;
} dl_disconnect_req_t;
```

Description

The **DL_DISCONNECT_REQ** primitive requests the data link service (DLS) provider to disconnect an active data link connection or one that was in the process of activation. The **DL_DISCONNECT_REQ** primitive can be sent in response to a previously issued **DL_CONNECT_IND** or **DL_CONNECT_REQ** primitive. If an incoming **DL_CONNECT_IND** primitive is being refused, the correlation number associated with that connect indication must be supplied. The message indicates the reason for the disconnect.

Parameters

dl_primitive Specifies the **DL_DISCONNECT_REQ** primitive.

dl_reason Specifies one of the following reasons for the disconnect:

DL_DISC_NORMAL_CONDITION
Indicates a normal release of a data link connection.

DL_DISC_ABNORMAL_CONDITION
Indicates an abnormal release of a data link connection.

DL_CONREJ_PERMANENT_COND
Indicates a permanent condition caused the rejection of a connect request.

DL_CONREJ_TRANSIENT_COND
Indicates a transient condition caused the rejection of a connect request.

DL_DISC_UNSPECIFIED
Indicates the reason for the disconnect was not specified.

dl_correlation Specifies one of the following values:

0 Indicates either the disconnect request is releasing an established connection or is cancelling a previously sent **DL_CONNECT_REQ** primitive.

>0 Specifies the correlation number that was contained in the **DL_CONNECT_IND** primitive being rejected. This value permits the DLS provider to associate the primitive with the proper **DL_CONNECT_IND** primitive when the provider rejects an incoming connection.

States

Valid The message is valid in any of the following states:

- **DL_DATAXFER**
- **DL_INCON_PENDING**
- **DL_OUTCON_PENDING**
- **DL_PROV_RESET_PENDING**
- **DL_USER_RESET_PENDING**

New The resulting state is one of the disconnect pending states.

Acknowledgments

Successful The **DL_OK_ACK** primitive is sent to the DLS user, resulting in the **DL_IDLE** state.

Unsuccessful The **DL_ERROR_ACK** primitive is returned, and the resulting state is unchanged.

Error Codes

DL_BADCORR Indicates the correlation number specified in this primitive did not correspond to a pending connect indication.

DL_OUTSTATE Indicates the primitive was issued from an invalid state.

DL_SYSERR Indicates a system error, which is specified in the **DL_ERROR_ACK** primitive.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_ERROR_ACK Primitive for X.25

Purpose

Informs the data link service (DLS) user that a request or response was invalid.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_error_primitive;
   ulong dl_errno;
   ulong dl_unix_errno;
} dl_ok_ack_t;
```

Description

The **DL_ERROR_ACK** primitive informs the DLS user that the previously issued request or response was invalid. The primitive identifies the primitive in error, specifies a data link provider interface (DLPI) error code, and if appropriate, indicates an operating system error code.

Parameters

dl_primitive Specifies the **DL_ERROR_ACK** primitive.

dl_error_prim Identifies the primitive that caused the error.

dl_errno Specifies the DLPI error code associated with the failure. See the individual request or response for the error codes that are applicable. In addition to those errors, it can have the following values:

DL_BADPRIM
Indicates an unrecognized primitive was issued by the DLS user.

DL_NOTSUPPORTED
Indicates an unsupported primitive was issued by the DLS user.

DL_SYSERR
Reports operating system failures that prevent the processing of a given request or response.

dl_unix_errno Specifies the operating system error code associated with the failure. This value should be nonzero only when the *dl_errno* parameter is set to **DL_SYSERR**.

States

Valid The message is valid in all states that have a pending acknowledgment or confirmation.

New The resulting state is the same as the one from which the acknowledged request or response was generated.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_INFO_ACK Primitive for X.25

Purpose

Returns information about the Data Link Provider Interface (DLPI) stream in response to the **DL_INFO_REQ** primitive.

Structure

This primitive consists of one **M_PCPROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_max_sdu;
   ulong dl_min_sdu;
   ulong dl_addr_length;
   ulong dl_mac_type;
   ulong dl_reserved;
   ulong dl_current_state;
   long dl_sap_length;
   ulong dl_service_mode;
   ulong dl_qos_length;
   ulong dl_qos_offset;
   ulong dl_qos_range_length;
   ulong dl_qos_range_offset;
   ulong dl_provider_style;
   ulong dl_addr_offset;
   ulong dl_version;
   ulong dl_brdcst_addr_length;
   ulong dl_brdcst_addr_offset;
   ulong dl_growth;
} dl_info_ack_t;
```

Description

The **DL_INFO_ACK** primitive returns information about the DLPI stream to the data link service (DLS). The **DL_INFO_ACK** primitive is a response to the **DL_INFO_REQ** primitive.

Parameters

dl_primitive
Specifies the **DL_INFO_ACK** primitive.

dl_max_sdu
Specifies the maximum number of bytes that can be transmitted in a data link service data unit (DLSDU). This value must be a positive integer greater than or equal to the value of the *dl_min_sdu* parameter.

dl_min_sdu
Specifies the minimum number of bytes that can be transmitted in a DLSDU. The minimum value is 1.

dl_addr_length
Specifies the length, in bytes, of the provider's data link service access point (DLSAP) address. For hierarchical subsequent binds, the length returned is the total length. The total length is the sum of the values for the physical address, service access point (SAP), and subsequent address length.

dl_mac_type
Specifies the type of medium supported by this DLPI stream. This parameter can have the following value:

DL_HDLC
Indicates the medium is a bit synchronous communication line.

dl_reserved
Indicates a reserved field whose value must be set to 0.

dl_current_state
Specifies the state of the DLPI interface for the stream when the DLS provider issued this acknowledgment.

dl_sap_length
Indicates the current length of the SAP component of the DLSAP address. The specified value must be an integer. The absolute value of the *dl_sap_length* parameter provides the length of the SAP component within the DLSAP address. The value can be one of the following:

>0 Indicates the SAP component precedes the **PHYSICAL** component within the DLSAP address.

<0 Indicates the **PHYSICAL** component precedes the SAP component within the DLSAP address.

0 Indicates that no SAP has been bound.

dl_service_mode
If this **DL_INFO_ACK** primitive is returned before the **DL_BIND_REQ** primitive is processed, specifies the service mode the DLS provider can support. This parameter contains a bit-mask specifying the following value:

DL_CODLS
Indicates connection-oriented data link service.

Once this service mode has been bound to the stream, this parameter returns that specific service mode.

dl_qos_length
Specifies the length, in bytes, of the negotiated and selected values of the quality of service (QOS) parameters. The returned values are those agreed upon during negotiation.

If QOS has not yet been negotiated, default values are returned. These values correspond to those that are applied by the DLS provider on a connect request. For any parameter the DLS provider does not support or cannot determine, the corresponding entry will be set to **DL_UNKNOWN**. If the DLS provider does not support any QOS parameters, this length field is set to 0.

dl_qos_offset
Specifies the offset from the beginning of the **M_PCPROTO** block to where the current QOS parameters begin.

dl_qos_range_length
Specifies the length, in bytes, of the available range of QOS parameter values supported by the DLS provider. This range is available to the calling DLS user in a connect request.

For any parameter the DLS provider does not support or cannot determine, the corresponding entry is set to **DL_UNKNOWN**. If the DLS provider does not support any QOS parameters, this length field is set to 0.

dl_qos_range_offset
Specifies the offset from the beginning of the **M_PCPROTO** block to where the available range of QOS parameters begin.

dl_provider_style
Specifies the style of DLS provider associated with the DLPI stream. This parameter has the following value:

DL_STYLE1
Indicates the PPA is implicitly attached to the DLPI stream by opening the appropriate major or minor device number.

dl_addr_offset
Specifies the offset of the address that is bound to the associated stream. If the DLS user issues a **DL_INFO_REQ** primitive before binding a DLSAP, the value of the *dl_addr_length* parameter is set to 0.

dl_version
Indicates the version of the supported DLPI.

dl_brdcst_addr_length
Indicates the length of the physical broadcast address.

dl_brdcst_addr_offset
Indicates the offset of the physical broadcast address from the beginning of the **PCPROTO** block.

dl_growth
Specifies a growth field for future use. The value of this parameter is 0.

States

Valid The message is valid in any state in response to a **DL_INFO_REQ** primitive.
New The resulting state is unchanged.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_OK_ACK Primitive for X.25

Purpose

Acknowledges that a previously issued primitive was received.

Structure

This primitive consists of one **M_PCPROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_correct_primitive;
} dl_ok_ack_t;
```

Description

The **DL_OK_ACK** primitive acknowledges to the DLS user that a previously issued primitive was received. It is only initiated for the primitives listed in the ″States″ section.

Parameters

dl_primitive	Specifies the **DL_OK_ACK** primitive.
dl_correct_primitive	Identifies the received primitive that is being acknowledged.

States

Valid The message is valid in response to the following primitives:

- **DL_CONNECT_RES**
- **DL_DETACH_REQ**
- **DL_DISCON_REQ**
- **DL_RESET_RES**
- **DL_UNBIND_REQ**

For more information, see Appendix B.

New The resulting state depends on the current state.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_RESET_CON Primitive for X.25

Purpose

Informs the reset-initiating data link service (DLS) user that the reset has completed.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
} dl_reset_con_t;
```

Description

The **DL_RESET_CON** primitive informs the DLS user initiating the reset that the reset has completed.

Parameters

dl_primitive Specifies the **DL_RESET_CON** primitive.

States

Valid The message is valid in the **DL_USER_RESET_PENDING** state.
New The resulting state is **DL_DATAXFER**.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_RESET_IND Primitive for X.25

Purpose

Indicates a data link service (DLS) connection has been reset.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
   ulong dl_originator;
   ulong dl_reason;
} dl_reset_ind_t;
```

Description

The **DL_RESET_IND** primitive informs the DLS user that either the remote DLS user is resynchronizing the data link connection, or the DLS provider is reporting loss of data from which it cannot recover. The primitive indicates the reason for the reset.

Parameters

dl_primitive Specifies the **DL_RESET_IND** primitive.
dl_originator Specifies the originator of the reset. Possible values are:

DL_USER
Indicates the DLS user requested the reset.

DL_PROVIDER
Indicates the DLS provider requested the reset.

dl_reason Indicates one of the following reasons for the reset:

DL_RESET_FLOW_CONTROL
Indicates flow control congestion.

DL_RESET_LINK_ERROR
Indicates a data link error situation.

DL_RESET_RESYNCH
Indicates a request for resynchronization of a data link connection.

States

Valid The message is valid in the **DL_DATAXFER** state.
New The resulting state is **DL_PROV_RESET_PENDING**.

Acknowledgments

The DLS user should issue a **DL_RESET_RES** primitive to continue the resynchronization procedure.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_RESET_REQ Primitive for X.25

Purpose

Requests that the data link service (DLS) provider begin resynchronizing a data link connection.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
} dl_reset_req_t;
```

Description

The **DL_RESET_REQ** primitive requests that the DLS provider begin resynchronizing a data link connection.

Attention: Data in transit when the **DL_RESET_REQ** primitive is initiated may not be delivered.

Parameters

dl_primitive Specifies the **DL_RESET_REQ** primitive.

States

Valid The message is valid in the **DL_DATAXFER** state.
New The resulting state is **DL_USER_RESET_PENDING**.

Acknowledgments

Successful	There is no immediate response to the reset request. However, as resynchronization completes, the **DL_RESET_CON** primitive is sent to the initiating DLS user, resulting in the **DL_DATAXFER** state.
Unsuccessful	The **DL_ERROR_ACK** primitive is returned, and the resulting state is unchanged.

Error Codes

DL_OUTSTATE	Indicates the primitive was issued from an invalid state.
DL_SYSERR	Indicates a system error. The system error is specified in the **DL_ERROR_ACK** primitive.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_RESET_RES Primitive for X.25

Purpose

Directs the data link service (DLS) provider to complete resynchronizing of the data link connection.

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
} dl_reset_res_t;
```

Description

The **DL_RESET_RES** primitive directs the DLS provider to complete resynchronizing of the data link connection.

Parameters

dl_primitive	Specifies the **DL_RESET_RES** primitive.

States

Valid	The message is valid in the **DL_PROV_RESET_PENDING** state.
New	The resulting state is **DL_RESET_RES_PENDING**.

Acknowledgments

Successful	The **DL_OK_ACK** primitive is sent to the DLS user resulting in the **DL_DATAXFER** state.
Unsuccessful	The **DL_ERROR_ACK** primitive is returned and the resulting state is unchanged.

Error Codes

DL_OUTSTATE	Indicates the primitive was issued from an invalid state.
DL_SYSERR	Indicates a system error. The system error is specified in the **DL_ERROR_ACK** primitive.

Implementation Specifics

This primitive is part of X.25 Licensed Program.

DL_UNBIND_REQ Primitive for X.25

Purpose

Requests the data link service (DLS) provider to unbind a data link service access point (DLSAP).

Structure

This primitive consists of one **M_PROTO** message block with the following structure:

```
typedef struct {
   ulong dl_primitive;
} dl_unbind_req_t;
```

Description

The **DL_UNBIND_REQ** primitive requests that the DLS provider unbind the DLSAP that had been bound by a previous **DL_BIND_REQ** primitive. If one or more DLSAPs were bound to the stream with a **DL_SUBS_BIND_REQ** primitive and have not been unbound with a **DL_SUBS_UNBIND_REQ** primitive, the **DL_UNBIND_REQ** primitive unbinds all the subsequent DLSAPs for that stream along with the DLSAP bound with the previous **DL_BIND_REQ** primitive.

At the successful completion of the request, the DLS user can issue a new **DL_BIND_REQ** primitive for a potentially new DLSAP.

Parameters

dl_primitive	Specifies the **DL_UNBIND_REQ** primitive.

States

Valid	The message is valid in the **DL_IDLE** state.
New	The resulting state is **DL_UNBIND_PENDING**.

Acknowledgments

Successful	The **DL_OK_ACK** primitive is sent to the DLS user resulting in the **DL_UNBOUND** state.
Unsuccessful	The **DL_ERROR_ACK** primitive is returned and the resulting state is unchanged.

Error Codes

DL_OUTSTATE	Indicates the primitive was issued from an invalid state.
DL_SYSERR	Indicates a system error. The error is specified in the **DL_ERROR_ACK** primitive.

Implementation Specifics

The **DL_UNBIND_REQ** primitive is part of X.25 Licensed Program.

Chapter 7. X.25 and SNA Networks

You can use an X.25 connection to connect an AIX machine to a System Network Architecture (SNA) network and access a remote X.25 machine, such as a System 390 or eServer zSeries host.

The information included in this chapter is for Communications Server Version 6 for AIX. The information is intended for system administrators who are responsible for:

- Installing Communications Server for AIX
- Configuring the system to match the network to which it is connected
- Keeping the system running

SNA administrators should be familiar with a node and operating procedures for the operating system. Also, they should know the network to which the system is being connected and the general concepts of SNA. In addition, individuals assuming administrative responsibilities should know how to use the System Management Interface Tool (SMIT). Some of the tasks related to system administration are:

- Customizing Communications Server for AIX
- Starting and stopping Communications Server for AIX
- Defining network security
- Obtaining network status

Communications Server for AIX provides menu dialogs, commands, and procedures for system administration purposes.

The following sections contain detailed information about using an X.25 connection to connect a host machine to an SNA network and access a remote X.25 machine, such as a System 390 or eServer zSeries host.

Accessing an SNA Network with X.25

The X.25 protocol allows you to access an SNA network. You can make the connection by using the Communications Server for the operating system licensed program product.

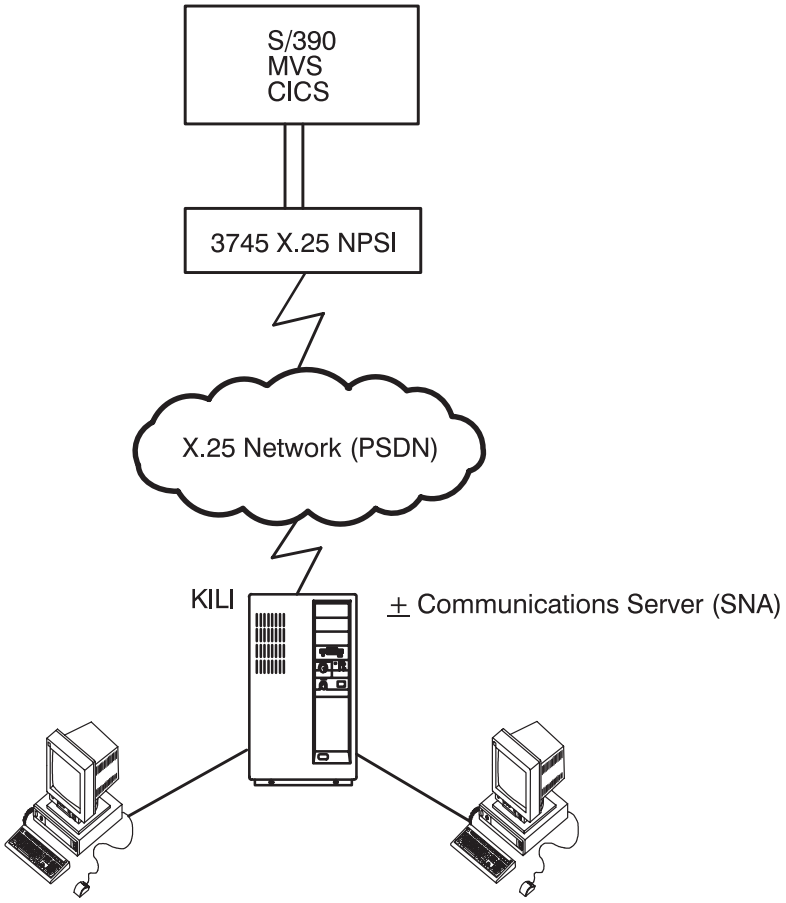


Figure 11. Example of a SNA Connection Using X.25. An X.25 network (PSDN) being used to provide communication between (a) an RS/6000 or eServer pSeries system running SNA services with X.25 and (b) a System 390 or eServer zSeries host.

QLLC with Reference to X.25 Support

Qualified Logical Link Control (QLLC) is only for SNA support. It permits the use of additional link control information that SNA needs, but which X.25 does not need. QLLC architecture specifies the mapping between Synchronous Data Link Control (SDLC) frames and X.25 packets.

When SNA is used over X.25, it uses the qualifier-bit (Q-bit) in the X.25 packet header to indicate special link control information. This information is relevant for SNA control between the two systems communicating with each other, but it is of no concern to X.25 link control. These qualified packets help SNA to determine who is calling whom between the two communicating systems and indicate such items as maximum message size.

QLLC must be used in the following situations:

- When two systems are communicating with each other by means of SNA over X.25
- When a system is communicating with a host, for example, a System 390 or eServer zSeries, by means of SNA over X.25

Some systems can control the segmenting and unsegmenting of messages themselves, rather than leaving it to the X.25 protocol . These systems use additional control to perform this. The RS/6000 does

not support this additional control of Logical Link Control (LLC) headers, embedded in X.25 packets, to segment messages. The system leaves it to the X.25 protocol to handle segmenting.

> **Note:** The X.25 ports that are to be used for SNA must have COMIO emulation configured on them. This produces ports such as `x25s0` which are used by QLLC. For example, the X.25 port `sx25a0` cannot be used directly by SNA or QLLC.

Introducing Communications Server Version 6

Communications Server for AIX implements two SNA components that control the operation of the local node in the network. These components are as follows:

- Physical Unit (PU) Communications Server for AIX provides the capabilities of a PU Type 2.1 node
- Logical Unit (LU) classified as one of the following types:

LU0 Program-to-program. This logical unit manages communication between devices associated with the retail industry (for example, point of sale terminals). It provides primary and secondary LU support through an application programming interface (API).

LU1 Host program to RJE workstation. This protocol manages many input/output devices such as printers, card readers and punches, storage devices, and an operator console.

LU2 Host program to 3270 display. Used by applications related to the emulation of an IBM 3270 terminal, allowing keyboard input, display output, and file transfer, using the SNA 3270 display data stream.

LU3 Host program to 3270 printer. Used by applications related to the emulation of an IBM 3270-attached printer allowing for printed output, using the SNA 3270 printer data stream.

LU6.2 Advanced Program-to-Program Communications (APPC). For communications between two programs on a peer-to-peer basis.

SNA Configuration Definitions

Communications Server for AIX Version 6 (CS/AIX) keeps all network definitions in the following flat files in the **/etc/sna** directory:

sna_node.cfg
node, dlc, port, linkstations and LUs

sna_domn.cfg
CPI-C Side Information definitions

sna_tps
local TP definitions

Much of the information contained in the CS/AIX configuration files is independent of the link type being used. Only the [define_qllc_dlc], [define_qllc_port], and [define_qllc_ls] stanzas in the **sna_node.cfg** file have information that is specific to X.25. There are rules about the order in which CS/AIX definitions can be created, and some definitions that are required before the X.25 specific definitions can be created. This hierarchy of definitions is as follows:

[define_node]
The node definition identifies the local node's function in the network as well as uniquely identifying itself on the network (through the CP name). The control point is the default INDEPENDENT LU 6.2 LU for the system. The XID node ID value identifies this node when a link station profile specifies to use the control point's XID instead of supplying one of its own

[define_qllc_dlc]
Defines a dlc_name (label) and the CITT support level (1980, 1984, 1988). This entity corresponds to the AIX QLLC DLC provided by the **bos.qllc.dlc** fileset.

[define_qllc_port]
Defines a port_name (label) and points to the associated dlc_name. It also defines the COMIO interface (x25s#), with the port_number parameter and the local X.25 network address (NUA). It also defines default values for parameters like the maximum I-field size, transmit and receive window counts, and various polling frequencies, counts, and secondary inactivity time-out values.

[define_qllc_ls]
Defines an ls_name (label) and points to the associated port_name. It also defines the VC type (switched or permanent) and remote X.25 network address (NUA) for an SVC or the logical channel number (LCN) for a PVC. It also defines the local XID node ID, response to link inactivity, station type, link activation parameters, virtual circuit type, supported session types, restart parameters, and transmission group characteristics.

Other CS/AIX definitions are probably required depending on the specific needs and the type of remote SNA node. These include independent LUs and/or dependent LUs. See the ″IBM Communications Server for AIX Administration Command Reference Version 6″ for details on these definitions.

SNA Configuration Commands

The CS/AIX definitions can be added, changed, and deleted with several different configuration methods, including::

snaadmin
Command line

smit AIX menu system

xsnaadmin
Xwindows GUI

webadmin
HTTP browser GUI

NOF API
C language programming API

You can also edit the CS/AIX configuration files directly, but this is not recommended and requires that the CS/AIX system be stopped with the 'sna stop' command before the editing is done.

To display the syntax of the **snaadmin** command, type:

```
snaadmin -d -h
```

To display details on the parameters for each **snaadmin** subcommand, for example, type:

```
snaadmin -d -h define_qllc_ls
```

The SMIT menus for CS/AIX can be accessed with the fast path `smit sna` or by the following path: `smit` -> **Communications Applications and Services** -> **Communications Server For AIX**. Only the most common parameters are contained in the SMIT menus. All parameters can be changed with the **snaadmin** command.

The xnaadmin GUI is an XWindows application. If the AIX system has a graphics console and runs an X server (such as CDE), invoke the **xsnaadmin** command. If the AIX system does not have a graphics console, but a remote X server is available, set the **DISPLAY** environment variable to point to that remote X server and then invoke the **xsnaadmin** command. For example:

```
export DISPLAY=remoteaix:0
xsnaadmin
```

In **xsnaadmin**, there is no separate menu for the [define_qllc_dlc] stanza. It is automatically created when the [define_qllc_port] stanza is defined. Only the most common parameters are exposed in the SMIT menus. All parameters can be changed with the **snaadmin** command.

The **webadmin** GUI uses a Web server (such as the IBM HTTP Server) and an applet server (such as WebSphere Application Server) on the AIX machine and any Java capable web browser. After the Web server and applet server have been defined and started, point the Web browser to the following address:

```
http://aixname/SnaAdmin/
```

where *aixname* is the TCP/IP name or address of the CS/AIX system.

The NOF API is a C language programming API for the Node Operator Facility. The API has calls for most of the commands similar to the **snaadmin**.

Consult the ″IBM Communications Server for AIX Administration Guide Version 6″ for more details on these CS/AIX configuration methods.

Communications Server for AIX Definition Types and Parameters

The following table lists some of the definition types related to X.25, some of their corresponding parameters, and a brief description.

Communications Server for AIX DefinitionTypes and Definition Parameters

Definition Type	Definition Parameter	Description
define_node	*node_type*	Type of the node. Possible values are: **LEN_NODE** Low entry **END_NODE** APPN end node **NETWORK_NODE** APPN network node **BRANCH_NETWORK_NODE** APPN branch network node
	fqcp_name	Fully qualified control point (CP) name of the node. The name is a type-A character string, consisting of 1-8 character network name, a period character, and a 1-8 character CP name.
	cp_alias	Locally used LU alias for the control point (CP) LU. This alias can be used by APPC applications to access the CP LU. This alias is a string of 1-8 characters.
	node_id	Node identifier used in XID exchange. This ID is a 4-byte hexadecimal string, consisting of a block number (three hexadecimal digits) and a node number (five hexadecimal digits). Defaults to 0x07100000.
define_qllc_dlc	dlc_name	Name of the DLC. This name is a character string using any locally displayable characters.

Communications Server for AIX DefinitionTypes and Definition Parameters

Definition Type	Definition Parameter	Description
	support_level	level X.25 support level provided by the adapter. Possible values are: **1980** 1980 standard **1984** 1984 standard **1988** 1988 standard
define_qllc_port	port_name	Name of the port to be defined. This name is a character string using any locally displayable characters.
	dlc_name	Name of the associated DLC. The specified DLC must have already been defined.
	port_number	The number of the COMIO interface name (x25s#).
	address	Local X.25 address of the port used for incoming calls. If the local station becomes secondary after Link Station role negotiation, this address is used in the response to an incoming call. If the local station is primary, or if the port is used only for outgoing calls, this parameter is reserved.
define_qllc_ls	ls_name	Name of the link station to be defined.
	port_name	Name of port associated with this link station. This name must match the name of a defined port.
	address	Destination address of the remote link station. This parameter is used only for SVC outgoing calls (defined by the vc_type parameter); it is ignored for incoming calls or for PVC. The address is a string of 1-15 characters. The address is in X.25 (1980) format; later address formats are not supported.
	pvc_id	PVC identifier. Set this parameter to a decimal number to identify which PVC (from the range of PVCs defined for your X.25 provider software) to use for this LS. This parameter is reserved if vc_type is set to SVC.
	vc_type	The virtual circuit type of the LS. Possible values are: **SVC** Switched virtual circuit **PVC** Permanent virtual circuit

Communications Server for AIX DefinitionTypes and Definition Parameters

Definition Type	Definition Parameter	Description
	local_node_id	Node ID sent in XIDs on this LS. This ID is a 4-byte hexadecimal string, consisting of a block number (three hexadecimal digits). To use the node ID specified in define_node, do not specify this parameter.
	loc_packet	Packet size used for sending data on switched virtual circuits from the local station to the remote station. This parameter is used only if the vc_type parameter is set to SVC. The packet size you specify is sent as an optional facility on the outgoing call.
	rem_packet	Packet size used for receiving data on switched virtual circuits from the remote station. This parameter is used only if the vc_type parameter is set to SVC.
	loc_wsize	Window size used for sending data from the local station to the remote station. Specify a value in the range 1-7, or 0 (zero) to indicate using the default window size for the network.
	rem_wsize	Window size used for receiving data from the remote station. Specify a value in the range 1-7, or 0 (zero) to indicate using the default window size for the network.

Customizing Communications Server for AIX

Use the following procedures to customize the Communications Server to use emulation within an X.25 network.

Each numbered step is described in the subsequent sections.

Customizing Communications Server to Use X.25

1. Install the Communications Server for AIX Licensed Program Product.
2. Install the Qualified Logical Link Control (QLLC) device driver.
3. Get host definitions.
4. Create the CS/AIX Definitions:
   - Define the node ([define_node]).
   - Define the X.25 SNA DLC ([define_qllc_dlc])
   - Define the X.25 SNA Port ([define_qllc_port])
   - Define the X.25 SNA Link Station ([define_qllc_ls])
   - Define any local or remote LUs
5. Start the SNA node.
6. Start the SNA link station.
7. Display the SNA status.

Installation of the Communications Server (SNA) Program

The licensed program product consists of the following parts:

sna.msg.*LANG*.rte	Contains the messages and helps in the specified language for the run-time environment
sna.rte	Contains the Communications Server (SNA) base program

The licensed program may also contain update files.

Installation of the Qualified Logical Link Control

Install the **bos.dlc.qllc** fileset from the AIX installation media.

The QLLC is the device driver that is used by Communications Server (SNA) to communicate through the supported X.25 adapters. QLLC has a special file under the **/dev** directory named **dlcqllc**.

To show the predefined device information for QLLC, run the following:

```
lsdev -P -H -c dlc -t x25_qllc
```

The output from that command looks similar to the following:

```
class type       subclass   description
dlc   x25_qllc   dlc        X.25 QLLC Data Link Control
```

You can use one of the following methods to install the X.25 QLLC Data Link Control:

- To use the SMIT fast path, type:

  ```
  smit cmddlc_qllc_mk
  ```

- To use SMIT , type:

  ```
  smit
  ```

  Make the following selections from the SMIT menus:

  Select **Devices**.

  Select **Communication**.

  Select the appropriate adapter:

  - **IBM ARTIC960 Adapter**
  - **Portmaster Adapter/A**
  - **X.25 CoProcessor/2 or Multiport/2 Adapter**
  - **X.25 CoProcessor/1**
  - **IBM 2-Port Mutiprotocol PCI Adapter** (see Note below)
  - **IBM ARTIC960HX PCI Adapter**

    Note: If you select **IBM 2-Port Multiprotocol PCI Adapter**, do the following:

    1. Select **Manage HDLC Device Driver**
    2. Select **Manage Additional Protocols / Emulators**
    3. Select **Manage X.25 over HDLC Device Driver**

  Select **Services**.

  Select **Data Link Controls**.

  Select **Add a QLLC Data Link Control**.

- To install from the command line, type:

```
mkdev -c dlc -s dlc -t x25_qllc
```

Getting Host Definitions

To start configuring Communications Server (SNA), request some information from the Virtual Telecommunication Access Method (VTAM) administrator for the SNA node you want to connect to. The following table is the minimum information required in this example:

SNA Host Definitions		
VTAM Parameter	**Value**	**Profile**
Network User Address	3106001984	Link station profile
IDBLK	071	3270 LU profiles
IDNUM	06000	3270 LU profiles
SSCP ID	20	3270 LU profiles
LOCADDR	2, 3	3270 LU profiles

Network User Address: The X.25 address, *3106001984*, uniquely identifies the X.25 line to which the System 390 or eServer zSeries host is attached. This parameter is used in the ″Link station profile″ definition.

IDBLK The three hexadecimal digits of this parameter provide an identifier, or block number, that is unique to each product on the network. For the RS/6000, this number is **071**.

IDNUM The five hexadecimal digits of this ID number distinguish a specific piece of equipment from all others of a similar kind on the network. The number usually is given by the VTAM administrator. In our case, an ID number of `06000` was assigned to our RS/6000.

SSCPID The ID of the controlling System Services Control Point (SSCP) in the SNA network (decimal value).

LOCCADR Local addresses of the 3270 displays and printers in the 3270 cluster. One decimal value is used for each display or printer connected.

Create the CS/AIX Definitions

To create the CS/AIX definitions needed to support X.25, the examples show how to use SMIT do this. The definitions must be done in the order shown.

Creating the CS/AIX Definitions consists of the following steps:

1. “Define the node”
2. “Define the SNA X.25 DLC” on page 104
3. “Define the SNA X.25 Port” on page 104
4. “Define the SNA X.25 Link Station” on page 105

Define the node

1. Type `smit sna`.
2. Select **Configure SNA Resources**.
3. Select **Local Node Resources**.
4. Select **Node Definition**. The following SMIT panel displays:

```
                                Node Definition

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Control Point alias                                 []
  Description                                         []
```

```
* Control Point name                                []
  APPN Support                                       END_NODE              +
  Node ID                                           [07100000]              X
  NM-API Style                                       NORMAL                +
  If BACK_LEVEL,
        Queue NMVTs?                                 YES                   +
```

Define the SNA X.25 DLC

1. Type `smit sna`.
2. Select **Configure SNA Resources**.
3. Select **Local Node Resources**.
4. Select **Connectivity**.
5. Select **DLCs, Ports and Link Stations**.
6. Select **Add Connectivity Resource**.
7. Select **Add X.25 Resource**.
8. Select **Add X.25 DLC**. The following SMIT panel displays:

```
                                Add X.25 DLC

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                   [Entry Fields]
* DLC name                                        []
  Description                                     []
  Negotiable link stations supported?              YES                   +
  Initially active?                                YES                   +
  Adapter Number                                  [0]                     #
  X.25 level                                       1980                  +
```

Define the SNA X.25 Port

1. Type **smit sna**.
2. Select **Configure SNA Resources**.
3. Select **Local Node Resources**.
4. Select **Connectivity**.
5. Select **DLCs, Ports and Link Stations**.
6. Select **Add Connectivity Resource**.
7. Select **Add X.25 Resource**.
8. Select **Add X.25 Port**. The following SMIT panel displays:

```
                                Add X.25 Port

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                   [Entry Fields]
* Port name                                       []
  Description                                     []
* DLC Name                                        []                    +
* Local X.25 sub-address                          []
  Initially active?                                YES                   +
  Use HPR on implicit links?                       YES                   +
  Maximum BTU size allowed                        [265]                   #
  Maximum number of active links allowed          [255]                   #
  Implicit DSPU template                          []                    +
  Implicit links to end nodes are uplinks?         NO                    +
```

Define the SNA X.25 Link Station

1. Type **smit sna**.
2. Select **Configure SNA Resources**.
3. Select **Local Node Resources**.
4. Select **Connectivity**.
5. Select **DLCs, Ports and Link Stations**.
6. Select **Add Connectivity Resource**.
7. Select **Add X.25 Resource**.
8. Select **Add X.25 Link Station**. The following SMIT panel displays:

```
                              Add X.25 Link Station

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                              [Entry Fields]
* Link station name                               []
  Description                                     []
* Port name                                       []                        +
  Activation                                       BY_ADMINISTRATOR         +
  Independent LU Traffic:
  Remote node Control Point name                  []
  Remote node type                                 LEARN_NODE               +
  Dependent LU Traffic:
  Downsteam PU services supplied                   NONE                     +
  If SNA Gateway or DLUR,
        Downstream PU name                        []
  If DLUR,
        DLUS server name                          []
  Local node id                                   []                         X
  Remote node id                                  []                         X
  Virtual circuit type                             SVC                      +
  Contact Information:
  If switched virtual circuit
        Remote X.25 address                       []
  If permanent virtual circuit,
        Channel number                            [1]                        #
  Advanced:
  Maximum BTU size to be sent                     [265]                      #
  Host type                                        SNA                      +
  Request CP-CP sessions?                          YES                      +
  Remote node is a network node server?            NO                       +
  Use HPR?                                         NO                       +
  TG number                                       [0]                        #
  Solicit SSCP sessions?                           YES                      +
  Solicit SSCP sessions?                           YES                      +
  Link station role                                USE_PORT_DEFAULTS        +
  Branch link type                                 NONE                     +
  Reactivate link station after failure            YES                      +
  Window size                                     [7]                        #
  Frame retransmit limit                          [2]                        #
  Frame retransmit timer                          [8]                        #
  Packet size                                      DEFAULT                  +
  Call User Data                                  []                         X
[BOTTOM]
```

Additional CS/AIX definitions may be required, depending on the specific environment. Such definitions include Local LU6.2 LUs, Partner LU6.2 LUs, Local Dependent LUs. See the ″IBM Communications Server for AIX Administration Guide Version 6″ for more details.

Defining the LU Profiles for 3270 Communication

Start SNA Node

Before any application can use Communications Server, the SNA node must be active.

To start the SNA node, log in as the root user and type the following command:

```
snaadmin init_node
```

Start SNA Link Station

A link station can be defined to start automatically when an application requires it, or it can be started manually. Starting the link station begins the physical connection with the remote machine, so a call request packet is sent from the X.25 adapter.

To start a link station, log in as the root user and type the following:

```
snaadmin start_ls, ls_name=X
```

where *X* is the link station name.

Starting the link station automatically starts the associated SNA DLC and SNA Port if they are not already active.

Display Current Status of SNA Node

To obtain status information about CS/AIX, log in as the root user and enter the following commands:

```
snaadmin status_node
snaadmin status_all
snaadmin query_ls
```

Chapter 8. Packet Assembler/Disassembler (PAD)

X.25 defines the relationship between a Date Terminal Equipment (DTE) node and a network's DCE. The CCITT also defines the attachment of asynchronous start-stop devices to the PSDN in recommendations X.3, X.28 and X.29. The asynchronous device is typically a low-cost terminal such as an IBM 3151 or 3152. These terminals lack the intelligence of a full-function node on the network and rely on a device called a packet assembler/disassembler (PAD). A PAD is a protocol converter interfacing asynchronous terminals with an X.25 network or an X.25 network with applications written for asynchronous terminals.

Low-cost asynchronous terminals are connected by the public-switched telephone network to a local public outgoing PAD, often simply referred to as a PAD. The PAD takes the ASCII data stream coming from the terminal, buffers it, converts it into a properly formatted X.25 packet, then sends it over the X.25 PSDN addressed to the desired DTE. Data from the terminal that has been divided into packets and shipped by a PAD to the DTE node must be received by some incoming PAD software (also referred to as an X.29 interface). This incoming PAD unpacks the data and passes it to the application (typically, the TTY driver) for processing as if it were coming from directly attached asynchronous terminals. The other way, data coming from the application is first put into packets by the X.29 PAD then transferred over the network. When the PAD receives a packet addressed to the terminal, it reverses the process. This presents the data in a form that the terminal can accept.

The computer industry has expanded the definition of a packet assembler/disassembler (PAD) to include protocol conversion between X.25 and various teleprocessing protocols, for example, SDLC PADs, BSC-3270 PADs, and so on.

Basic PAD functions include:

- Assembly of characters into packets (asynchronous-to-X.25).
- Disassembly of the packet's user data field (X.25-to-asynchronous).
- Handling of virtual call setup and clearing.
- Handling of virtual call reset and interrupt procedures.
- Generation of service signals.
- Mechanism for forwarding packets when the proper conditions exist.
- Mechanism for transmitting data characters (including start, stop, and parity elements) as appropriate to the start-stop terminal.
- Mechanism for handling a break signal from the start-stop terminal.
- Editing of PAD command signals.
- Mechanism for setting and reading the current value of PAD parameters.

Optional functions include:

- Mechanism for selection of a standard profile.
- Automatic detection of data rate, code, parity, and optional characteristics.
- Mechanism for a remote DTE to request a virtual call between the start-stop mode DTE and another DTE.

X.3, X.28 and X.29 Standards

X.3 Defines a set of parameters that the PAD uses to identify and control the attached terminals. A complete set of parameters is called a *profile*, and each DTE-C (Data Terminal Equipment-Character) has its own profile selected or set for use with the PAD.

X.28 Defines the control procedures used to establish the physical connection to the PAD, the commands the user sends to the PAD, and the service signals sent by the PAD to the terminal user. Simply, X.28 defines how a terminal user can control the X.3 PAD parameters. It specifies the command and response formats, and the status indicators.

X.29 Defines the way in which a PAD and a remote DTE (or another PAD) exchange control messages on an X.25 virtual circuit. The messages are formatted as special X.25 packets called *qualified data packets* (Q-bit). For example, the packet mode DTE may use an X.29 message to change one of the internal X.3 parameters in the PAD. X.29 messages will only be effective when an X.25 call is established. The user at the terminal is not explicitly aware of the X.29 communication between the PAD and the remote DTE. The user at the terminal logs into the host DTE in the same way as the user of an ASCII terminal locally attached to the same host. The following diagram illustrates the interaction between X3, X28, and X29.

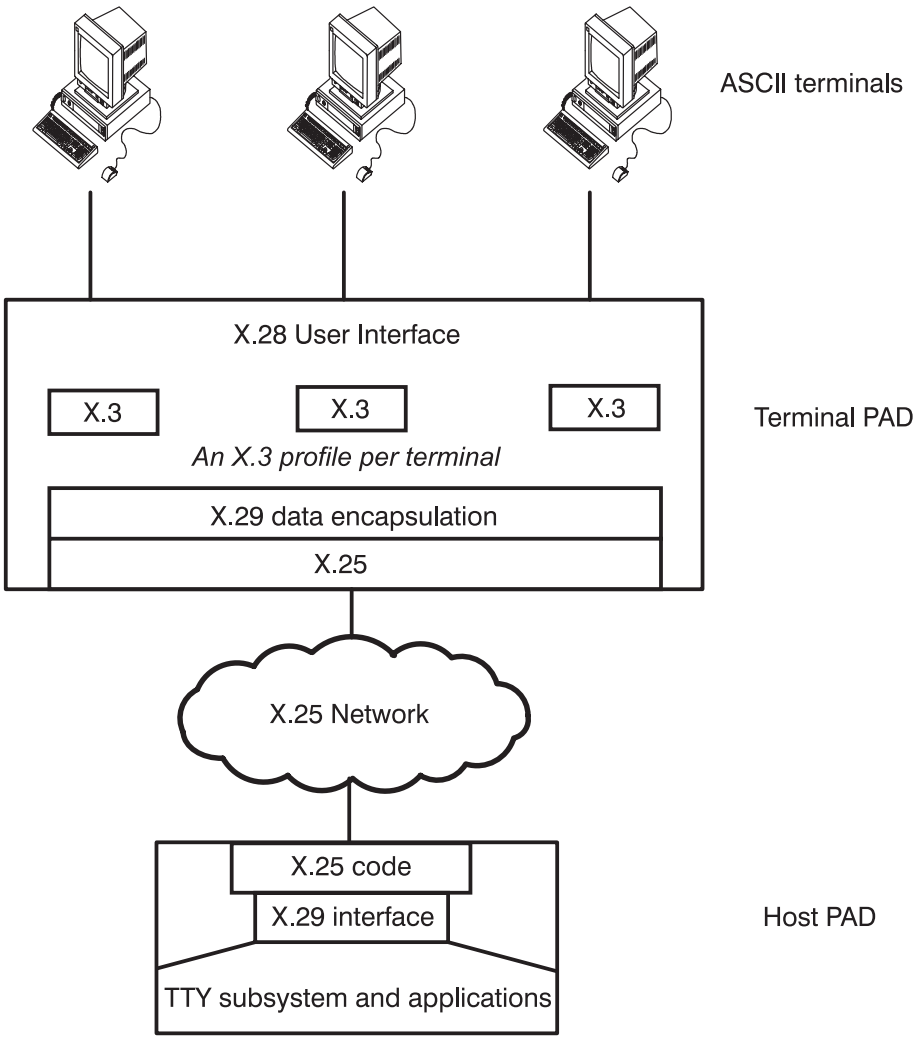


Figure 12. How X.3, X.28, and X.29 Interact

Public PAD Provided by the Post Telephone and Telegraph (PTT) administration or network provider that you can access by a local call on the public switched network. Once your terminal is connected to the public PAD, you enter the NUA of your host DTE and the PAD establishes the communications.

Private PAD	A hardware device with several asynchronous connectors and an X.25 attachment. This commercially available device can be connected to a public or private X.25 PSDN and has the same capabilities as a public PAD.
Integrated PAD Software	A software product emulating a PAD and requires a computer supporting an X.25 connection. It generally provides an asynchronous terminal emulator, but can use existing ones such as ATE or CU. With this product, the **xspad** function performs the asynchronous terminal-emulator role.

PAD Setup

To use the PAD software on the system, configure it through SMIT. See Managing the Triple-X PAD for more details. Once the PAD software is enabled on the system, it can be used as a terminal PAD for supporting local terminal users, or as a host PAD to work with remote terminal PADs. Both modes of operation can be run at the same time.

For use as a terminal PAD, each terminal or terminal emulator that requires a PAD session executes the **xspad** command. The **xspad** command connects the terminal to an application that provides the standard X.28 interface. See Using the PAD for more details on the use of terminal PAD sessions.

When the system is to be used as a host PAD, no user interaction takes place on the system itself. Once the system is configured to enable the PAD, users can connect from terminals connected to remote terminal PADs. The system running the host PAD allows login and application execution similar to any other type of remote ASCII terminal.

> **Note:** PADs support ASCII terminals using 7-bit character set defined in International Alphabet 5 (IA5). PAD supports 8-bit data. The characteristics of the terminal must agree with its configuration data on the system unit it is attached to if using **xspad** on an ASCII terminal. This is also true for any attached TTY.

Using the PAD

This section discusses starting an X.28/X.3 terminal PAD session. For information on how to start X.29 on the system, refer to Managing the Triple-X PAD.

To start a terminal PAD session, run the **xspad** command. On systems where the PAD is configured this starts a terminal PAD session and provides the PAD prompt (*, the asterisk character). Enter the PAD commands at the PAD prompt.

To start a PAD session:

1. Ensure the PAD is configured on the system.
2. Run **xspad -l sx25a**# where # is the port number.
3. Issue the call to the remote X.25 host at the PAD prompt.
4. Log on to the X.25 host and run the desired application.

For more information, see the **xspad** Command in Appendix A.

Commands

Once connected to an X.3/X.28 terminal PAD session, various commands can be issued to the PAD. Some of these commands can only be issued after a connection has been established with the remote X.25 host. Based on the PAD's profile, the commands understood by the PAD are either based on the CCITT standard or can be in a less cryptic style (advanced mode).

The start/stop DTE user may interact with the PAD using PAD commands. These commands provide:

- Establishment and clearing of virtual calls.
- Selection of PAD profiles.
- Display of the PAD parameter values.
- Changing the PAD parameter values.
- Sending an interrupt or a reset packet.

To enter a PAD command at the terminal, the user enters the command at the PAD prompt. As long as there is not a remote host connection, the current prompt is the PAD prompt (*, the asterisk character). To get to the prompt when connected to a remote session, press Ctrl-P.

At the prompt, the following commands can be issued:

break Sends a break character to the PAD.

call Establishes a connection to the remote X.25 host: `call 34511884`, where `34511884` is the remote NUA.

clear Clears the connection with the remote X.25 host. The clear is sent from the PAD immediately. See the **iclear** command.

help Requests help text. See Getting Help for more details.

iclear Sends an ″invitation to clear″ to the remote X.25 DTE. This allows the remote host to send any pending data before clearing the call.

interrupt Sends an interrupt packet. The packet contents cannot be user-specified.

language Sets the language for the PAD help text to English, French, or Spanish. For example, `language french.`

nui This command is not implemented in the PAD.

par Reads the PAD parameters.

par? Displays all parameters.

par? 2, 3 Displays parameters 2 and 3.

In advanced mode, **read** or **parameter** can be used instead of **par**.

profile Displays which profiles are available, or allows you to change the profile. Enter `PROFILE` followed by the profile name, such as `PROFILE PROFILE_51` to make changes.

read See **par**.

rpar/rread Reads the PAD parameters as **par?** does for the local PAD in implementations where a remote PAD DTE is supported.

rset/resetread Writes the PAD parameters as **set?** does for the local PAD in implementations where a remote PAD DTE is supported.

set Sets one or more of the PAD parameters. For example:

```
set 2:1, 14:2
or in advanced mode
set echo:1, lfpad:2
```

> **Note:** The default profile does not allow the parameters to be changed locally as they are being controlled remotely through the TTY subsystem and the **stty** command. **xspad** does not require the use of the **set** command or any of the remote PAD commands (**rread**, **rpar**, **rset**, and **rsetread**).

status Indicates whether a connection to the remote X.25 host is active. Returns `ENGAGED` if it is active or `FREE` if no connection is established.

Establishing Calls

To establish a connection with a remote X.25 host, a call must be made to it. The call must be given the remote DTE's address and any call user data or facilities being requested.

X.28 Facility Codes		
Facility Code	**Function Requested**	**Value** (text that follows the facility code)
C	Charging Information	

E	Called Address Extension	Extended address
F	Fast select with no restriction	
G	Closed User Group	CUG number
N	Network User Identification	NUI string
Q	Fast select with restrictions on response	
R	Reverse charging	
D	Call User Data	ASCII CUD string to be added to the standard PAD call CUD. (Must be the last part of the command string.)
-	Facility end marker	End of facilities marker (allow CUD).

Example Calls

The following are typical calls that might be made from the PAD command prompt:

`call 3536647`	Establish a call to NUA 3536647
`call 3536647 F,R-`	Request fast select and reverse charge
`call 3536647 R- D user1`	Have ASCII based CUD ″user1″
`call 3536647,9093388`	Make the call with the local (calling) address of 9093388 included

Connected Sessions

After a call has been placed to the remote X.25 host, it can be rejected by the network, or the remote host, or accepted by the remote host. If rejected, a clear packet is received and results in a message similar to those described in Ending Calls . Once connected, the terminal is under the control of the TTY system of the remote host. Typically a login screen is presented. After the user has logged in, it can be used the same as other attached TTYs.

Once a session is established, the **TERM** type and **stty** settings should be verified. The **TERM** type should be set to the type of terminal you are using, for example, `TERM=aixterm`. The **stty** settings can be verified with `stty -a`. The row and column attributes need to be updated if using X-Windows and the size of the window is not equal to 80x25 (default setting). The **echo**, **echoe**, and **echok** attributes might also need to be turned `On/Off`.

To change an **stty** attribute, enter one of the following commands:

`stty echo (turns echo` **`on`**`)`

`stty -echo (turns echo` **`off`**`)`

`stty 50 rows (updates the row setting to 50)`

Ending Calls

Once a connection is established, there are a number of ways in which it can be terminated. Termination of the login shell used to log in to the remote X.25 host closes the connection and results in a message such as the following:

```
CLEAR DTE 0 241 - Call cleared, by remote device, data may be lost
```

As the session was terminated above the PAD layer, the PAD code did not expect it and so could not tell if it was user-initiated or not. If it was user-initiated, then no data was lost.

An alternative is to press Ctrl-P to get to the PAD command prompt, and then enter **iclear**. The **iclear** command causes the local terminal PAD to send a request to the host PAD to clear the connection. The host PAD software then issues the clear, but without regard for any applications that might be running on the remote host under this login. As the remote PAD software issues the command to clear the connection, the diagnostic software is likely to reflect this as being an expected clear.

```
CLEAR PAD 0 0 - Call cleared, remote request
```

The additional text is given when advanced mode is enabled through the profile; otherwise the base clearing reason is given.

Exiting xspad

From the PAD command prompt that is reached by ending the call as described above, enter ^k (Ctrl-K) to terminate the **xspad** program. The **xspad** application must not have a call established when attempting to terminate **xspad**.

Automatic Termination and Identification for AIXlink/X.25 1.1.3 (and later)

The PAD application is automatically exited upon termination of the X.25 Host connection. This optional feature is enabled using the **-x** flag with the **xspad** command (for example, `xspad -x -l sx25a0`).

The PAD default is to enter ^k (Ctrl-K) before exiting. If an X.25 Host connection terminates and the user does not exit, the user can initiate another X.25 connection using the same X.28 STREAMs connection.

> **Note:** This feature applies only to **xspad** (the user space PAD application).

Getting Help

Help is provided through the **help** command. To list the help topics, enter `help list`. Each of the PAD commands has help text associated with it (for example, `help call`). Also each PAD parameter and profile has help text. The language in which the text is displayed can be changed using the **language** command.

Clearing Codes

When the call is cleared, the information passed in the X.25 clear packet is displayed. When advanced mode has been selected through use of the appropriate profile, a more detailed text explanation of the clear is given. Otherwise, the base cause is displayed as follows:

OCC	Remote DTE busy
NC	Network congestion
INV	Invalid facility
NA	Access barred
ERR	Local procedure error
RPE	Remote procedure error
NP	Number not assigned
DER	DTE out of order
PAD	DTE clearing
DTE	DTE device clearing
RNA	Reverse charging rejected
ID	Incompatible destination
SA	Ship cannot be contacted
FNA	Fast select rejected
ROO	Cannot route as requested

PAD Parameters

The operation of the PAD depends on the value of a set of internal variables called **PAD parameters**. An independent set of parameters exists for each start/stop mode DTE. The current value of each PAD parameter defines the operational characteristic of the related function.

Initially, the values of the parameters are chosen by selecting one of the available profiles. Subsequent changes for a given terminal session are done under control of the session's TTY subsystem on the host PAD. When the application being run requires a change to the terminal characteristics, it modifies the session's **stty** structure which in turn causes the X.29 protocol to issue a change to the session's X.3 parameters held in the terminal PAD.

The following is a list of the X.3 PAD parameters:

par 1
PAD Recall

This parameter specifies the character that you can type at the terminal to interrupt data flow with the X.25 host (in order to enter commands addressed to the PAD). Possible values are:

0 Data flow can not be interrupted. The user can not enter PAD commands.

1 The escape character sequence is Ctrl-P.

32-126 Value of the PAD recall character in decimal.

Note: If you set this parameter to **0**, you will not be able to change the PAD parameters any further.

par 2
echo

This parameter specifies whether the characters received from the terminal are to be transmitted back to the terminal as well as being interpreted by the PAD. Possible values are:

0 No echo by the PAD.

1 Local echo by the PAD.

Note: When working in line mode, this parameter should be set to **1** to get an echo from the PAD, and the echo provided by the host should be disabled with the command `stty -echo`.

par 3
forward

This parameter specifies the conditions under which a string of input characters from a terminal is converted to a data packet and forwarded to the X.25 Network. Possible values are:

0 Data forwarding not controlled by a character.

1 Data forwarding on alphanumeric characters.

2 Data forwarding on receipt of carriage return (ENTER).

4 Data forwarding on receipt of ESC, BEL, ENQ, or ACK.

8 Data forwarding on receipt of DEL, CAN, or DC2.

16 Data forwarding on receipt of EXT or EOT.

32 Data forwarding on receipt of HT, LF, VT, or FF.

64 Data forwarding on receipt of any character from column 0 and 1 of the ASCII code page.

If this parameter is set to **0**, every character typed at the terminal is sent by the PAD in an individual packet. For the other values, the PAD acts as a buffer and sends a packet only when the specified character is entered. If you are working on your terminal in line mode, entering UNIX commands for example, you may set this parameter to 18 and packets will be sent to the host only when you press Enter, Ctrl-C, or Ctrl-D.

par 4

idle This parameter specifies a timeout period for the reception of characters from the terminal. After this timeout expires, characters already received by the PAD are formatted into a data packet and sent to the X.25 network. Possible values are:

0 No timeout period is used.

1-255 Timeout period expressed in units of 0.05 seconds.

This parameter, when set to a non-zero value, improves the data transfer rate from the terminal to the host. In this case, the time delay between the PAD receiving two consecutive characters is lower than the timeout period, so the PAD accumulates and sends data only when it has a full packet, or when the file transfer application has stopped sending characters. When using line mode, the value of this parameter should be **0**.

par 5 Controls the ancillary device. Possible values are:

0 No use of X-ON and X-OFF.

1 Use of X-ON and X-OFF during data transfer.

2 Use of X-ON and X-OFF during commands and data transfer.

par 6 Signals Controls the service signals and dialog mode for the terminal. Possible values are:

0 No service signals are sent to the terminal.

1 Service signals other than *prompt PAD service* are transmitted in standard format.

4 Service signal *prompt PAD service* is transmitted in standard format.

8-15 PAD service signals are transmitted in a network-dependent format.

16 English extended dialog mode.

32 French extended dialog mode.

48 Spanish extended dialog mode.

par 7 Break Specifies the action of the PAD on receipt of a break signal from the terminal. Possible values are:

0 Not applicable.

1 Interrupt.

2 Reset.

4 Send to DTE an indication of break PAD message.

8 Escape from data transfer state.

16 Discard output to terminal.

par 8 Discard Selects whether the data output should be sent to the terminal or discarded. Possible values are:

0 Normal data delivery.

1 Discard output to terminal.

par 9 CR Padding Specifies the number of padding characters sent to the terminal after a CR character (Enter)**.** Range 0 to 255.

par 10 - Line Folding Specifies the number of characters sent to the terminal without inserting an appropriate formatting character. Possible values are:

0 No line folding.

1-255 Number of characters.

par 11

Speed

Specifies the line speed of the terminal attached to the PAD. This parameter can only be read, not changed.

Line Speed Parameter Values

Speed (bps)	value	Speed (bps)	value
50	10	1800	7
75	5	75/1200	11
100	9	2400	12
110	0	4800	13
134.5	1	9600	14
150	6	19200	15
200	8	48000	16
300	2	56000	17
600	4	64000	18
1200	3		

par 12
Terminal Flow Control

Controls the flow between the PAD and the terminal. Possible values are:

0 No use of X-ON and X-OFF.

1 Use of X-ON and X-OFF.

par 13
LF/CR

Specifies whether a Line Feed character is inserted after a Carriage Return**.** Possible values are:

0 No linefeed insertion.

1 Send linefeed to the terminal after sending a carriage return.

2 Add a linefeed after each carriage return to the data being sent from the terminal.

4 Echo a linefeed after each carriage return

par 14
LF Padding

Specifies the number of padding characters sent to the terminal after an LF character. Range 0 to 255.

par 15
Editing

Controls use of local editing to the terminal and PAD. The PAD allows the terminal to edit the PAD command signals sent to the PAD before they are processed. Parameters 15 through 19 control this function. Possible values are:

0 No use of editing in data transfer mode.

1 No linefeed insertion.

par 16
Char Delete

Specifies the character from International Alphabet 5 (IA5) that is to request character deletion for the PAD editing buffer.

0 - 127 Valid range. 127 is character 7/15 (DEL).

par 17
Line Delete

Specifies the character from IA5 that causes the PAD to delete the contents of its edit buffer.

0 - 127 Valid range. 24 is character 1/8 (CAN).

par 18
Line Display

Specifies the character from IA5 that causes the PAD to display the contents of its edit buffer.

0 - 127 Valid range. 18 is character 1/2 (DC2).

par 19
Editing Signals

Controls the editing of PAD service signals. Possible values are:

0 No editing allowed.

1 Editing allowed for printing terminals.

2 Editing allowed for display terminals.

8, 32-126
Character from IA5.

par 20
mask Selects what characters are echoed when local echo is enabled (parameter 2). Possible values are:

0 All characters are echoed.

1 CR character is not echoed.

2 LF character is not echoed.

4 VT, HT, FF characters are not echoed.

8 BEL and BS characters are not echoed.

16 ESC and ENQ characters are not echoed.

32 ACK, NAK, STX, SOH, EOT, ETB, and ETX characters are not echoed.

64 Characters designated by PAR 16-18 are not echoed.

128 No echo of characters in columns 0 and 1, not mentioned above, or the DEL character.

Note: If parameters **5**, **12**, or **22** is set to a non-zero value, the characters XON and XOFF are NOT echoed.

par 21
Parity Specifies the parity used when sending data to and from the terminal**.** Possible values are:

0 No parity generated or checked.

1 Parity checking.

2 Parity generating.

par 22
Page Wait Controls screen scrolling if page wait is to be used. Possible values are:

0 Page wait disabled.

1-255 Number of line feed characters before page wait.

PAD Profiles

Within the terminal PAD, there are four available profiles. Which profile is loaded for a particular session is controlled by the `PROF` command. The command to list what profiles are available is **help prof**. By default, the `PROFILE_51` profile is used.

The following profiles are implemented:

50 Standard profile with minimum textual information.
51 Standard profile with extended textual information.
90 CCITT simple profile.
91 CCITT transparent profile.

The settings of the X.3 parameters for each of the profiles are:

X.3 Parameter Settings for Each Profile				
	50	**51**	**90**	**91**
1	1	1	1	0
2	1	1	1	0
3	2	2	126	0
4	0	0	0	20

5	0	0	1	0
6	5	16	1	0
7	21	21	2	2
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	14	14	14	14
12	1	1	1	0
13	5	5	0	0
14	0	0	0	0
15	1	1	0	0
16	8	8	127	127
17	3	3	24	24
18	18	18	18	18
19	2	2	1	1
20	64	64	0	0
21	0	0	0	0
22	0	0	0	0

PAD for AIXlink/X.25 Running on AIX Version 4 (and later)

The configuration files for the AIXlink/X.25 product are:

x28parm Defines the profiles and its values for the PAD X.28 module, along with standard values of **50**, **51**, **90**, and **91**. Three new values given are: **10** (default), **20** (titn_test), and **30** (print).

x29parm Defines the profiles and its values for the PAD X.29 module. The values for the five preset profiles given are:

- 90 (CCITT Standard Simple Profile)
- 91 (CCITT Standard Transparent Profile)
- 10 (Default Profile)
- 20 (AIX/TITN Extended Compatibility Profile)
- 30 (Remote Printing Profile).

x29tty Defines the tty profile and its attributes for the PAD X.29. The default is ″cooked″ mode.

x28user Selects the values for an X.28 user. The default uses the default profile and permits all of the calling addresses.

x29user Selects the values for an X.29 user. The default uses the ″cooked″ tty mode and the default profile.

x29access Selects key features based on the address. The default uses the ″cooked″ tty mode and the default profile.

qdata Defines the values needed for locally initiated PAD printing.

Default Initial Application

Allows a host user connected through X.25 and X.29 to select the initial application. The criteria that must be met are configurable and based on the X.25 calling address. Only a highly privileged user (such as a network administrator) can configure the relevant data that resides on the host.

Three kinds of initial applications can be started upon acceptance of an incoming X.25 call packet using the X.29 protocol ID:

logged user

Login is done in the normal way and conventional password validation and security processing are performed. There can be multiple ″logged users″ per X.25 calling address.

This kind of initial application processing is started based on the following:

- The X.25 calling address is not in the **x29access** configuration file.
- The X.25 calling address is configured, but additional X.29 access criteria (specified in the **x29access** file) are not satisfied.

  A ″logged user″ that supplies a login has a non-default initial TTY-X.29/X.3 profile set under the following two conditions:

- The user has an X.25 address in the **x29access** file, and the data for that address references a valid TTY-X.29/X.3 profile set.
- The user's login is specified in the **x29user** configuration file, and the data for that login references a valid TTY-X.29/X.3 profile set.

  It is possible for multiple references to the same TTY-X.29/X.3 profile set. For more information, see Selectable Profile .

trusted user

Login is done so that it circumvents login ID and password validation. There is only one ″trusted user″ per X.25 calling address.

This initial application processing is started when the X.25 calling address is configured as ″trusted″ in the **x29access** configuration file.
Note: The **x29d** daemon must be restarted so that the **x29access** file will be re-read.

The system requires that **/etc/security/user** and **/etc/security/login.cfg** files be configured for trusted users. This can also be done through the System Management Interface Tool (SMIT).

A trusted user has a non-default TTY-X.29/X.3 profile set if the **x29access** data for the X.25 calling address is configured to reference a valid profile set. For more information, see Selectable Profile .
Note: Since there is only one trusted user per X.25 calling address and no login is supplied for the login process, selection of a profile set is based on the X.25 calling address. This does not prevent a trusted user from selecting an initial profile set by other means.

selective user

Starts an application preconfigured in the **x29access** configuration file. The application is responsible for security arrangements.

Additional criteria linked to the X.25 calling address may need to be satisfied before the initial application is started and the ″selective user″ actually exists. The criteria are specified in the **x29access** file on a per-X.25 calling address basis. The X.25 calling address must match one in the **x29access** file.

For each X.25 calling address in the **x29access** file, there is an `access_class` field that specifies one of the following values:

- **REMOTE**

  The X.25 calling address is the only criteria, and it must match an address in the **x29access** file. The format of the X.25 address may include variable-match syntax (wildcard characters).
- **USER_DATA**

  A subset of the X.25 call packet user data must match user data in the **x29access** file for the corresponding X.25 calling address. The format of the user data also may include wildcard characters.
- **SUB_ADDRESS**

  Must match an X.25 called address specified in the **x29access** file.
  Note: This criteria is evaluated only if the **REMOTE** value is satisfied.
- **USER_SUB_ADDRESS**

  All criteria, including the **REMOTE**, **USER_DATA**, and **SUB_ADDRESS** values, must be satisfied.
- **LOGGED**

  Applies to ″logged″ rather than ″selective″ users. The initial application started for the X.25 calling address is login. For more information, see logged user .
- **TRUSTED**

  Applies to ″trusted″ rather than ″selective″ users. The initial application started for the X.25 calling address is login. The format and characteristics of this value are the same as that for the **REMOTE** value.

  A ″selective user″ must satisfy the criteria described for the **REMOTE**, **USER_DATA**, **SUB_ADDRESS**, or **USER_SUB_ADDRESS** values. If the **x29access** data for the X.25 calling address is configured to reference a valid profile set, a ″selective user″ will have a non-default TTY-X.29/X.3 profile set. For more information, see Selectable Profile .

Notes:

1. A TTY-X.29/X.3 profile set has the following two components:
   - A TTY characteristic profile defined in the **x29tty** file.
   - An X.29/X.3 parameter profile defined in the **x29parm** file
2. Ambiguity is possible in X.25 calling addresses that contain variable-matching (wildcard) characters. The algorithm for searching the **x29access** file defines a match as the first address in the file that matches the one from a call packet. Consequently, if there are similar addresses in the **x29access** file, the order in which they must be arranged is from those with the most rigid matching conditions to those with the least rigid conditions.

Selectable Profile

The selectable profile mechanism allows for the selection of non-default profiles.

For outgoing PAD application connections (X.28/X.25), new functionality allows non-default X.28/X.3 parameter profiles to be selected by a name or a CCITT-defined profile number.

For incoming X.29/X.25 connections, both non-default X.29/X.3 profiles and TTY parameter profiles (TTY-X.29/X.3 profile sets) can be selected. TTY-X.29/X.3 profile sets are selected based on data extracted from the X.25 call packet and a variety of configuration data located in the **x29access** and **x29user** files on the host.

Incoming X.29/X.25 Connection Profile Selection

The method for selecting non-default TTY-X.29/X.3 profile sets is quite different for ″logged users″ versus other kinds of users (for example, ″trusted″ and ″selective″ users).

Depending on how the data in the **x29access** and **x29user** files is configured and the user's login, there are several ways a ″logged user″ can select non-default TTY-X.29/X.3 profile sets. In all cases, a member-type profile must be referenced in order to select a non-default profile set. The **x29parm** file contains members of the X.29/X.3 parameter-type, and the **x29tty** file contains members of the TTY characteristic-type. A profile set is composed of an X.29/X.3 parameter profile and a TTY characteristic profile.

Non-default TTY/X.3 profile sets are selected based on the following:

The profile set is referenced only in the x29access file.: This is only possible if the X.25 calling address is in **x29access**. If so, the `tty_profile` field or the `x3_profile` field of the corresponding data set reference the profile set, and the referenced members are in the **x29tty** and **x29parm** files.

The `tty_profile` field can contain the name or the alias of a TTY characteristic profile in the X.29/TTY profile file. If `tty_profile` is not set, the default TTY characteristic profile is selected.

The `x3_profile` field may contain the name or number designating an X.29/X.3 parameter profile in the X.29/X.3 profile file. If `x3_profile` is not set, the default X.29/X.3 parameter profile is selected.

The profile set is referenced only in the x29user file.: This is only possible if the login supplied is in the **x29user** file. If so, the `tty_profile` or `x3_profile` fields for the corresponding data set reference the profile set, and the referenced members are in the **x29tty** and **x29parm** files.

The profile set is referenced in both the x29access file and the x29user file.: This is only possible if the X.25 calling address is in the **x29access** file, and the login supplied is in the **x29user** file. For example, the login must be supplied by the ″logged user″ rather than extracted from the **x29access** file. If so, the `tty_profile` or `x3_profile` fields for the corresponding data sets reference the same profile set, and the referenced members are in the **x29tty** and **x29parm** files.

The multiple references to the same TTY-X.29/X.3 profile set must be resolved. Both the **x29access** and **x29user** files contain `tty_priority` and `x3_priority` fields for exactly this purpose. The priority fields can be initialized to **-1**, which means they are not to be used. The basic algorithm is as follows:

```
If both the x29access and x29user priority are not -1
   If the x29access priority < x29user priority,
     Use the reference in x29access.
   Else
     Use the reference in x29user.
Else
If the x29access priority is not -1,
   Use the reference in x29access.
Else
Use the reference in x29user.
This means if no priority is specified
the reference in x29user automatically has priority.
```

Notice that the lower the numerical value the higher the priority. Also, the algorithm is executed independently for TTY characteristic and X.29/X.3 parameter profiles. Finally, once the references are resolved, the `tty_profile` and `x3_profile` fields function the same as specified above.

Profile sets have no explicit references.: In this case, there is no X.25 calling address in the **x29access** file and no login in the **x29user** file. Consequently, there is no choice but to use the default TTY-X.29/X.3 profile set.

The ″trusted″ and ″selective″ users can select the TTY-X.29/X.3 profile sets when the `tty_profile` field or `x3_profile` field in the **x29access** file reference a profile set member in the **x29tty** or **x29parm** files, respectively.

If the `tty_profile` field or the `x3_profile` field in the **x29access** file references a profile set member in the **x29tty** file or the **x29parm** file, the non-default profile members are selected.

The `tty_profile` and `x3_profile` fields used in the **x29access** and **x29user** files have the following syntax:

```
tty_profile = [NAME]
```

Where `NAME` can be the name or alias for a profile in the **x29tty** file, and it must be a printable ASCII string. If `NAME` is not specified, there is no profile reference.

```
x3_profile = [NAME | NUMBER]
```

Where `NAME` is the name of a profile in the **x29parm** file and must be a printable ASCII string not beginning with a : (colon) or # (number sign). `NUMBER` is the numerical identifier of a profile in the **x29parm** file and consists of the # (number sign) character followed by a string of ASCII characters 0 through 9. If neither is specified, there is no profile reference.

Outgoing X.28/X.25 Profile Selection

The PAD application can select a non-default initial X.28/X.3 profile in three ways:

- Using **xspad -p** *Profile*, where *Profile* is a selected name or a defined numeric profile number.
- Preconfiguring a non-default profile in the **x28user** file. It is prerequisite that the file contains a data set for the user's login. Then, the `x3_profile` field can be used to specify a profile.
- Using the standard **profile** command. See the **xspad** command in Appendix A for more information.

The available profiles are all stored in the **x28parm** file.

Distribution of Profile Data

All of the X.28/X.3 parameter profiles are stored in the **x28parm** file. However, the **xspad** command does not directly extract profiles from the **x28parm** file.

Before any profile selection occurs, the X.28 STREAMs module is pushed on X.25. At that time, all the profiles in the **x28parm** file are downloaded to the X.28 STREAMs module. This is necessary because a new non-initial profile can be selected at a later stage of **xspad** execution.

Configurable Profile

The **x28parm** file contains all the X.3 parameter profiles and is used to configure the characteristics of outgoing X.28/X.25 PAD sessions.

The **x29parm** file, which contains all the X.3 parameter profiles, and the **x29tty** file, which contains all the TTY characteristic profiles, are used to configure the characteristics of incoming X.29/X.25 sessions.

Note: The **x29d** daemon must be restarted so these files will be read in the outgoing case.

Both the **x28parm** and the **x29parm** files have the same format. Each profile contains all 22 of the standard CCITT X.3 parameters, descriptive information, and optionally some network-specific parameters. Each profile has a unique name and numerical identifier.

All the fields must be present even if they are not used. The fields can be in any order, although it is recommended that they be kept in increasing X.3 numerical order. An exception is the **x3_11_dte_speed** parameter, because it is a read-only parameter and ignored.

A syntax enhancement allows the profiles to be referenced both by name and standardized CCITT numeric profile identifiers:

```
NUMBER:NAME
```

Where NUMBER is the numerical profile identifier represented as a contiguous string of ASCII characters 0 through 9, and NAME is a descriptive profile identifier and a string of printable ASCII characters not beginning with a : (colon) or # (number sign) character:

```
90:ccitt_default_profile
```

The following new fields have been added:

- x3_english_text
- x3_french_text
- x3_spanish_text
- x3_23_national_parm_1
- x3_24_national_parm_2
- x3_25_national_parm_3.

The **x29tty** file contains TTY characteristic profiles. Each profile has a unique name and possibly an alias name. The field names are similar to those used in the normal TTY structures, and the semantics are the same. It is easy to specify TTY characteristics, because almost all TTY characteristics can be enabled or disabled by setting the corresponding field to **ON** or **OFF**. All the fields must be present, but the order is optional.

To allow aliases, use the following syntax:

```
NAME:[ALIAS]
```

Where NAME and ALIAS are descriptive profile names consisting of printable ASCII characters beginning with either the : (colon) or # (number sign) character.

```
cooked_profile:default
```

Some fields that represent a character have the following syntax:

```
FIELD_NAME = CHAR_VALUE
```

Where FIELD_NAME is a printable ASCII string without white space, and CHAR_VALUE is either a single ASCII character or an ASCII character prefixed with the ″^″ character:

```
erase = @
kill = ^H
```

Note: The TTY characteristics specified by the tab0_tabs, eof_vmin_min, and eol_vtime_time fields are context-dependent.

Security Features

Security for outgoing X.28/X.25 PAD sessions is on a per-user basis and is implemented by restricting the X.25 addresses to which a particular user can connect.

Security for incoming X.29/X.25 sessions depends on the user's X.29 access category:

logged user Uses the conventional login security, and the functionality is implemented.

trusted user	Controlled by the X.25 calling address and data in the **x29access** file. **Note:** The implementation changes do not change the security method.
selective user	Controlled by data in the **x29access** file.

PAD Printing

The following packages must be installed to use PAD printing:

- **printers.rte**
- **printers.rte.** (printer type being used)
- **printers.hpJetDirect.attach**

  Note: The **printers.hpJetDirect.attach** package is important because rather than X.25 PAD creating its own **pioin** and **pioout** applications, it uses the standard that comes with this package.

PAD printing can be accomplished by one of the following methods:

- The remote PAD printer can poll the host for print jobs queued.
- The host can automatically transmit queued jobs to the PAD printer. This method can only be used if the PAD printer has a unique NUA and if a VC between the PAD printer and the host has been established and does not interfere with other devices attached to the PAD.

Both methods require an X.25/X.29 connection between the host and the PAD to which the remote printer is attached.

The host needs substantial configuration and some software modules to support either PAD printing method.

There are two categories of configuration data. The first describes remote printer attributes, such as **speed**, and the second associates NUAs and printer queues with actual remote printers.

The software functionality specific to the host-initiation method is the establishment of an X.25/X.29 connection.

The software functionality specific to the remote-initiation method is as follows:

- Acceptance of an incoming X.25/X.29 connection and the incidental criteria validation.
- Association of a destination NUA subaddress with a particular printer queue.
- Configuration of X.3 parameters.
- Activation of a queue.

The software functionality specific to both PAD printing methods is:

- Obtaining jobs from the front of a printer queue.
- Obtaining ″control″ of an X.25/X.29 remote printer connection.
- Transmission of a job at a rate that does not overrun the remote printer.
- Releasing an X.25/X.29 connection when **piobe** stops passing jobs.

Remote Initiation

The printer must query the host for print jobs. Printing is initiated when a remote PAD transmits an X.25/X.29 call to the host. Upon reception of the call, the host uses preconfigured data to identify a specific printer queue associated with the X.25 destination subaddress.

Since the PAD and printer have the same X.25 source address, use the X.25 destination sub-address to identify the printer. This does not affect the criteria for validating the X.25 source address.

Once the X.25/X.29 connection is complete and the printer queue is identified, any jobs on the queue are transmitted to the printer. The X.25/X.29 connection is then cleared.

This kind of printing is used when a printer does not have its own NUA or the network configuration makes establishment of a VC between the host and printer unfeasible. Printing is usually triggered when the printer is turned on. At this point, an X.25/X.29 call is sent to the host. After the connection is complete, the PAD routes data from the host to the port to which the PAD is attached. The port must be configured with the X.3/X.28 attributes to automatically place an X.25 call.

The host must have mechanisms for ″listening″ for calls that have a destination subaddress corresponding to the remote printer queues, that map the subaddresses to the right queue, detect print jobs on the queues, and transmit data over the X.25 connection from the remote printer. Some commercial products do this by first setting up ″plumbing″ to the right queue, replacing **pioout** with a proprietary program, and finally transmitting the queued print jobs.

Host Initiation

When a print job is queued for the remote printer, the host sends it regardless of how the job was queued. When the host detects a job on the remote printer's queue, it initiates an X.25/X.29 connection using the destination address of the remote PAD to which the printer is attached. There must be a preconfigured mapping between the remote printer queue and the NUA at the host. The initial request for the print job may or may not originate at the PAD to which the printer is attached. If the request originated at the PAD, the printer must have a different X.25 subaddress than other devices attached to the PAD.

When the X.25/X.29 connection is complete, the host transmits the print job and clears the connection. This kind of printing requires that the printer have a unique NUA. Also, it must be likely that the attempts by the host to establish a VC to the printer will succeed at any time, since print jobs can be queued at any time.

PAD printing is triggered when a call with the appropriate NUA is received. The PAD then completes the X.25/X.29 connection and routes data from the host to the port to which the printer is attached. The port must be configured with the appropriate X.3/X.28 parameters and configured to accept incoming calls.

Printing is triggered at the host when a job is placed on the print queue of a remote printer. This is achieved by some products by replacing **pioout** with a proprietary program with the ability to make X.25/X.29 connections and the ability to send the print job without overrunning the remote printer.

PAD Printing Processes

The two processes that interact with the printer **qdaemon** to accomplish pad printing are:

- **piox25**
- **piox25start**

For both remote and local printing, **piox25** is started indirectly each time the **qdaemon** removes a job from a particular print queue. Then, the job is piped from **piobe** (the printer queue backend) to **piox25** that transmits it to the network through an X.25 connection. Pacing and formatting are handled by **piobe**; therefore, the **printers.hpJetDirect.attach** fileset needs to be installed.

The **piox25start** process is used only in the case of remote printing and interacts with **x29d**, as well as **piox25**.

Configuration for Remote Printing

The X.25 calling address of the printer needs to be configured in the **/etc/sx25pad/x29access** file as follows:

1. The `access_class` field must be set to one of the selective user types: **REMOTE**, **SUB_ADDRESS**, **USER_DATA**, or **USER_SUB_ADDRESS**. The `user_data` and `sub_address` fields must have values consistent with the type.

2. The `initial_application` field must be set to **/usr/lpd/piox25start** *QNAME* where *QNAME* is the name of a print queue associated with a remote printer.
3. The `tty_profile` field should be set to **raw**, and the `x3_profile` field should be set to an X.2/X.29 profile consistent with remote printing. Currently, one is available in the **X29parm** file as number 30 or the name ″print.″
4. In the **/etc/qconfig** file, the particular printer queue needs to be configured so the `backend` field points to **/usr/sbin/piox25remote** as shown in the following example:

```
pad_q:
        device = hp@stocks
hp@stocks:
        file = /var/spool/lpd/pio/@local/dev/hp@stocks#hpJetDirect
        header = never
        trailer = never
        access = both
        backend = /usr/sbin/piox25remote pad_q
```

When **x29d** receives an X.25/X.29 call from the specified X.25 calling address indicating the criteria was satisfied, it spawns **piox25start**.

The **piox25start** process does two things:

- Activates the specified print queue using the **enq** command.
- Indicates the name of the device over which the X.28/X.29 connection is established.

When the printer **qdaemon** receives the **enq** request to activate a particular queue, it enables scheduling of print jobs on that queue. When the **qdaemon** schedules a print job, it spawns **piox25** as follows:

- It invokes the **/usr/lpd/piox25remote** shell script. The shell script starts **piobe** which is piped to **piox25**. The **piox25** process is started using the **-r** flag, indicating a remote job, and the **-q** *QNAME* flag, where *QNAME* is the print queue name.
- Upon completion of transmission of the print job, **piobe** closes the pipe causing the **piox25** command to read an EOF. The **piox25** command then waits for the remainder of the print job to be transmitted over the X.25/X.29 connection to the remote printer, and then tries to elicit another print job from **piobe**.

  The steps for eliciting another job while the connection is established are:

  1. Using the **enq** command, deactivate the print queue to prevent timing problems.
  2. Enter **piox25start -r** *DEVICE QNAME*, where *DEVICE* is the name of the device, and *QNAME* is the print queue name.
  3. Use the **piox25** command to exit.

The cycle of running the **piox25start** and the **piox25** commands, alternately, continues until the print queue is empty or the network connection is lost. At that point, the **piox25** command fails to receive SIGUSR1, then exits; and the network connection is closed.

Configuration for Local Printing

The steps for configuring for local printing are as follows:

1. In the **/etc/xs25pad/qdata** file, enter:

   ```
   QNAME LDEVICE DEST
   ```

   Where `QNAME` is a print queue name; `LDEVICE` is a logical device name, such as `sx25a1`; and `DEST` is the X.25 called address of the hardware PAD to which the local printer is connected (for example, 32154123). In most cases, the X.25 called address contains a sub-address corresponding to the hardware PAD port to which the printer is connected.

   Note: This data is used to derive the line number and X.25 calling address from the ODM.
2. In the **/etc/qconfig** file, configure the printer queue so the `backend` field points to the **/usr/sbin/piox25local** file as shown in the following example:

```
pad_q:
      device = hp@stocks
hp@stocks:
      file = /var/spool/lpd/pio/@local/dev/hp@stocks#hpJetDirect
      header = never
      trailer = never
      access = both
      backend = /usr/sbin/piox25local pad_q
```

3. Preconfigure the X.3 parameters for the remote hardware PAD. Features such as parity checking and generation need to be disabled. X.3 parameters, such as **ancillary device control**, need to be set for the type of printer.

Removing a Job from the Print Queue

A job is removed from the print queue by invoking **piox25** as follows:

1. Use the **piox25local** command to pipe the output of **piobe** (the backend) to **piox25**. **piox25** is called only using **-q** *QNAME*, where *QNAME* is the print queue name.
2. Use the **piox25** command to call the **pp_connect( )** subroutine to make a normal X.25 connection to the remote pad. The **piox25** command transmits the print job and then exits closing the X.25 connection. **piobe** closes the pipe causing **piox25** to read EOF when it is finished sending the print job. It is important that **piox25** wait for a substantial period of time before closing the X.25 connection so the tail of the job has time to traverse the network.

Transmission Logic

Both local and remote printing share the same job transmission logic. Data read from **piobe** is sent to the network until EOF is encountered. Since it is possible for transmissions to the network to be blocked, new data is only read from **piobe** when no other transmission is in progress.

On occasion, it may seem that the print job takes some time. In some cases, such as remote initiation, the connection must be established, the printer queue must be set up, the converting of the data from X.25 to printable ASCII must be done, and lastly, the transmission of the job through the printer commands must be completed.

Configuration File Format (AIXlink/X.25 Version 1 and later)

Each PAD configuration file contains a set of stanzas of the following form:

```
key1:
   attribute1 = value1
   attribute2 = value2
   attribute3 = value3
   ...
key2:
   ...
```

Each new key begins after a blank line. The `key` field and some attribute values may contain pattern-matching characters that are defined as follows:

*****	Matches zero or more characters.
?	Matches any single character.
[ ]	Matches any single character in the enclosed set of characters; ranges of characters can be defined using the '-' separator, for example, ″[0-9a-f]″.
|	Separates values in a list of values.

The following is an example of attribute values:

```
12*   Matches any value beginning with "12".
12?   Matches any value beginning with "12" followed by any
   single character.
```

```
[1-3]*   Matches any value beginning with "1", "2", or "3".
12*|23*   Matches any value beginning with either "12" or "23".
```

The configuration file is searched sequentially, and the first key value that matches the search key is used. To avoid redundant attribute definitions, it is possible to use the special **default** attribute to specify another stanza that supplies attribute values not explicitly defined in the matching stanza. If the attribute definition is still not found after searching any specified **default** stanzas, the stanza with the special **default** key is checked. For example:

```
default:
   attribute1 = value1
   attribute2 = value2
   attribute3 = value3
key1:
   attribute1 = value1
key2:
   default    = key1
   attribute2 = value2
key3:
   default    = key2
   attribute3 = value3
```

In this case, the value for `key2`'s `attribute1` comes from `key1`, since it is not defined in `key2`'s stanza; and `key1` is `key2`'s default stanza. Since an arbitrary number of default links can be followed, `key3`'s `attribute1` value also comes from `key1`, as the default links are followed from `key3` to `key2` and `key2` to `key1`. Finally, the value for `key2`'s `attribute3` comes from the default stanza, since neither `key2` nor `key2`'s default stanza, `key1`, defines a value for it.

Call packets can be received without a calling address specified. In these cases, the NULL key value can be used in the **address** file to match a nonexistent calling address.

x29d Usage Changes

The default x29d mode can run as an outgoing PAD call validation daemon and an X.29 listener daemon (a single process with dual functionality).

If started using the **-n** flag, the x29d runs as an outgoing PAD call. If started in the default mode using no flags, x29d runs as an incoming X.29 call server daemon.

You can start x29d as follows:

```
x29d [-a] [-p user] | [-n]
```

`-p user` Sets the initial TTY and X.3 attributes of an X.29 session, where `user` indicates the login ID. This flag cannot be used with the **-n** flag and requires that x29d is already running as the X.29 listener daemon.

`-a` Outputs the calling X.25 address of the system initiating the X.29 session to stdout. x29d runs and obtains the X.25 address, but a login is not needed since this flag is not related to X.3 or TTY attributes. This flag cannot be used with the **-n** flag and requires that x29d is running as the X.29 listener daemon.

`-n` Starts x29d as a outgoing call validation daemon. The **-a** and **-p** flags cannot be used with this flag.

Chapter 9. X.25 Simple Network Management Protocol

Support for a Simple Network Management Protocol (SNMP) proxy agent comes with AIXlink/X.25 Version 2.0. . This support allows statistical data related to the X.25 interfaces to be gathered and transferred to an SNMP agent for analysis. The items of data gathered form a subset of the Management Information Base for X.25 specified in RFCs 1381 and 1382 . RFC 1381 covers the frame layer and 1382 the packet layer.

For more details on the individual MIB objects, refer to the appropriate RFCs.

Data collection from X.25 for SNMP is enabled and disabled by the config methods:

- Enabled at the time the first port is configured.
- Disabled at the time the last port is unconfigured.

Installation Notes for SNMP Multiplexer X.25 Peer Daemon (x25smuxd)

It is assumed that the **snmpd** is running and all the necessary software needed to ensure that the **snmpd** is properly functioning is installed and working. The X.25 proxy agent **x25smuxd** and the MIB defs file **x25smuxd.defs** resides in `/usr/sbin` directory. Before invoking **x25smuxd** by the config methods at the port configuration time, use the following installation procedure:

1. Obtain root user authority.
2. Check to see if the `/usr/sbin/x25smuxd.defs` file is installed.
3. In the /usr/sbin/x25smuxd.defs file, find the following line:

   ```
   -- object definitions compiled from RFC1381-MIB { iso 3 6 1 2 1 }
   ```

   Append every line from this line on to the end of the file to the bottom of the `/etc/mib.defs` file.
4. Add the following entry to the bottom of the `/etc/snmpd.peers` file:

   ```
   "x25smuxd"     1.3.6.1.2.1.10.16     "x25smuxd_password"
   ```
5. Add the following entry to the bottom of the `/etc/snmpd.conf` file:

   ```
   smux     1.3.6.1.2.1.10.16     x25smuxd_password     #x25smuxd
   ```
6. Refresh the **snmpd** daemon so that it rereads the `/etc/snmpd.conf` file with the following command:

   ```
   refresh -s snmpd
   ```

   Notes:

   a. Only run **x25smuxd** when you are logged in with root user authority.

   b. Never start more than one instance of **x25smuxd** as it can cause conflicts with the interprocess communication mechanism.

   c. Line status change notification is not implemented in this release of X.25 Licensed Program.

Frame Layer Objects

The following objects from RFC1381 are supported in read-only mode:

LapbOperEntry
lapbOperIndex
lapbOperControlField
lapbOperTransmitN1FrameSize
lapbOperReceiveN1FrameSize
lapbOperTransmitKWindowSize
lapbOperReceiveKWindowSize
lapbOperN2RxmitCount

lapbOperT1AckTimer
lapbOperT2AckDelayTimer
lapbOperT3DisconnectTimer
lapbOperT4IdleTimer
lapbOperPortId

LapbFlowEntry
lapbFlowIfIndex
lapbFlowCurrentMode
lapbFlowRejOutPkts
lapbFlowRejInPkts
lapbFlowT1Timeouts

Packet Layer Objects

The following objects from RFC1382 are supported in read-only mode:

X25OperEntry
x25OperInterfaceMode
x25OperPacketSquencing
x25OperRestartTimer
x25OperCallTimer
x25OperResetTimer
x25OperClearTimer
x25OperWindowTimer
x25OperDataRxmtTimer
x25OperInterruptTimer
x25OperRejectTimer
x25OperRegistrationRequestTimer
x25OperRestartCount
x25OperResetCount
x25OperClearCount
x25OperDataRxmtCount
x25OperRejectCount
x25OperRegistrationRequestCount
x25OperNumberPVCs
x25OperDataLinkId

X25StatEntry
x25StatIndex
x25StatInCalls
x25StatInRestarts
x25StatInDataPackets
x25StatInInterrupts
x25StatOutCallAttempts
x25StatOutInterrupts
x25StatOutDataPackets
x25StatRestartTimeouts
x25StatCallTimeouts
x25StatResetTimeouts
x25StatClearTimeouts
x25StatDataRxmtTimeouts
x25StatInterruptTimeouts

X25ChannelEntry
x25ChannelIndex
x25ChannelLIC
x25ChannelHIC
x25ChannelLTC
x25ChannelHTC
x25ChannelLOC
x25ChannelHOC

X25CircuitEntry
x25CircuitIndex
x25CircuitChannel
x25CircuitInOctets
x25CircuitInPdus
x25CircuitInInterrupts
x25CircuitOutOctets
x25CircuitOutPdus
x25CircuitOutInterrupts
x25CircuitDataRetransmissionTimeouts
x25CircuitResetTimeouts
x25CircuitInterruptTimeouts

Chapter 10. Common Input/Output Emulation

Prior to AIX Version 4, there was base X.25 support that provided a number of programming interfaces, higher layer protocols and applications. This support was provided through devices such as `/dev/x25s0`. With the X.25 support provided by the separate Licensed Programs, this former interface is emulated through common input/output (COMIO) emulation.

Each X.25 port `sx25a` that requires COMIO emulation should have it selected through SMIT configuration. COMIO emulation is provided as a compatibility interface for existing user applications. NPI and DLPI are the interfaces for new application development.

Note: All the documentation in this chapter refers to use of the COMIO emulator, its programming interfaces or applications. Ensure that a suitable COMIO emulation port is available for use.

X.25 Application Programming Interface Overview

The X.25 application programming interface (API) can be used to write applications tailored to specific needs.

This overview provides the following information about programming X.25 communications:

- X.25 Application Programming Interface (API)
- Processing Calls with the X.25 API
- X.25 Example Programs

X.25 Application Programming Interface (API)

X.25 communications can be used to provide a network service for higher-level protocols, such as SNA, or directly with commands or the API. The X.25 commands can be used as soon as you have set up X.25 communications. The application programming interface (API) can be used to write applications tailored to specific needs. The following sections discuss portions of the X.25 API:

- Using the X.25 Structures and Flags
- /dev/x25sn Special File
- X.25 Error Codes
- Using Processes in X.25 Applications

Before you can use the COMIO emulation X.25 API, the X.25 Licensed Program must be installed and configured, and emulation ports added as required. You also need access to a C compiler.

The X.25 API includes a library of C subroutines that use the services of the X.25 adapter and adapter code. Application programs call these subroutines to access X.25 functions. The subroutines use a number of structures to pass information between the X.25 functions and the application programs. Further information can be found in:

- Using the X.25 Structures and Flags

The X.25 API provides the following types of identifiers for use in programs:

- Listen identifiers, **listen_id**, for potential incoming calls
- Connection identifiers, **conn_id**, for established calls
- Counter identifiers, **ctr_id**, for notification of incoming messages

In addition to the subroutine library and the header files for the structures, there are example programs that demonstrate the use of the subroutines. Further information can be found in “X.25 Example Programs” on page 142.

Background information for using the subroutines is included in “Processing Calls with the X.25 API” on page 135.

Using the X.25 Structures and Flags

For many of the subroutines, parameters are placed in a structure, and the subroutine is passed a pointer to this structure. Definitions of these structures are supplied as a header file, **/usr/include/x25sdefs.h**. Include the following line in programs:

```
#include <x25sdefs.h>
```

x25sdefs.h lists all the structures included in the **/usr/include/x25sdefs.h** file.

Each of the fields in a structure has an associated flag. This flag tells the API whether the associated field has been used. If the corresponding flag has not been set, the field is ignored by the API. To use the flag, which is a constant, **OR** it with the **unsigned long flags** in the structure. This sets the appropriate bit in the `flags` field.

Before invoking a subroutine, the appropriate `flags` field must be set to 0 or to a particular flag constant. For example, to set the `flags` field to 0 before invoking the **x25_call** subroutine, use the following:

```
cb_call.flags = 0
```

To indicate that the `link_name` field is being used, before invoking the **x25_call** subroutine, use the following:

```
cb_call.flags = X25FLG_LINK_NAME
```

Some flags, for instance, **X25FLG_D_BIT**, do not correspond to structure elements.

/dev/x25sn Special File

The **/dev/x25s***n* special file is provided through the COMIO emulation migration.

The emulator supports the **/dev/x25s***n* special file as a character-multiplex special file. The special file must be opened for both reading and writing (**O_RDWR**). There are no particular considerations for closing the special file. The special file name used in an **open** call differs depending on how the device is to be opened. Types of special file names are:

/dev/x25s*n*	Starts the device handler on the next available port.
/dev/x25s*n***/R**	Starts the device handler for updating the routing table.

X.25 Error Codes

The X.25 subroutines set the **x25_errno** and **errno** flags to indicate error conditions.

If an error condition results from an X.25 API subroutine call, it is indicated in one of the following ways:

- For X.25-specific error conditions, the **x25_errno** flag indicates the error; for example, `X25ACKREQ`. The **errno** flag is not set in these conditions.
- For other error conditions, the **x25_errno** flag is set to `X25SYSERR` and the **errno** flag indicates the error, for example, `EFAULT`.

In a production program, check for each condition that is likely to occur, giving the end user a message telling what action to take. The code example for the **x25_link_statistics** subroutine shows how to do this. All the other subroutines can be handled in a similar way.

API Error Codes lists the error codes that may be returned by X.25 subroutines.

Using Processes in X.25 Applications

To improve performance, applications should be divided into multiple processes. For instance, you can start a separate process for sending or receiving data. Or, you can have one process to listen for calls and one process to make calls.

Child processes should not be created while a call is being established or after an incoming call has been received. However, you can start a process:

- Before making a call
- After receiving confirmation of an outgoing call
- Before receiving an incoming call

After an incoming call is successfully received, an open file is created in the current process that is used by the API library functions for subsequent communication on this connection. The file is closed when the call is cleared. Creating child processes after the call is received may leave the file open indefinitely, leading to an error due to reaching the limit on open files (**errno 23**).

A connection identifier can be used by the process that made or received the call, or by the process' children. It cannot be used by other processes.

A child process can use a virtual circuit established by its parent, but a parent process cannot use a virtual circuit established by its child.

Processing Calls with the X.25 API

This API-level overview tells you how to use the application programming interface through the emulation port for both switched and permanent virtual circuits. The following refer to the example programs and discuss how the subroutines are used:

- Initializing and Terminating
- Using the Connection Identifier for Calls
- Using Counters to Correlate Messages
- Listening for Incoming Calls
- Making and Receiving a Call
- Transferring and Acknowledging Data
- Clearing, Resetting, and Interrupting Calls

Initializing and Terminating

The application programming interface (API) must be initialized for a specific X.25 port before any other subroutines can be used on that port. If the program uses more than one X.25 port, the API must be initialized for each. Use the **x25_init** subroutine (as in example program **svcxmit**).

If the application uses a permanent virtual circuit (PVC), you must use the **x25_pvc_alloc** subroutine to allocate the PVC, identifying it by its logical channel number and X.25 port name (as in example program **pvcxmit**). Use SMIT to find out which logical channel numbers are valid.

A PVC must be freed, using the **x25_pvc_free** subroutine, before the program is terminated (as in example program **pvcxmit**).

You must terminate the API for each X.25 port, using the **x25_term** subroutine (as in example program **svcxmit**). However, before terminating the API for a port, do the following:

1. Clear any calls, using **x25_call_clear.**
2. Remove any counters, using **x25_ctr_remove.**
3. Stop listening for calls, using **x25_deafen**.
4. Free any permanent virtual circuits, using **x25_pvc_free**.

Using the Connection Identifier for Calls

Because the API or a single application can simultaneously control multiple virtual circuits, there must be a way of identifying a call uniquely. To do this, the API assigns to each call a positive integer known as the *connection identifier*.

The *conn_id* parameter is used by the API subroutines to pass the connection identifier.

Obtaining a Connection Identifier

On a switched virtual circuit, for an outgoing call, the connection identifier is returned by the **x25_call** subroutine (as in example program **svcxmit**). When receiving an incoming call the connection identifier is allocated by the **x25_receive** subroutine to the first of its parameters (as in example program **svcrcv**).

On a permanent virtual circuit, the connection identifier is returned by the **x25_pvc_alloc** subroutine (as in example program **pvcxmit**).

Using a Connection Identifier

The connection identifier is assigned on return from these subroutines:

- Using the **x25_call** subroutine to make a call on a switched virtual circuit
- Using the **x25_pvc_alloc** subroutine to establish a permanent virtual circuit
- Using the **x25_receive** subroutine to receive an incoming call
- Using the **x25_receive** subroutine to receive data from any currently connected call

The connection identifier is passed as a parameter to these subroutines:

- Using the **x25_receive** subroutine to receive data from a particular call
- Using the **x25_call_accept** subroutine to accept a call
- Using the **x25_send** subroutine to send data
- Using the **x25_ack** subroutine to acknowledge data
- Using the **x25_call_clear** subroutine to reject or terminate a call
- Using the **x25_reset** subroutine to reset a call
- Using the **x25_reset_confirm** subroutine to confirm that a reset arrived
- Using the **x25_interrupt** subroutine to interrupt a call
- Using the **x25_pvc_free** subroutine to free a permanent virtual circuit
- Using the **x25_circuit_query** subroutine to get information about a virtual circuit

Restrictions on the Use of the Connection Identifier

A connection identifier can be used only by the process that made the call, established the permanent virtual circuit, or received the call, and by that process' child processes. Any attempt to use the connection identifier of another process results in the **X25BADCONNID** error code.

Using Counters to Correlate Messages

Many applications use the network at once and each application may have several calls active at one time. An application may also be listening for calls for several different routing list names. How does the application know when a message has arrived on a particular virtual circuit, or for a particular call? A *counter*, supplied by the API, is incremented when a message arrives. The application issues an **x25_ctr_wait** subroutine, which returns when the counter has been incremented. The counter is decremented when the message has been received (using the **x25_receive** subroutine).

Counters allow an application to wait for messages on several virtual circuits at one time; it is the responsibility of the application to correlate counters with particular virtual circuits. Optionally, an application can use the **x25_ctr_wait** subroutine to accumulate several messages against a particular counter before being notified.

Counter Identifiers

Each counter has a *counter identifier*. The **ctr_id** parameter is used by some of the API subroutines to pass the counter identifier. The **x25_ctr_wait** subroutine uses an array of structures (**ctr_array_struct**) each of which contains a counter identifier and a value. This allows an application to wait for any of a number of counters to change.

Counters in Applications

You must decide how to use counters in your application depending on what the application has to do. Use of counters is not required, but the use of the **x25_ctr_wait** subroutine is the recommended way of notifying the application that a message has arrived.

For an application that makes calls, use a separate counter for each call. For an application that listens for and receives calls, use one counter to listen for incoming calls and then use a separate counter to accept each call and receive its subsequent messages. For an application that receives messages from any one of a number of connected calls, use a single counter.

The application is responsible for ensuring that it gets enough counters.

Obtaining a Counter

The application gets a counter from the API by calling the **x25_ctr_get** subroutine (as in example program **svcxmit**). This subroutine returns a counter identifier that is unique across the system.

The two applications (the one that makes a call and the one that receives it) each use a different counter for the call. Each tracks the messages independently.

Using a Counter

The counter identifier is passed as a parameter to the following subroutines:

- The **x25_call** subroutine assigns a counter to a specific connection when making a call on a switched virtual circuit.
- The **x25_pvc_alloc** subroutine assigns a counter to a specific connection when establishing a permanent virtual circuit.
- The **x25_listen** subroutine assigns a counter to a listening process when starting to listen for incoming calls.
- The **x25_call_accept** subroutine assigns a counter to a specific connection when accepting a call.
- The **x25_ctr_wait** subroutine uses an array of counters to wait for an incoming call or a message.
- The **x25_ctr_test** subroutine uses one counter to determine how many messages are waiting to be received for a call.

Waiting for an Incoming Call or a Message:

Normally, the **x25_ctr_wait** subroutine notifies an application program that a message has arrived. The program invokes the **x25_ctr_wait** subroutine, passing it a pointer to an array of counter structures. This enables an application to wait for messages from more than one call.

The example programs show the **x25_ctr_wait** subroutine being used in several situations, but always with only one counter. If you want to wait for messages using multiple counters, you must assign them all to the **ctr_array_struct** structure before invoking the **x25_ctr_wait** subroutine.

> **Note:** If you are writing a program that uses multiple counters to identify multiple calls, you are responsible for storing the counter identifiers with their corresponding connection identifiers.

Example Uses of the x25_ctr_wait Subroutine:

1. Set up the **ctr_array_struct** structure and wait for an incoming call (as in example program **svcrcv**).
2. Wait for an acknowledgement indicating that the **ctr_array_struct** structure was set up earlier in the program (as in example program **svcxmit**).

Determining How Many Messages Are Waiting to Be Received for a Call: The **x25_ctr_test** subroutine is provided to determine the number of messages waiting to be received for a call, as follows:

1. Assign the counter identifier to the *ctr_id* parameter.
2. Invoke the **x25_ctr_test** subroutine, passing *ctr_id* as a parameter.
3. The return value is the number of messages waiting to be received.

However, if you use the **x25_ctr_wait** subroutine when expecting a message to arrive and receive every message when it arrives, you should not need to use the **x25_ctr_test** subroutine.

Removing a Counter

Before an application terminates, it must remove all counters in use. A counter cannot be removed while its value is greater than 0, indicating that there is a message to be received. First, receive any messages, and then use the **x25_ctr_remove** subroutine, passing it the counter identifier as a parameter (as in example program **svcxmit**).

Restrictions on the Use of Counters

Any application can test the value of a counter or wait for it to change. Only the application that requested the counter with the **x25_ctr_get** subroutine, or a root user, can remove the counter using use the **x25_ctr_remove** subroutine.

Listening for Incoming Calls

When it is listening for calls, the **x25_listen** subroutine uses a positive integer, the listen identifier, to identify an incoming call. After the call has been received and accepted, the listen identifier is used again to listen for subsequent incoming calls.

Obtaining a Listen Identifier

The **x25_listen** subroutine can be used to listen for incoming calls.

1. Get a counter.
2. Invoke the **x25_listen** subroutine, passing it two parameters: the counter identifier and the name of an entry in the X.25 routing list (as in example program **svcrcv**).
3. The **x25_listen** subroutine returns a listen identifier.

Using the Listen Identifier

After obtaining a listen identifier, the application must wait for an incoming call. When an incoming call arrives for that listen identifier, the application assigns the listen identifier to the *conn_id* parameter and uses the **x25_receive** subroutine to receive the incoming call.

Removing the Listen Identifier (Stop Listening):

1. Remove the counter associated with this listening process.
2. Invoke the **x25_deafen** subroutine, passing it the listen identifier as a parameter. Always do this before terminating a program that has been listening for incoming calls (as in example program **svcrcv**).

Restrictions on the Use of the Listen Identifier:

The use of this variable is restricted to the user who received the *listen_id* from the **x25_listen** subroutine. The user may have one application that listens and notifies the user of an incoming call, and another application that actually receives the call.

Making and Receiving a Call

The following information describes the processes involved in making and receiving a call.

Making an Outgoing Call

To make a call on a switched virtual circuit (SVC) (as in example program **svcxmit**):

1. Set up the **cb_call_struct** with the relevant information.
2. Invoke the **x25_call** subroutine, passing two parameters: a pointer to **cb_call_struct** and a counter identifier.
3. The **x25_call** subroutine returns a connection identifier, which the application must use to identify the call.
4. Store a counter identifier with the connection identifier.

When using a permanent virtual circuit (PVC), do not make any calls. Once you have allocated a PVC, you can send and receive data until you free the PVC.

After making a call, the calling application must wait for the called application's response, using the **x25_ctr_wait** subroutine, and then receive it, using the **x25_receive** subroutine (as in example program **svcxmit**). The response can be either a call-connected message or the clear-indication message.

Receiving an Incoming Call

When you know there is an incoming call waiting because the counter associated with the listen identifier has been incremented, you must use the listen identifier to receive the incoming call (as in example program **svcrcv**).

1. Assign the listen identifier to the *conn_id* parameter.
2. Invoke the **x25_receive** subroutine, passing two parameters:
   - The address of *conn_id* (which currently contains the listen identifier).
   - A pointer to the message control block, **cb_msg_struct**.
3. On return, the **x25_receive** subroutine assigns the *connection identifier* of the incoming call to the *conn_id*. (The listen identifier is still valid for further incoming calls.)
4. On return from the **x25_receive** subroutine, the message control block includes the **msg_type**, which indicates the type of message (for example, `X25_INCOMING_CALL`). You do not need to check it because it is the only message that can be received using the listen identifier. The **cb_call_struct** control block contains the incoming-call message, which may include call user data.
5. Free any structures allocated by the **x25_receive** subroutine (as in example program **svcrcv**).

Accepting or Rejecting an Incoming Call

To accept an incoming call after receiving it (as in example program **svcrcv**):

1. Get a new counter, to be used for accepting the call and receiving any subsequent messages for it. (This allows the counter that was used for listening to continue to be used to listen for calls.)
2. Optionally, set up **cb_call_struct** with the relevant information.
3. Invoke the **x25_call_accept** subroutine, passing the connection identifier, the counter identifier, and **cb_call_struct** as parameters.
4. The **x25_call_accept** subroutine sends an `X25_CALL_CONNECTED` message, which must be received by the caller.

At this point, after accepting a call, you should deal with the call user data if necessary. After dealing with the call user data, free the storage used (as in example program **svcrcv**).

Instead of accepting an incoming call, you can reject it, using the **x25_call_clear** subroutine to clear it.

Transferring and Acknowledging Data

The following information describes the processes involved in transferring and acknowledging data.

Sending Data

Either the called or the calling application can send data when:

- On a permanent virtual circuit (PVC), the PVC has been allocated.
- On a switched virtual circuit (SVC), a call has been made, received, and accepted.

To send data (as in example program **svcxmit**):

1. Ensure that any data sent previously with the D-bit set to a value of 1 has been acknowledged. Otherwise, this **x25_send** subroutine will fail.
2. Assign to the `data` variable in **cb_data_struct** a pointer to the data you want to send.
3. Assign to the `data_len` variable in **cb_data_struct** the length of the data.
4. Invoke the **x25_send** subroutine, passing two parameters: the connection identifier and a pointer to **cb_data_struct**.

Asking for Receiver Acknowledgment of Data Sent

To ask for the receiver to acknowledge the data, set the flags to **X25FLG_DBIT** in **cb_data_struct**, before using the **x25_send** subroutine (as in example program **svcxmit**). The application must then wait for and receive the `X25_DATA_ACK` message that is sent back.

> **Note:** To allow the use of the D-bit, it should also be set on the **x25_call** subroutine (as in example program **svcxmit**) or the **x25_call_accept** subroutine.

Long Messages

If the data length is greater than the packet size, the API automatically splits the data into packets which it sends separately. It sets on the M-bit in each packet to indicate that there is more data. Only the final packet has the D-bit set and only one acknowledgment is expected.

To allow better recovery in the event of a transmission failure, avoid sending data longer than the packet size. Specify as large a packet size as possible in the maximum transmit packet size attribute (for an SVC) or PVC maximum transmit packet size. Otherwise specify as large a packet size as possible in the **psiz_cld** or **psiz_clg** field in the **cb_fac_struct**. If necessary, split up the data yourself in the application, if you want to receive an acknowledgment for each packet, and thus maintain data integrity. Otherwise, if one piece of the data does not arrive, all of the data may need to be sent again.

Receiving Data

To receive data that has arrived for a particular call (as in example program **svcrcv**):

1. Ensure that you have acknowledged any data received previously with the D-bit set to a value of 1. Otherwise this **x25_receive** subroutine will return **X25NODATA**.
2. Invoke the **x25_receive** subroutine subroutine, passing the address of the connection identifier and the address of the message structure (**cb_msg_struct**) as parameters.
3. The **x25_receive** subroutine receives a complete packet sequence. That is to say, if a long message was split up when it was sent, the X.25 API attempts to rebuild it before notifying the application that there is a message waiting. If any packet (other than data or interrupt) arrives before the sequence is completed, the attempt to rebuild is *either* abandoned and the sequence made available to the application up to its current position, *or* the incoming packet is made available to the application ahead of the as-yet-unfinished sequence.
4. On return from the **x25_receive** subroutine, the message structure (**cb_msg_struct**) includes the **msg_type**, which indicates the type of message. In this case it is `X25_DATA`, indicating that the message is available in the **cb_data_struct** control block.
5. The counter that indicated the waiting message is decremented when the message is received.

To receive data from any call that is currently connected, assign a value of 0 to the *conn_id* parameter and invoke the **x25_receive** subroutine, passing the address of *conn_id* as a parameter. On return from the **x25_receive** subroutine, the *conn_id* parameter contains the connection identifier of the call whose data was returned by **x25_receive** subroutine.

Acknowledging Data Packets

For each data packet that was sent with the D-bit set to 1, invoke the **x25_ack** subroutine to confirm that it arrived (as in example program **svcrcv**).

The application should ensure that the acknowledgment is given as soon as possible after receiving a message with the D-bit set to 1.

Clearing, Resetting, and Interrupting Calls

The following information describes the processes involved in clearing, resetting, and interrupting calls.

Clearing a Call

Clearing removes the call from the network. You can send a clear-request message to reject a call, after receiving fast-select data, to terminate a call, or to clear a call.

If you clear a call without ever accepting it, you are, in effect, rejecting it.

If it is a fast-select call, the fast-select data is in the incoming-call packet. You can clear the call immediately after receiving this or you can receive further messages on the call.

Clearing is the normal way of terminating a call. Either the caller or the called application can clear a call. To clear a call:

1. Optionally, assign any data you want to send to the *user_data* field in the **cb_clear_struct** control block, and set the user-data flag.
2. Optionally, assign a cause code and a diagnostic code to the appropriate fields in the **cb_clear_struct** control block, and set the appropriate flags.
3. Invoke the **x25_call_clear**, passing the connection identifier and a pointer to **cb_clear_struct** as parameters. The third parameter can be used for return data. If you do not need this, set the third parameter to null.

Example program **svcxmit** shows how a call is cleared.

> **Note:** Example program **svcrcv** could have cleared the call after receiving the data; the **svcxmit** program is therefore prepared for the call to be cleared by the other application. A call does not have to be cleared by the application that made it.

Resetting a Call

A reset flushes any data being sent from the network at the time of the reset. To reset a call (as in example program **pvcxmit**):

1. Optionally, assign a cause code and a diagnostic code to the appropriate fields in the **cb_res_struct** control block and set the appropriate flags.
2. Invoke the **x25_reset** subroutine, passing it the connection identifier and a pointer to the **cb_res_struct** control block.
3. Wait for and receive the reset-confirmation message.

When an application receives a message of `X25_RESET_INDICATION`, it must send a reset-confirmation message immediately by invoking the **x25_reset_confirm** subroutine (as in example program **pvcrcv**).

Interrupting a Call

An interrupt is placed at the beginning of the queue of incoming messages. To send an interrupt:

1. Assign the connection identifier to the *conn_id* parameter.
2. Invoke the **x25_interrupt** subroutine, passing it the *conn_id* parameter and a pointer to the **cb_int_data_struct** control block.
3. Wait for and receive the interrupt-confirmation message. (Using this X.25 API, the interrupt-confirmation message is sent automatically.)

X.25 Example Programs

To help you learn how to use the X.25 subroutines there are two pairs of example programs. One pair demonstrates the use of a switched virtual circuit, and the other the use of a permanent virtual circuit. The example programs are:

- X.25 Example Program svcxmit: Make a Call Using an SVC
- X.25 Example Program svcrcv: Receive a Call Using an SVC
- X.25 Example Program pvcxmit: Send Data Using a PVC
- X.25 Example Program pvcrcv: Receive Data Using a PVC

General information on using the examples includes:

- Preparing, Compiling, and Running the Example Programs
- Using the Example Code

Preparing, Compiling, and Running the Example Programs

The example programs (**svcxmit.c**, **svcrcv.c**, **pvcxmit.c**, and **pvcrcv.c**) are in the samples directory **/usr/lpp/sx25/samples**.

Each of the example programs has variables to which values are assigned at the start. These include **CALLING_ADDR**, **CALLED_ADDR**, **LINK_NAME**, and **LOG_CHAN_NUM**. These must be set to appropriate values for your setup before you can run the programs.

To compile the example programs, enter:

```
cd /usr/lpp/bosext2/x25app/samples
cc svcxmit.c -lx25s -o svcxmit
cc svcrcv.c -lx25s -o svcrcv
cc pvcxmit.c -lx25s -o pvcxmit
cc pvcrcv.c -lx25s -o pvcrcv
```

This creates the executable files **svcxmit**, **svcrcv**, **pvcxmit**, and **pvcrcv**.

To run a program, type the name of the executable file at the shell prompt. Run them in pairs: **svcxmit** talks to **svcrcv**, and **pvcxmit** talks to **pvcrcv**.

> **Note:** You cannot run the PVC programs unless your network allows the use of permanent virtual circuits.

Using the Example Code

The example programs are for demonstration purposes only. When creating your own programs, you may find it useful to copy parts of the code from the examples. Be aware that the examples do not, in most cases, check the return codes from the subroutines. When you invoke an X.25 subroutine in a production program, you should assign the return value into a variable, as in the following:

```
rc = x25_...(...);
```

Then test the value of the return code.

If you do not want to write your own programs, use the **xtalk** command to communicate with other users.

X.25 Example Program svcxmit: Make a Call Using an SVC

This example program uses a switched virtual circuit (SVC) to make a call and transmit data, as follows. Example program **svcrcv** is designed to receive the data sent by this program.

Program Description

To use the X.25 program:

1. Initialize the API for the port specified by **LINK_NAME** (**x25_init**).
2. If initialization failed, the program displays a message. Exit from the program.
3. Get a counter (**x25_ctr_get**).
4. Make a call from this address specified by the **CALLING_ADDR** flag to the address specified by the **CALLED_ADDR** flag, enabling D-bit acknowledgment (**x25_call**).
5. Wait for a call-clear or call-connected message (**x25_ctr_wait**).
6. Receive the message (**x25_receive**).
7. If the message is call-connected:
   a. Send data (**x25_send**), without the D-bit set.
   b. Send data (**x25_send**), with the D-bit set.
   c. Wait for (**x25_ctr_wait**) and receive (**x25_receive**) acknowledgment of the data sent with the D-bit set.
   d. Clear the call (**x25_call_clear**).
8. If the call was cleared by the remote DTE (the other user), the program displays a message.
9. Remove the counter (**x25_ctr_remove**).
10. Terminate the API (**x25_term**).

Example Program svcxmit

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <NLchar.h>
#include <x25sdefs.h>
#define LINK_NAME    "x25s0"           /* Name of X.25 port.          */
#define CALLING_ADDR "54321"           /* Calling Network User Address */
#define CALLED_ADDR  "1234502"         /* Called  Network User Address */

#define SAMPLE_NAME "IBMSAMP"         /* A name in the X.25 routing list.*/

#define INFO        "Hello World"
#define INFO2       "Goodbye Everyone"

/*****************************************************************************/
/* Function     main                                                         */
/* Description  This program is designed to demonstrate usage of the X.25    */
/*              API. It makes a call, transmits some data, and then clears   */
/*              the call.                                                    */
/*              Example program svcrcv is designed to receive the data sent */
/*              by this program.                                             */
/*              Note that, in a production program, you should check the     */
/*              return code from each subroutine call and take appropriate   */
/*              action.                                                      */
/* Return       0 if successful                                              */
/*              1 if error occurs                                            */
/*****************************************************************************/

int main(
  int argc,
  char *argv[])
{
  int conn_id;        /* Connection identifier,                             */
                      /*  to identify the call once it is made.              */
  int ctr_id;         /* Counter identifier to be associated with the call.  */
  int rc;             /* Used for return codes.                              */
  int ctr_num = 1;    /* The number of counters in counter array.            */
 /*****************************************************************************/
 /* The following structures are defined in the x25sdefs.h file.             */
 /*****************************************************************************/

struct cb_msg_struct cb_msg;
struct cb_link_name_struct cb_link_name;
struct ctr_array_struct ctr_array[1];
struct cb_call_struct cb_call;
struct cb_clear_struct cb_clear;
struct cb_data_struct cb_data;

  /*****************************************************************************/
  /* Initialize the API for access to a link.                                  */
  /*****************************************************************************/

  cb_link_name.flags = X25FLG_LINK_NAME;
  cb_link_name.link_name = LINK_NAME;
  rc = x25_init (&cb_link_name);
```

```
if (rc < 0)
{
  (void)printf("%s: x25_init failed : x25_errno = %d errno = %d\n",
              argv[0],x25_errno,errno);
  return(1);
}
else
{

  /*********************************************************************/
  /* Get a counter to be used to notify us of incoming messages.       */
  /*********************************************************************/
   ctr_id = x25_ctr_get();

  /*********************************************************************/
  /* Set the flags in the cb_call structure to indicate which fields   *
  /* have been filled in. The fields which this program sets           */
  /* are the calling and called addresses, and the link on which to call. */
  /* The D-bit field must also be set, as there will be a data packet  */
  /* sent later which sets the D-bit.                                  */
  /*********************************************************************/
 cb_call.flags = X25FLG_LINK_NAME;      /* Set flag for using linkname.  */
 cb_call.link_name = LINK_NAME;

 cb_call.flags |= X25FLG_CALLING_ADDR;  /* Set flag for calling address. */
 cb_call.calling_addr = CALLING_ADDR;

 cb_call.flags |= X25FLG_CALLED_ADDR;   /* Set flag for called address.  */
 cb_call.called_addr = CALLED_ADDR;

  cb_call.flags |= X25FLG_D_BIT;         /* Set flag for D-bit.         */
  /* Now that cb_call structure has been set up, make the call.        */
  /* The return code is the connection identifier, which will be used to  */
  /* refer to this call later.                                         */

  conn_id = x25_call(&cb_call,ctr_id);
  if (conn_id == -1)
  {
    (void)printf("%s: x25_call failed : x25_errno = %d errno = %d\n",
                argv[0],x25_errno,errno);
    return(1);
  }
  else
    (void)printf("%s: Placed outgoing call\n",argv[0]);

  /*********************************************************************/
  /* After making the call, prepare for either a call-connected or a   */
  /* clear-indication message to arrive:                               */
  /* wait for the counter value to change indicating an incoming message. */
  /* (If there were more than one counter in the array, you would have to */
  /* test the counter identifier to see which one had been incremented.   */
  /* In this case there is only one, so we do not have to do this.)    */
  /*********************************************************************/

  ctr_array[0].flags = X25FLG_CTR_ID;
  ctr_array[0].flags |= X25FLG_CTR_VALUE;
  ctr_array[0].ctr_id = ctr_id;
```

```
ctr_array[0].ctr_value = 0;

(void)x25_ctr_wait(ctr_num,ctr_array);

/* Receive the call-clear or call-connected packet.                    */
(void)x25_receive(&conn_id,&cb_msg);

/*********************************************************************/
/* If the incoming message shows that the call has been connected,   */
/* send some data.                                                   */
/*********************************************************************/

if (cb_msg.msg_type == X25_CALL_CONNECTED)
{
  cb_data.flags = X25FLG_DATA;
  cb_data.data_len = strlen(INFO);
  cb_data.data = INFO;
  (void)x25_send(conn_id,&cb_data);
  (void)printf("%s: Data sent\n",argv[0]);

  /*******************************************************************/
  /* Send some more data but this time with the D bit set. This      */
  /* requires the receiver to send an acknowledgement to this data,  */
  /* so we have to wait for the acknowledgment to arrive.            */
  /*******************************************************************/

  cb_data.flags = X25FLG_DATA;
  cb_data.flags |= X25FLG_D_BIT;
  cb_data.data_len = strlen(INFO2);
  cb_data.data = INFO2;
  (void)x25_send(conn_id,&cb_data);
  (void)printf("%s: Data sent\n",argv[0]);

  /* Wait for and receive acknowledgement                              */
  (void)x25_ctr_wait(ctr_num,ctr_array);
  (void)x25_receive(&conn_id,&cb_msg);

  if (cb_msg.msg_type == X25_DATA_ACK)
    (void)printf("%s: Data has been acknowledged.\n",argv[0]);
  else
    (void)printf("%s: Unexpected packet received.\n",argv[0]);

  /*******************************************************************/
  /* Clear the call now that transmission is completed.              */
  /*******************************************************************/

  cb_clear.flags = X25FLG_CAUSE;
  cb_clear.flags |= X25FLG_DIAGNOSTIC;
  cb_clear.cause = 0;            /* The CCITT code for DTE-originated   */
  cb_clear.diagnostic = 0;       /* No further information              */

  (void)printf("%s: Clearing the call.",argv[0]);

  /* The x25_call_clear function can return information from the clear  */
  /* confirmation packet. However, this isn't required here, so set the */
  /* third parameter to NULL.                                          */
```

```
      (void)x25_call_clear(conn_id,&cb_clear,(struct cb_msg_struct *)NULL);
    }
    /*************************************************************************/
    /* If the message received was a clear-indication,                       */
    /* print out a message before terminating the program.                   */
    /*************************************************************************/
    else if (cb_msg.msg_type == X25_CLEAR_INDICATION)
    {
      (void)printf("%s: Call cleared. Cause = 0x%02x Diagnostic = 0x%02x\n",
        argv[0], cb_msg.msg_point.cb_clear->cause,
        cb_msg.msg_point.cb_clear->diagnostic);
    }

    /*************************************************************************/
    /* Finally, tidy up by removing the counter and terminating the API.     */
    /*************************************************************************/

    (void)x25_ctr_remove(ctr_id);
    (void)x25_term(&cb_link_name);
  }
  return(0);
}
```

X.25 Example Program svcrcv: Receive a Call Using an SVC

This program receives a call over a switched virtual circuit (SVC), accepts it, and then prints any data received. Example program **svcxmit** is designed to send the data received by this program.

Program Description

The X.25 program uses the following steps:

1. Initialize the API for the port specified by **LINK_NAME** (**x25_init**).
2. If initialization failed, the program displays a message and exits.
3. Get a counter for listening for incoming calls (**x25_ctr_get**).
4. Start listening for incoming calls (**x25_listen**).
5. Wait for an incoming call (**x25_ctr_wait**).
6. Receive the incoming call (**x25_receive**).
7. Get a counter for handling this call (**x25_ctr_get**).
8. Accept the call (**x25_call_accept**).
9. Free any memory allocated by the API to **cb_msg_struct**.
10. Wait for a message (**x25_ctr_wait**).
11. Receive message (**x25_receive**).
12. If the message is data:
    a. Acknowledge if the D-bit is set (**x25_ack**).
    b. Display the data on the screen.
    c. Free any memory allocated to **cb_msg_struct** by the API.
13. If the message is a clear indication:
    a. Display a message to say the call has been cleared.
    b. Remove the counter (**x25_ctr_remove**).
    c. Stop listening for calls (**x25_deafen**).
    d. Terminate the API for the port (**x25_term**).

14. If the message is a reset indication, send a reset confirmation (**x25_reset_confirm**).
15. For any other message type, do nothing.

Example Program svcrcv

```
/* X.25 Example Program svcrcv. */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <NLchar.h>
#include <x25sdefs.h>

#define LINK_NAME   "x25s0"            /* Name of X.25 port.               */
#define SAMPLE_NAME "IBMSAMP"          /* A name in the X.25 routing list.  */

/*****************************************************************************/
/* Function     main                                                         */
/* Description  This program is designed to demonstrate usage of the X.25    */
/*              API. It waits for an incoming call, accepts it, and then     */
/*              prints any data received.                                    */
/*              Example program svcxmit is designed to send the data         */
/*              received by this program.                                    */
/*              Note that, in a production program, you should check the     */
/*              return code from each subroutine call and take appropriate   */
/*              action.                                                      */
/* Returns      0 if successful                                              */
/*              1 if error                                                   */
/*****************************************************************************/

int main(
  int argc,
  char *argv[])
{

  /***************************************************************************/
  /* The following structures are defined in the x25sdefs.h file.            */
  /***************************************************************************/
  struct cb_call_struct cb_call;
  struct ctr_array_struct ctr_array[1];       /* This program waits for only */
                                               /* one counter at a time.     */
  struct cb_msg_struct cb_msg;
  struct cb_link_name_struct cb_link_name;


  NLchar name[8];      /* 1 longer than SAMPLE_NAME for NULL terminator.     */
  int listen_id;       /* Listen identifier for x25_receive.                 */
  int conn_id;         /* Connection identifier to identify the call after   */
                       /* receiving it.                                      */
  int listen_ctr_id;   /* Counter identifier to associate with incoming calls*/
  int call_ctr_id;     /* Counter identifier to associate with accepted call */
  int ctr_num;         /* Number of entries in ctr_array.                    */
  int rc;              /* Return code                                        */

  /***************************************************************************/
  /* Initialize the API for access to a link.                                */
  /***************************************************************************/
```

```
cb_link_name.flags = X25FLG_LINK_NAME;
cb_link_name.link_name = LINK_NAME;

rc = x25_init(&cb_link_name);
if (rc < 0)
{
  (void)printf("%s: x25_init failed : x25_errno = %d errno = %d\n",
               argv[0],x25_errno,errno);
  return(1);
}
else
{

  /*************************************************************************/
  /* Prepare to receive incoming calls:                                    */
  /* 1. Get a counter to be used to notify us of incoming calls.           */
  /* 2. Listen for calls that satisfy the criteria specified by a name in */
  /* the routing list.                                                     */
  /*************************************************************************/

  listen_ctr_id = x25_ctr_get();          /* Get a counter.               */

  (void)NCdecstr(SAMPLE_NAME,name,8);     /* Convert to NLchar.           */
  listen_id = x25_listen(name,listen_ctr_id);
  if (listen_id < 0)
  {
    (void)printf("%s: x25_listen failed : x25_errno = %d errno = %d\n",
      argv[0],x25_errno,errno);
    return(1);
  }
  else
    (void)printf("%s: Awaiting incoming call...\n",argv[0]);

  /*************************************************************************/
  /* Wait for an incoming call. The x25_ctr_wait subroutine returns        */
  /* when a message arrives.                                               */
  /*************************************************************************/

  ctr_num = 1;
  ctr_array[0].flags = X25FLG_CTR_ID;
  ctr_array[0].flags |= X25FLG_CTR_VALUE;
  ctr_array[0].ctr_id = listen_ctr_id;
  ctr_array[0].ctr_value = 0;
  rc = x25_ctr_wait(ctr_num,ctr_array);

  /*************************************************************************/
  /* Receive an incoming call.                                             */
  /* In this example, we can assume that the message that has arrived      */
  /* (causing the counter to be incremented and x25_ctr_wait to return)    */
  /* is an incoming-call message.  Therefore we assign the listen          */
  /* identifier to the conn_id parameter before invoking x25_receive and   */
  /* we do not check the return code.                                      */

 /* On return, conn_id is set to the connection identifier for this call.*/
 /*************************************************************************/
```

```
conn_id = listen_id;
(void)x25_receive(&conn_id,&cb_msg);

(void)printf("%s: Incoming call received\n",argv[0]);

/*************************************************************************/
/* Get a new counter for handling data from this call before           */
/* accepting the call.                                                   */
/* No additional information needs to be put into the call-accept       */
/* packet, so the flags field is set to zero.                            */
/*************************************************************************/
call_ctr_id = x25_ctr_get();
cb_call.flags = 0;
(void)x25_call_accept(conn_id,&cb_call,call_ctr_id);
(void)printf("%s: Call accepted.\n",argv[0]);
/*************************************************************************/
/* x25_receive allocates storage to return information. Although there  */
/* are no storage constraints in this application, the allocated        */
/* storage is freed once the information is no longer needed.           */
/*************************************************************************/
if (cb_msg.msg_point.cb_call != NULL)
{
  cb_msg.msg_point.cb_call -> flags = 0;
  if (cb_msg.msg_point.cb_call->link_name != NULL)
    free(cb_msg.msg_point.cb_call->link_name);

  if (cb_msg.msg_point.cb_call->calling_addr != NULL)
    free(cb_msg.msg_point.cb_call->calling_addr);

  if (cb_msg.msg_point.cb_call->called_addr != NULL)
    free(cb_msg.msg_point.cb_call->called_addr);

  if (cb_msg.msg_point.cb_call->user_data != NULL)
    free(cb_msg.msg_point.cb_call->user_data);

  free(cb_msg.msg_point.cb_call);
}
/*************************************************************************/
/* The call has now been received and accepted. Now wait for the data.  */
/*************************************************************************/
do
{

   /*************************************************************************/
   /* Wait for counter to indicate that data is waiting to be received.  */
   /*************************************************************************/
   ctr_num = 1;
   ctr_array[0].flags = X25FLG_CTR_ID;
   ctr_array[0].flags |= X25FLG_CTR_VALUE;
   ctr_array[0].ctr_id = call_ctr_id;
   ctr_array[0].ctr_value = 0;
   (void)x25_ctr_wait(ctr_num,ctr_array);

   /*************************************************************************/
   /* Receive the message that is now ready.  The types of message that  */
   /* the program can handle are data, clear-indication, and             */
   /* reset-indication; other message types are ignored.                 */
   /*************************************************************************/
   (void)x25_receive(&conn_id,&cb_msg);
```

```
      switch (cb_msg.msg_type)
      {
      case X25_DATA:

        /*********************************************************************/
        /* Acknowledge the data if the D-bit (delivery confirmation) is set.*/
        /*********************************************************************/
        if ((cb_msg.msg_point.cb_data->flags) & X25FLG_D_BIT)
            (void)x25_ack(conn_id);
       /*********************************************************************/
       /* Print the received data. Assume it is a normal string.           */
       /*********************************************************************/
       if ((cb_msg.msg_point.cb_data -> flags) & X25FLG_DATA)
       {
         (void)printf("%s: Incoming Data : ",argv[0]);
         (void)printf("%s\n",cb_msg.msg_point.cb_data->data);
         free(cb_msg.msg_point.cb_data->data);    /* Free memory allocated */
         free(cb_msg.msg_point.cb_data);
       }
       break;

      case X25_CLEAR_INDICATION:
        /*********************************************************************/
        /* When the call has been cleared, do the tidying up:               */
        /*   Remove the counters.                                           */
        /*   Stop listening for calls.                                      */
        /*   Terminate the API.                                             */
        /*********************************************************************/
        (void)printf("%s: Call cleared. Cause = 0x%02x Diagnostic = 0x%02x\n",
          argv[0], cb_msg.msg_point.cb_clear->cause,
          cb_msg.msg_point.cb_clear->diagnostic);
        (void)x25_ctr_remove(call_ctr_id);
        (void)x25_ctr_remove(listen_ctr_id);
        (void)x25_deafen(listen_id);
        (void)x25_term(&cb_link_name);
        break;
      case X25_RESET_INDICATION:
        /*********************************************************************/
        /* Respond to the arrival of a reset-indication message, by sending */
        /* a reset-confirmation message.                                     */
        /*********************************************************************/
        (void)x25_reset_confirm(conn_id);
        break;
      default:
        /* Ignore packet types other than data, clear-indication, and     */
        /* reset-indication.                                              */
         break;
      }
    } while (cb_msg.msg_type != X25_CLEAR_INDICATION);
  }
  return(0);
}
```

X.25 Example Program pvcxmit: Send Data Using a PVC

This program uses a permanent virtual circuit (PVC) to make a call. It allocates the circuit, sends some data and then sends a reset. After receiving the reset-confirmation, the program sends some more data. Example Program **pvcrcv** is designed to receive the data sent by this program.

Program Description

The following steps outline the **pvcxmit** program:

1. Initialize the API for the port specified by the **LINK_NAME** value (**x25_init**).
2. If initialization failed, the program displays a message and exits.
3. Get a counter to be used to wait for incoming messages (**x25_ctr_get**).
4. Allocate a PVC to the port, using the logical channel number specified by the **LOG_CHAN_NUM** value (**x25_pvc_alloc**).
5. If PVC allocation failed, the program displays a message and exits.
6. Send some data (**x25_send**).
7. Send a reset (**x25_reset**).
8. Wait for the reset-confirmation message (**x25_ctr_wait**).
9. Receive the reset-confirmation message (**x25_receive**).
10. Send some more data (**x25_send**).
11. Send the end-of-transmission indicator specified by the **END_OF_TRANS** value (**x25_send**).
12. Free the permanent virtual circuit (**x25_pvc_free**).
13. Remove the counter (**x25_ctr_remove**).
14. Terminate the API for the port (**x25_term**).

Example Program pvcxmit

```
/* X.25 Example Program pvcxmit. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <x25sdefs.h>

#define LINK_NAME    "x25s0"            /* Name of X.25 port.               */
#define LOG_CHAN_NUM  (1)               /* PVC logical channel number.      */

#define INFO  "Hello World"             /* Data to be sent.                 */
#define INFO2 "Goodbye Everyone"        /* More data to be sent.            */
#define END_OF_TRANS "EOP"              /* End-of-transmission indicator:   */
                                        /* must be the same as in pvcrcv.   */

/*****************************************************************************/
/* Function     main                                                         */
/* Description  This program is designed to demonstrate usage of the X.25    */
/*              API.                                                         */
/*              It allocates a permanent virtual circuit, sends some data    */
/*              and then sends a reset.  After receiving the                 */
/*              reset-confirmation, the program sends some more data.        */
/*              Example Program pvcrcv is designed to receive the data sent  */
/*              by this program.                                             */
/*              Note that, in a production program, you should check the     */
/*              return code from each subroutine call and take appropriate   */
/*              action.                                                      */
/* Return       0 if successful                                              */
/*              1 otherwise                                                  */
/*****************************************************************************/
```

```
int main(
  int argc,
  char *argv[])
{

  /*************************************************************************/
  /* The following structures are defined in the x25sdefs.h file.          */
  /*************************************************************************/
  struct ctr_array_struct ctr_array[1];         /* One counter in the array. */
  struct cb_msg_struct cb_msg;
  struct cb_pvc_alloc_struct cb_pvc;
  struct cb_res_struct cb_res;
  struct cb_link_name_struct cb_link_name;
  struct cb_data_struct cb_data;

  int conn_id;          /* Connection identifier to associate with this link.*/
  int ctr_id;           /* Counter identifier for this link.                 */
  int ctr_num = 1;      /* Number of counters in the counter array.          */
  int rc;               /* Return codes from various subroutines.            */

  /*************************************************************************/
  /* Initialize the API for access to a link.                              */
  /*************************************************************************/
  cb_link_name.flags = X25FLG_LINK_NAME;
  cb_link_name.link_name = LINK_NAME;
  rc = x25_init(&cb_link_name);
  if (rc < 0)
  {
    (void)printf("%s: x25_init failed : x25_errno = %d errno = %d\n",
                 argv[0],x25_errno,errno);
    return(1);
  }
  else
  {

    /*************************************************************************/
    /* Get a counter to be used to notify us of incoming messages.           */
    /*************************************************************************/
    ctr_id = x25_ctr_get();

    /*************************************************************************/
    /* Set up flags to show that a link and a channel number are supplied.   */
    /* Then allocate the permanent virtual circuit for this application.     */
    /*************************************************************************/
    cb_pvc.flags = X25FLG_LINK_NAME | X25FLG_LCN;
    cb_pvc.link_name = LINK_NAME;
    cb_pvc.lcn = LOG_CHAN_NUM;

    conn_id = x25_pvc_alloc(&cb_pvc,ctr_id);

    if (conn_id < 0)
    {
      (void)printf("%s: x25_pvc_alloc failed : x25_errno = %d errno = %d\n",
                   argv[0],x25_errno,errno);
      return(1);
```

```
}
else
{

  /*********************************************************************/
  /* Now the PVC is available, send some data.                          */
  /*********************************************************************/

  cb_data.flags = X25FLG_DATA;
  cb_data.data_len = strlen(INFO);
  cb_data.data = INFO;
  (void)printf("%s: Sending some data...",argv[0]);
  (void)x25_send(conn_id,&cb_data);

  /*********************************************************************/
  /* Send a reset.                                                     */
  /*********************************************************************/
  (void)printf("%s: Resetting the circuit...",argv[0]);
  (void)x25_reset(conn_id,&cb_res);

  /*********************************************************************/
  /* After sending a reset packet, you must wait for the reset confirm  */
  /* to arrive.                                                        */
  /*********************************************************************/
  ctr_array[0].flags = X25FLG_CTR_ID;
  ctr_array[0].flags |= X25FLG_CTR_VALUE;
  ctr_array[0].ctr_id = ctr_id;
  (void)x25_ctr_wait(ctr_num,ctr_array);

 /*********************************************************************/
  /* There is now a message ready to be received.  If it is anything    */
  /* other than the expected reset-confirmation, we:                   */
  /*   free the permanent virtual circuit                              */
  /*   remove the counter.                                             */
  /*   terminate the API.                                              */
  /*********************************************************************/
  (void)x25_receive(&conn_id,&cb_msg);

  if (cb_msg.msg_type != X25_RESET_CONFIRM)
  {
    (void)printf("%s: Did not receive expected reset confirm",argv[0]);
    (void)x25_pvc_free(conn_id);
    (void)x25_ctr_remove(ctr_id);
    (void)x25_term(&cb_link_name);
    return(1);
  }

  (void)printf("%s: Received reset confirm...",argv[0]);
 /*********************************************************************/
 /* Now send some more data                                           */
 /* The last block of data to be sent is the end-of-transmission      */
 /* indicator specified by END_OF_TRANS.  This is understood by the   */
 /* PVC receiver example program, pvcrcv.                             */
 /*********************************************************************/
 cb_data.data_len = strlen(INFO2);
 cb_data.data = INFO2;
```

```
    (void)printf("%s: Sending some data...",argv[0]);
      (void)x25_send(conn_id,&cb_data);

      (void)printf("%s: Sending last block of data...",argv[0]);
      cb_data.data_len = strlen(END_OF_TRANS);
      cb_data.data = END_OF_TRANS;
      (void)x25_send(conn_id,&cb_data);

      /*********************************************************************/
      /* Free up any resources allocated during the program before ending:  */
      /*   free the permanent virtual circuit                              */
      /*   remove the counter.                                             */
      /*   terminate the API.                                              */
      /*********************************************************************/
      (void)x25_pvc_free(conn_id);
      (void)x25_ctr_remove(ctr_id);
      (void)x25_term(&cb_link_name);
    }
  }
  return(0);
}
```

X.25 Example Program pvcrcv: Receive Data Using a PVC

This program uses a permanent virtual circuit (PVC) to make a call. It allocates the circuit, receives data, and is prepared to handle a reset by sending a reset-confirmation packet. Example program **pvcxmit** is designed to send the data received by this program.

Program Description

The X.25 program uses the following steps:

1. Initialize the API for the port specified by the **LINK_NAME** value (**x25_init**).
2. If initialization failed, the program displays a message and exits.
3. Get a counter to be used to wait for incoming messages (**x25_ctr_get**).
4. Allocate a PVC to the port, using the logical channel number specified by the **LOG_CHAN_NUM** value (**x25_pvc_alloc**).
5. If PVC allocation failed, the program displays a message and exits.
6. Wait for an incoming message (**x25_ctr_wait**).
7. Receive the incoming message (**x25_receive**).
8. Test the **msg_type** in **cb_msg_struct**:
   a. If the incoming message is a reset indication, send a reset confirmation (**x25_reset_confirm**).
   b. If the incoming message is data, display it on the screen. (If it is the end-of-transmission indicator specified in the **END_OF_TRANS** value, print a message saying that transmission has ended.) Free the storage allocated to the structure **cb_msg_struct**.
9. Free the permanent virtual circuit (**x25_pvc_free**).
10. Remove the counter (**x25_ctr_remove**).
11. Terminate the API for port x25s1 (**x25_term**).

Example Program pvcrcv

```
/* X.25 Example Program pvcrcv. */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <x25sdefs.h>

#define LINK_NAME    "x25s0"           /* Name of X.25 port.               */
#define LOG_CHAN_NUM  (1)              /* PVC logical channel number.      */
#define END_OF_TRANS "EOP"             /* End-of-transmission indicator;   */
                                       /* must be the same as in pvcxmit.  */

/*****************************************************************************/
/* Function     main                                                         */
/* Description  This program is designed to demonstrate usage of the X.25    */
/*              API.                                                         */
/*              It allocates a permanent virtual circuit, receives data and  */
/*              is prepared to handle a reset, by sending a                  */
/*              reset-confirmation.                                          */
/*              Example Program pvcxmit is designed to send the data         */
/*              received by this program.                                    */
/*              Note that, in a production program, you should check the     */
/*              return code from each subroutine call and take appropriate   */
/*              action.                                                      */
/* Return       0 if successful                                              */
/*              1 otherwise                                                  */
/*****************************************************************************/
int main(
  int argc,
  char *argv[])
{

  /***************************************************************************/
  /* The following structures are defined in the x25sdefs.h file.            */
  /***************************************************************************/
  struct ctr_array_struct ctr_array[1];         /* One counter in the array. */
  struct cb_msg_struct cb_msg;
  struct cb_pvc_alloc_struct cb_pvc;
  struct cb_link_name_struct cb_link_name;

  int conn_id;          /* Connection identifier to associate with this link.*/
  int ctr_id;           /* Counter identifier for this link.                 */
  int rc;               /* Return codes from various subroutines.            */
  int ctr_num = 1;      /* Number of counters in the counter array.          */
  int end_tx = 0;       /* Whether end of transmission has been reached.     */

  /***************************************************************************/
  /*  Initialize the API for access to a link.                               */
  /***************************************************************************/

  cb_link_name.flags = X25FLG_LINK_NAME;
  cb_link_name.link_name = LINK_NAME;
  rc = x25_init(&cb_link_name);
  if (rc < 0)
  {
   (void)printf("%s: x25_init failed : x25_errno = %d errno = %d\n",
               argv[0],x25_errno,errno);
    return(1);
```

```
}
else
{

  /*********************************************************************/
  /* Get a counter to be used to notify us of incoming messages.       */
  /*********************************************************************/
  ctr_id = x25_ctr_get();

  /*********************************************************************/
  /* Set up flags to show that a link and a channel number are supplied.  */
  /* Then allocate the permanent virtual circuit for this application.    */
  /*********************************************************************/
  cb_pvc.flags = X25FLG_LINK_NAME | X25FLG_LCN;
  cb_pvc.link_name = LINK_NAME;
  cb_pvc.lcn = LOG_CHAN_NUM;


  conn_id = x25_pvc_alloc(&cb_pvc,ctr_id);

 if (conn_id < 0)
 {
   (void)printf("%s: x25_pvc_alloc failed : x25_errno = %d errno = %d\n",
              argv[0],x25_errno,errno);
   return(1);
 }
 else
 {

    /*********************************************************************/
    /* The PVC link has now been set up and data can be received.       */
    /* Wait for any message to arrive for this application              */
    /*********************************************************************/

    ctr_array[0].flags = X25FLG_CTR_ID;
    ctr_array[0].flags |= X25FLG_CTR_VALUE;
    ctr_array[0].ctr_id = ctr_id;
    ctr_array[0].ctr_value = 0;
    do
    {
      (void)x25_ctr_wait(ctr_num,ctr_array);

      /*********************************************************************/
      /* Receive the message                                               */
      /*********************************************************************/
      (void)x25_receive(&conn_id,&cb_msg);

      /*********************************************************************/
      /* If a reset-indication message is received, we must                */
      /* send a reset-confirmation message as soon as possible.           */
      /*********************************************************************/
      if (cb_msg.msg_type == X25_RESET_INDICATION)
      {
        (void)printf("%s: Received reset indication...",argv[0]);
        (void)x25_reset_confirm(conn_id);
      }
```

```
        /* If data is received, we display it on the screen, unless it is    */
        /* end-of-transmission indicator specified by END_OF_TRANS.          */
        else if (cb_msg.msg_type == X25_DATA)
        {
          (void)printf("%s: Incoming Data : ",argv[0]);
          (void)printf("%s\n",cb_msg.msg_point.cb_data->data);
          if (strcmp(cb_msg.msg_point.cb_data->data,END_OF_TRANS) != 0)
          {
            (void)printf("%s",cb_msg.msg_point.cb_data->data);
            (void)printf("\n");
          }
          else
          {
            (void)printf("%s: End of transmission received",argv[0]);
            end_tx = 1;
          }
          /*****************************************************************/
          /* The X.25 API allocates memory for information to be returned.  */
          /* Although there are no memory constraints in this application,  */
          /* the space is freed when the information has been displayed.    */
          /*****************************************************************/
         free((char *)cb_msg.msg_point.cb_data->data);
         free((char *)cb_msg.msg_point.cb_data);
       }
       else
       {
         (void)printf("%s: Unexpected packet received",argv[0]);
       }
      }  while (end_tx == 0);

      /*************************************************************************/
      /* Free up any resources allocated during the program before ending:  */
      /*   free the permanent virtual circuit                               */
      /*   remove the counter                                               */
      /*   terminate the API.                                               */
      /*************************************************************************/
      (void)x25_pvc_free(conn_id);
      (void)x25_ctr_remove(ctr_id);
      (void)x25_term(&cb_link_name);
    }
  }
  return(0);
}
```

List of X.25 Programming References

This list, for programs using the COMIO emulation port, includes:

- Subroutines
- API structures
- API error codes
- Header file
- Example programs

Subroutines

The list of X.25 subroutines is organized by function:

- Initialization and termination subroutines
- Network subroutines
- Counter subroutines

- Management subroutines

Initialization and Termination Subroutines

The subroutines that begin and end X.25 sessions are:

x25_init	Initializes the API for a particular X.25 port.
x25_term	Terminates the API for a particular X.25 port.

Network Subroutines

The network subroutines that establish calls, transmit data, clear calls, and allocate network resources are:

x25_ack	Acknowledges data received with the D-bit set.
x25_call	Sets up a switched virtual circuit and establishes the call.
x25_call_accept	Accepts an incoming call.
x25_call_clear	Clears a call.
x25_listen	Starts listening for incoming calls.
x25_deafen	Turns off listening.
x25_interrupt	Sends an interrupt.
x25_pvc_alloc	Allocates a permanent virtual circuit.
x25_pvc_free	Frees a permanent virtual circuit.
x25_receive	Receives a message and indicates the message type.
x25_reset	Resynchronizes communications.
x25_reset_confirm	Sends a reset-confirmation message.
x25_send	Sends data.

Counter Subroutines

To monitor and control what is happening during a call, your application can use these counters supplied by the API:

x25_ctr_get	Gets a counter.
x25_ctr_remove	Removes a counter.
x25_ctr_test	Gets the current value of a counter.
x25_ctr_wait	Suspends the current process until one of the counters has exceeded a specified value, usually 0.

Management Subroutines

These subroutines can be used to control and monitor X.25 links:

x25_device_query	Returns information about some of the attributes of an X.25 adapter.
x25_circuit_query	Returns information about a virtual circuit.

API Structures

The list of X.25 API structures is organized by function:

- Miscellaneous structures
- Structures used to establish calls and transfer data
- Structures used to clear, interrupt, and reset calls
- Structures used to manage X.25 communications

Miscellaneous Structures

cb_link_name_struct	Used to indicate the name of the X.25 port.
cb_msg_struct	Used to indicate the type of message being received.

ctr_array_struct	Used to store the counter values and identifiers for use with the **x25_ctr_wait** structure.

Structures Used to Establish Calls and Transfer Data

cb_call_struct	Used for calls made and accepted.
cb_data_struct	Used for data transferred during a call.
cb_fac_struct	Used for information about optional facilities being used.
cb_pvc_alloc_struct	Used to indicate the logical channel number and port assigned to a PVC.

Structures Used to Clear, Interrupt, and Reset Calls

cb_clear_struct	Used for calls being cleared.
cb_int_data_struct	Used for data sent or received in an interrupt packet.
cb_res_struct	Used for data sent or received in a reset-request packet.

Structures Used to Manage X.25 Communications

cb_circuit_info_struct	Used for information about a virtual circuit.
cb_dev_info_struct	Used for information about an X.25 adapter.
cb_link_stats_struct, x25_query_data, and x25_stats	Used for statistics for an X.25 port.

API Error Codes

The list of X.25 API error codes is organized by function:

- X.25-Specific error codes
- System error codes

X.25-Specific Error Codes

For X.25-specific error conditions, the **x25_errno** flag is set to one of the following values:

X25ACKREQ	One or more packets require acknowledgement. Issue the **x25_ack** subroutine before continuing.
X25AUTH	The calling application does not have system permission to control the status of the link.
X25AUTHCTR	The application does not have permission to remove this counter because it did not issue the corresponding the **x25_ctr_get** subroutine.
X25AUTHLISTEN	The application cannot listen to this name, because the corresponding entry in the routing list has a user name that excludes the user running the application. Use another routing list name, or change the user name in the routing list entry.
X25BADCONNID	The connection identifier is invalid.
X25BADDEVICE	The X.25 port name is invalid.
X25BADID	The connection identifier or listen identifier is invalid.
X25BADLISTENID	The listen identifier is invalid.
X25CALLED	The called address is invalid. Check that the address is correct and is a null-terminated string.
X25CALLING	The calling address is invalid. Check that the address is correct and is a null-terminated string.
X25CTRUSE	The counter has a nonzero value.
X25INIT	X.25 is already initialized for this X.25 port, so it cannot be initialized again.
X25INVCTR	The specified counter does not exist. (In the case of the **x25_ctr_wait** subroutine, the counter is one of an array of counters.)
X25INVFAC	An optional facility requested is invalid. Check the structure **cb_fac_struct**.
X25INVMON	The monitoring mode is invalid.
X25LINKUP	The X.25 port is already connected.

X25LINKUSE	The X.25 port still has virtual circuits established; it may still be in use. Either free all virtual circuits or disconnect the port using the override.
X25LONG	The parameter is too long. Check each of the parameters for this subroutine.
X25MAXDEVICE	Attempts have been made to connect more X.25 ports than are available. Check the **smit** configuration to see how many ports are available.
X25MONITOR	X.25 traffic on this X.25 port is already being monitored by another application. The other application must stop monitoring before any other application can start monitoring.
X25NAMEUSED	Calls for this name are already being listened for.
X25NOACKREQ	No packets currently require acknowledgment.
X25NOCARD	The X.25 adapter is either not installed or not functioning.
X25NOCTRS	No counters are available.
X25NODATA	No data has arrived for this connection identifier. Issue the **x25_ctr_wait** subroutine to be notified when data arrives.
X25NODEVICE	The X.25 device driver is either not installed or not functioning.
X25NOLINK	The X.25 port is not connected. Issue the **x25_link_connect** subroutine, or use the **xmanage** command to connect it.
X25NONAME	The name is not in the routing list. Add the name or use one that is already in the list.
X25NOSUCHLINK	The X.25 port does not exist. Check the **smit** configuration.
X25NOTINIT	The application has not initialized X.25 communications. Issue the **x25_init** subroutine.
X25NOTPVC	This is not defined as a permanent virtual circuit (PVC). Check the **smit** configuration.
X25PROTOCOL	An X.25 protocol error occurred.
X25PVCUSED	This permanent virtual circuit (PVC) is already allocated to another application. The other application must free the PVC before it can be used.
X25RESETCLEAR	The call was reset or cleared during processing. Issue the **x25_receive** subroutine to obtain the reset-indication or clear-indication packet. Then issue the **x25_reset_confirm** or **x25_clear_confirm** subroutine, as necessary.
X25SYSERR	An error occurred that was not an X.25 error. Check the value of **errno**.
X25TABLE	The routing list cannot be updated because the **xroute** command is using it. Try again after **xroute** command has completed.
X25TIMEOUT	A time-out problem occurred.
X25TOOMANYVCS	No virtual circuits are free on the listed X.25 ports.
X25TRUNCTX	The packet size is too big for internal buffers, so data cannot be sent.

System Error Codes

For non-X.25-specific error conditions, the **x25_errno** flag is set to `X25SYSERR`, and the **errno** global variable is set to one of the following values:

EFAULT	Indicates a bad address pointer.
EINTR	A signal was caught during the call.
EIO	An I/O error occurred.
ENOMEM	Could not allocate memory for device information.
ENOSPC	There are no buffers available in the pool.
EPERM	Calling application does not have sufficient authorization.

Header Files

x25sdefs.h Contains the structures used by the X.25 application programming interface (API).

Example Programs

pvcrcv	Receives a call using a permanent virtual circuit.
pvcxmit	Sends a call using a permanent virtual circuit.
svcrcv	Receives a call using a switched virtual circuit.

svcxmit Sends a call using a switched virtual circuit.

Chapter 11. X.25 Power Management

This section applies to AIX 5.1 and earlier.

AIXlink/X.25 for drivers has been updated to provide for Power Management (PM) support. PM is a technique that enables hardware and software to minimize system power consumption. PM is generally only important to low-end models, such as Notebooks and systems operating on battery.

When Power Management is enabled, the system enters a power-saving mode under a number of conditions including the expiration of the idle timer, a direct command from the user, a low battery, or the closing of the Notebook lid. PM state transitions include: **enable**, **standby**, **suspend**, **hibernation**, and **shutdown**. Each transition implies further decreasing the power supply to the various system components. Only the **suspend**, **hibernation**, and **shutdown** states have any impact on X.25 connections.

Impact to External Connection - Network Provider

From an X.25 standpoint, bringing a system to the **suspend**, **hibernation**, or **shutdown** states results in loss of power to the adapters. All network connections are lost. Externally, this can be viewed as pulling the physical connection (cable), since there is a complete loss of signal power at the physical layer.

Since all signals are dropped at the physical layer, there is no opportunity for the local DTE packet layer to send out clear requests to the DCE packet layer. Similarly, the frame layer does not go through the usual DISC/UA sequence prior to bringing down the link level. All X.25 connections remain down until the system is re-enabled.

Once power is restored, all X.25 physical connections previously up are restored. Two possible exceptions are dial-up connections and ports controlled directly by DLPI applications. For dial-up, even though the physical layer is reinitialized for two-way activation, these connections are only physically reconnected when there is user activity, such as an outgoing call request or an incoming call. Likewise, for DLPI ports, it is necessary for the applications to reconnect (**DL_CONNECT** request) to activate the port.

> **Note:** Restoring an X.25 connection refers to bringing up the physical, frame, and packet layers between the local DTE and DCE. Switched virtual circuits (SVCs) that were active before the power loss have been cleared. Permanent virtual circuits (PVCs) have been reset. It is up to the corresponding applications to re-establish the SVCs and properly resynchronize the PVCs.

Impact to Local Applications: DLPI, TCP/IP, NPI, COMIO, and PAD

A system shutdown impacts X.25 applications differently than either a **suspend** or **hibernation** transition. In the case of a shutdown, all user applications are terminated. With a system shutdown, no attempt is made to preserve the current user state before powering off. On bring-up, those ports and interfaces defined in the ODM are newly configured. All X.25 user application need to be restarted after power-on.

In the case of the **suspend** and **hibernation** states, there is an attempt to preserve as much of the current system state as possible in order to make the off-on cycle transparent to user applications. All user applications remain active. For X.25, however, the network connections have been physically lost and restarted. Losing the network connections directly impacts X.25 applications. The Power Management suspend and hibernation power-off and power-on cycles are generally not completely transparent.

The following highlights the expected behaviors during a PM suspend or hibernation power-off and power-on cycle for each of the programming interfaces:

DLPI	Receives a DL_DISCONNECT indication. It is up to the DLPI application to properly handle the indication. The port is not activated until a new DL_CONNECT request is issued by the application. It is not necessary to re-bind to the port since all the DL_BINDs are maintained. For detailed information on DLPI, refer to the DLPI Overview.
TCP/IP	Provides a highly robust and transparent interface. In most cases, it is transparent to the application that the underlying virtual circuit was reset or cleared. As long as the application remains active, the X.25 IF driver attempts to re-establish the virtual circuit. If it succeeds in establishing the connection, the power-off/power-on cycle is transparent to the application. For information on sockets, refer to Sockets Overview in *AIX 5L Version 5.2 Communications Programming Concepts*. For information on TCP/IP commands, refer to *AIX 5L Version 5.2 Commands Reference*.
NPI	Receives **N_DISCON_IND**s on all active SVCs. Likewise, **N_RESET_IND**s (RESET_reason: **N_NET_LINK_DOWN**/**N_NET_LINK_UP**) are received on all configured and bound PVCs. The application needs to properly handle these primitives. Note that all **N_BIND_REQ**s are maintained as long as the application does not unbind, close, or exit. This implies that all PVCs remain bound, and all listens are still active. All SVCs need to be re-established through new **N_CONN_REQ**s/**N_CONN_IND**s. For detailed information on NPI, refer to the NPI Overview.
COMIO Emulation	In the case of a base X.25 API application running through the COMIO emulator, all active calls (SVCs) are cleared and all PVCs are reset. Before the sessions are re-established, the application needs to ensure that it has properly handled the completion of the aborted sessions. A Library API user needs to read all clears and resets (**x25_receive**), while an application using the COMIO emulator need to complete a **CIO_HALT**/**CIO_HALT_DONE** cycle for all sessions. It is not necessary for the application to reinitialize (**x25_init**), reattach any PVCs (**x25_pvc_alloc**/**CIO_START-logical_channel**), or re-establish any listens (**x25_listen**/ **CIO_START-listen_name**). However, it is up to the application to re-establish any SVCs (**x25_call**/**CIO_START**, **x25_call_accept**/ **CIO_START**). For detailed information on the base X.25 API, refer to X.25 Application Programming Interface in Common Input/Output Emulation.
PAD	All active PAD connections are cleared. If using the **xspad** application, a CLEAR is received on all active sessions. PAD sessions can be re-established by issuing the appropriate **CALL** commands. For more detailed PAD information, refer to Packet Assembler/Disassembler (PAD) Overview.

Power Management Limitation Warnings

The following is a list of Power Management warnings:

Changing configuration during suspend/hibernation	Altering the system configuration, such as devices, ports, etc., while the system is in the **suspend** or **hibernation** state transitions can cause unpredictable results. This could cause loss of data, file system corruption, system crashes, or a failure to resume from the **suspend** or **hibernation** states.

Non-PM-aware device drivers	If a device driver is installed that is not PM-aware, unpredictable results could occur when resuming from **suspend** or **hibernation**. If a non-PM-aware device driver is installed, the **suspend** and **hibernation** states must NEVER be used. The following command can be run with root authority to disable the states, effective on the next system boot:
	`/usr/lib/boot/disable_hibernation`
	
	To re-enable these states, use the following command, effective on the next system boot:
	`/usr/lib/boot/enable_hibernation`

Booting from CD-ROM or other media after hibernation	Accessing the rootvg from maintenance mode, such as a CD-ROM boot when a valid hibernation image exists, can result in loss of data and file system corruption.

Maintenance modes after hibernation

To avoid loss of data and file system corruption, maintenance modes should only be used after normal system shutdown or power-off, not after a hibernation power-off.

Network connections during suspend/hibernation

Network connections are disconnected during the **suspend** and **hibernation** states. Since locally cached data will not be available to other nodes on the network during this time and network activity cannot be monitored by the local node during this time, it is recommended that the **suspend** and **hibernation** states NOT be used when running network interfaces such as X.25, TCP/IP, NFS, AFS, DCE, SNA, OSI, NetBIOS, etc.

The following command can be run with root authority to disable the states, effective on the next system boot:

```
/usr/lib/boot/disable_hibernation
```

To re-enable this function, use the following command, effective on the next system boot:

```
/usr/lib/boot/enable_hibernation
```

Power Button Behavior

When PM is enabled, the power button is software controlled. If there is a system problem, the software necessary to make the requested Power Management state transition using the power switch may not be able to run. In such a situation, it should always be possible to turn off the power immediately by pressing the power button three times quickly (within a two-second period). This overrides whatever state transition was selected for the power switch and requires a full boot.

In addition, if the PM daemon (**/usr/bin/pmd**) is never started (by default, an entry in **/etc/inittab**), the power switch acts as if there was no Power Management. A single button press turns off the system. If **/usr/bin/pmd** is started and then killed, the first two button presses are ignored and the third turns off the system. These button presses can be over any period of time as long as **/usr/bin/pmd** is not restarted.

Chapter 12. X.25 Problem Determination

Flashing 888 Problems

Flashing 888-102

An initial value of 102 indicates an unexpected system halt during normal operation.

For unexpected system halts, the string of three-digit display values has the following format:

```
888 102 mmm ddd
```

where `mmm` is a value indicating the cause of the halt and `ddd` is a value indicating whether or not a system dump was obtained.

Refer to the hardware problem determination procedures supplied with your system. If these procedures return an SRN, record that SRN in item 4 of the Problem Summary Form and report the problem to your service organization.

If the diagnostics do not detect a problem, record SRN 101-*mmm* in item 4 of the Problem Summary Form and report the problem to your service organization. If a system dump was obtained, copy the dump to removable media and be prepared to make it available to your service organization.

The following list gives the possible values of *mmm*, the second value that follows the 888, and the cause of the system halt invoking that value:

- 200 Machine check due to memory bus error (RAS/CAS Parity).
- 201 Machine check due to memory timeout.
- 202 Machine check due to memory card failure.
- 203 Machine check due to address exception: address out of range.
- 204 Machine check due to attempted store into ROS.
- 205 Machine check due to uncorrectable ECC due to address parity.
- 206 Machine check due to uncorrectable ECC.
- 207 Machine check due to undefined error.
- 300 Data storage interrupt - processor type.
- 32*x* Data storage interrupt - input/out exception - IOCC. The number represented by *x* is the BUID.
- 38*x* Data storage interrupt - input/output exception - SLA. The number represented by *x* is the BUID.
- 400 Instruction storage interrupt.
- 500 External interrupt - Scrub - memory bus error (RAS/CAS Parity).
  External interrupt - DMA - memory bus error (RAS/CAS Parity).
  External interrupt - undefined error.
- 52*x* External interrupt - IOCC type - channel check.
  External interrupt - IOCC type - bus timeout.
  External interrupt - IOCC type - keyboard external.
  The number represented by *x* is the IOCC number.
- 700 Program interrupt.
- 800 Floating point unavailable.

The value of *ddd*, the third value following the 888, indicates the current dump status. The possible values and meanings of *ddd* are:

- 0c0 Dump completed successfully.

- 0c4 Partial dump completed.
- 0c5 Dump failed to start. An unexpected error occurred while the system was attempting to write to the dump device.
- 0c8 Dump failed. No primary dump device is configured.

Flashing 888-103

An initial value of 103 indicates a diagnostic message. Diagnostic messages are displayed in the three-digit display when the console display is not present, or is unavailable because of a display or adapter failure, or when a failure is detected that prevents the completion of IPL.

The string of three-digit display values identifies the SRN, and up to four field replacement Units (FRUs). The string of three-digit display values has the following format:

- 888 103 nnn nnn c01 1ee 2ee 3dd 4dd 5 ss 6ss 7ff 8ff
- c02 1ee 2ee 3dd 4dd 5ss 6ss 7ff 8ff
- c03 1ee 2ee 3dd 4dd 5ss 6ss 7ff 8ff
- c04 1ee 2ee 3dd 4dd 5ss 6ss 7ff 8ff

The two values nnn nnn represent the SRN. The values c01, c02, c03 and c04 indicate the first, second, third and fourth FRUs, respectively. For each FRU, the value sequence 1ee 2ee 3dd 4dd 5ss 6ss 7ff 8ff is the location code. Refer to your diagnostic information for interpretation of these location codes.

Record the SRN in item 4 of the Problem Summary Form and the location codes in item 6 of the Problem Summary Form. Then, report the problem to your service organization.

Forcing a System Dump

If the system did not produce a dump automatically because of a hang condition, obtain a dump while the problem exists.

> **Note:** The use of the Reset button is the preferred method of obtaining dumps, because in a hang condition it is the dump trigger most likely to succeed. If the system has dumped and halted automatically, the Reset button will scroll the LEDs rather than trigger another dump.
>
> **Attention:** Obtaining a dump overwrites a previous dump or other data stored on the dump device.

If the console or a tty is accepting commands, you can start a dump using the **sysdumpstart** command. The **sysdumpstart** command allows you to start a dump to the primary or the secondary dump device. If the system is accepting commands, use the following procedure:

1. To determine which devices have been assigned as the primary and secondary dump devices, enter:
   ```
   sysdumpdev -l
   ```
2. To start a dump to the primary dump device, log into the system as the root user and enter:
   ```
   sysdumpstart -p
   ```

If the system is not accepting commands, then try one of the following:

- If there is a keyboard attached to the system unit, you can start a dump using the dump key sequences (Ctrl-alt-numpad1 and Ctrl-alt-numpad2). The dump key sequences allow you to start a dump to the primary or the secondary dump device.
- The Reset button can be used to start a dump to the primary dump device. To start a dump, turn the Key Mode switch to the Service position and press the Reset button.

X.25 Problem Diagnosis

Before investigating any problem, ensure that X.25 communications are set up correctly. The following commands may help diagnose the problem:

- The **x25mon** command.
- The **lsx25** command.
- X.25 clear and reset codes .

If it is required to trace the internal working of the X.25 stack, the following trace points are available. These traces, along with a line trace from **x25mon**, a microcode trace using **sx25debug** and a system configuration table form **lsx25**, would help diagnose the code's behavior.

25C Packet layer
329 X.25 TCP/IP interface
32A NPI
32B X.25 system utilities
32C Triple-X PAD
33B COMIO emulation
33C Adapter driver
2D8 Frame layer (for ports using the **hdlc** driver)
41E Physical layer (for ports using the **hdlc** driver)

The traces produced from these trace points are not in a form that is directly useful to a system user or system administrator. They are designed to allow diagnosis of code flow.

The frame layer and physical layer code, running on the adapter, are also equipped with error point tracing. If necessary, it is possible to capture the error logs from the microcode using **sx25debug**.

See the following for further discussion of X.25 problems and solutions:

- Diagnosing Problems with Connecting to the X.25 Network
- Diagnosing Problems with Making an Outgoing X.25 Call
- Diagnosing Problems with Receiving an Incoming X.25 Call
- Diagnosing X.25 Packet Problems
- Diagnosing X.25 Command Problems.

Starting Traces

The following procedure describes how to take a trace:

1. Start the x25mon and system traces:

```
x25mon -fpct -n sx25a0
trace -a -j 25C,33B,329,33C
```

2. Recreate the problem.
3. Stop the traces:

```
trcstop

kill <pid> of x25mon
```

Possible Causes

1. Verify that the cables are attached correctly and are secure.
2. Verify that the modem-to-modem eliminator/switch is working properly.
3. Verify that the board is seated properly.
4. Verify that the device driver and ports are configured correctly.

Diagnosing Problems with Connecting to the X.25 Network

Problem 1

When adding a port or driver, the add fails with the message `A device is already configured at the specified location`.

Suggestions

- This indicates that another device is using the resources that are needed. For example, an X.25 port is already configured on the port number requested. Another example is a device driver is already available on the adapter selected. The **lsx25** command shows how the X.25 system is configured.

Problem 2

Attempts to connect an X.25 port fail when making the port via **mksx25** or through SMIT. The physical and the frame layers remain down. The **x25mon** command shows that there is no line activity.

Suggestions

- The cable is absent or loose.
- There is no carrier (DCD or RLSD) from the network terminating unit (NTU).
- There is a cabling problem such as an extra null.
- The X.25 adapter is not seated correctly.

Problem 3

Attempts to connect an X.25 port fail. The physical layer is established for several seconds, as shown by the lights on the NTU, but then goes down again. The **x25mon** command indicates that there is no line activity.

Suggestions

- The X.25 adapter is expecting a clock signal and not receiving one. Adjust the NTU to provide clocking.
- The cable is loose, causing the clock pin not to be connected.

Problem 4

Attempts to connect an X.25 port fail. The physical layer is connected but the frame layer fails to come up. **x25mon** monitoring the frame layer shows a sequence such as a string of `SABM`s.

Suggestion

The packet layer type of line attribute is DCE rather than DTE for this X.25 adapter. Ensure that the DTE/DCE switching configured in SMIT is suitable for the device being attached to.

The frame layer might need to be set for automatic detection. Use SMIT to change it.

Diagnosing Problems with Making an Outgoing X.25 Call

Problem 5

On starting **xtalk**, the initial screen contains the message `You cannot make outgoing calls`. This implies that there are no COMIO emulation ports, `x25s`.

Suggestion

Use **lsx25** to see if there any COMIO emulation ports available. If not, add COMIO emulation to any of the X.25 ports, `sx25a`, that require them.

Problem 6

Incoming calls are arriving, but outgoing calls cannot be made, on a switched virtual circuit (SVC).

Suggestions

- The configuration of the SVC logical channel numbers is incorrect. Check your network subscription and use SMIT to check the network attributes.
- Use SMIT to check that outgoing calls and local charges are allowed.

Problem 7

A call is being cleared with a cause code from 1 through 127.

Suggestion

The diagnostic code gives details of why the call is being cleared. Common causes are that the network user address (NUA) is not known, the X.25 line is not connected, or unsupported optional facilities have been requested.

> **Note:** Cause codes 80 through FF are set by SNA. When SNA is being used, the diagnostics are interpreted differently.

Diagnosing Problems with Receiving an Incoming X.25 Call

Problem 8

The **x25mon** command indicates that an incoming call has arrived at the adapter, but it is not being routed to the application that is running.

Suggestions

- If the call is being cleared with cause 0 and diagnostic 0, it may be that the application is not listening to a name in the routing list that matches the incoming call. There may be another name in the routing list that is a better match for the call and has reject (R) specified as the action.
- The incoming call may have requested an optional facility that is not allowed by the current configuration.
- The incoming call may have arrived on an invalid logical channel. Check your network subscription and the network configuration attributes in your SMIT configuration.
- Use SMIT to check that incoming calls are allowed.
- Check the cause and diagnostic codes with the standard list of codes and any network-specific codes.
- A different application may be listening to criteria that are a better match those which your application is listening. Check for another program using X.25.

Diagnosing X.25 Packet Problems

Problem 9

Whenever a packet is sent on a permanent virtual circuit (PVC), the network sends a clear-indication packet.

Suggestion

The PVC logical channel number ranges are set incorrectly. Check your network subscription and your SMIT configuration.

Problem 10

When sending a data packet with the D-bit set on an switched virtual circuit (SVC), a reset-request packet is sent instead.

Suggestion
The intention to use D-bits must be made clear when the call is originally established, either in the call-request packet or in the call-accepted packet.

Problem 11
When sending a data packet with the D-bit set on a permanent virtual circuit (PVC), a reset packet is sent instead.

Suggestion
Use SMIT to configure the PVC to use D-bits.

Problem 12
When sending an interrupt packet with more than one byte of user data, a reset packet is sent instead.

Suggestion
The 1980 version of X.25 supports exactly one byte of user data in the interrupt packet. The 1984 version supports up to 32 bytes. Check your subscription and the value of the CCITT support attribute in SMIT.

Problem 13
When sending large packets on slow lines, the link sometimes gets restarted.

Suggestion
The CCITT timer, T1, may have expired. Use SMIT to increase the value of the T1 attribute, or use smaller packets. Before changing T1 or packet size values, verify these values with your network provider.

Diagnosing X.25 Command Problems

Problem 14
Attempts to start up the **x25mon** command on the X.25 port fails.

Suggestion
Only root may start the X.25 monitoring.

Problem 15
The **xmanage**, **xcomms** and **xmonitor** commands are not in AIXlink/X.25 Version 2.0.

Suggestion
These commands are not shipped with AIXlink/X.25 Version 2.0. The x25mon command replaces xmonitor, and the x25status and lsx25 commands provide link status on ports that have COMIO emulation.

Problem 16
None of the X.25 commands get past the title screen.

Suggestion
Use the **echo $TERM** command to find out your terminal-type setting and make sure that it matches your actual terminal type.

Problem 17
The number of virtual circuits cannot be configured past an upper limit.

Suggestion
The maximum number of VCs supported depends on the license of the product that was purchased. Use `smit chg_sx25vcS` to see the licensed number of VCs.

Diagnosing PAD Problems

Problem 18
Unable to get a CALL to complete after running **xspad -l sx25a0**. For example:

```
CLEAR DTE 0 136 -
Call cleared, by remote device, data may be lost
```

Suggestion
Verify that the side you are calling has a PAD or X.29 daemon running. Verify that the switch/modem-to-modem/eliminator is configured and set up to acknowledge your call. Also verify that the NUA is configured properly.

Problem 19
When logging in through the PAD, the display isn't working as expected. For example, no backspace, echo, problems with row and column widths, etc.

Suggestion
Check the stty settings and verify that they are set like a ASCII terminal. Verify that your echo, row, and columns are set correctly.

Problem 20
Unable to get **xspad -l sx25a0** to work. For example:

```
Unable to retrieve port 'sx25a0' from CuAt
```

Suggestion
Verify that port `sx25a0` is configured.

Appendix A. X.25 Commands

The following are the commands needed to program X.25:

- backupx25 Command
- chsx25 Command
- lspvc Command
- lsx25 Command
- mkpvc Command
- mksx25 Command
- removex25 Command
- restorex25 Command
- rmsx25 Command
- sx25debug Command
- x25ip Command
- x25mon Command
- x25sessions Command
- x25status Command
- xroute Command
- xspad Command
- xtalk Command

backupx25 Command

Purpose

Backs up the configuration information for the X.25 LPP into files.

Syntax

backupx25 [**-f**] [**-d** *Directory*] [**-v**]

backupx25 -h

Description

The **backupx25** command saves information concerning the configuration and attribute information for a particular machine to assist you with system administration. This aids customers who want to re-install the X.25 LPP or want to replicate their setup to other machines, assuming the adapter and port information is the same. The files produced by the **backupx25** command are read by the **restorex25** command to restore your setup.

To reinstall the X.25 LPP, it is suggested that **backupx25** is used to save configuration information, port definitions be removed, re-install the **X.25 LPP** and then restore the configuration using the **restorex25** command.

Flags

-d *Directory*	Specifies the name of save directory. Defaults to current directory.
-f	Forces removal of existing backup files if they are already present in the save directory.
-h	Displays the command usage.
-v	Specifies verbose mode (displays messages).

Security

Access Control: You must have root authority to run this command.

Example

To backup configuration information for the X.25 LPP in the directory **/tmp/x25setup** in a verbose manner, enter:

```
backupx25 -d /tmp/x25setup -v
```

Files

/usr/bin/backupx25	Contains the **backupx25** command for AIX Version 4.
/usr/lpp/sx25/bin/backupx25	Contains the **backupx25** command for AIX 3.2.5.

Related Information

The **restorex25** command, **lsdev** command, **lspvc** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, **rmsx25** command.

---

chsx25 Command

Purpose

Re-initializes the attributes of an X.25 port.

Syntax

chsx25[**-l** *Name*] [**-a** *Attribute*=*Value...*] [**-p** *ParentName*] [**-P** | **-T**] [**-q**] [**-w** *ConnectionLocation*] [**-f** *File*]

chsx25 -h

Description

The **chsx25** command re-initializes the attributes of the specified X.25 port.

Flags

-l *Name*	Specifies the defined port, indicated by the *Name* parameter, in the customized devices object class.
-p *ParentName*	Specifies the new parent device logical name from the customized devices object class. This flag is used only when changing the parent of the port.
-w *ConnectionLocation*	Specifies the new connection location on the parent where this child device is to be defined. This flag is used only when changing the connection location of the port.

Note: *ConnectionLocation* value corresponds to the port number where X.25 is configured.

-a *Attribute=Value*
Specifies the X.25 port attributes and their values. You can either use one **-a** flag for a string of *Attribute=Value* pairs and enclose the string in single quotation marks, or use one **-a** flag for each pair.

Two attributes are accepted by the **chsx25** command; the **network_id** and **country_prefix**.

network_id
Network identifier. This attribute may have one of the following values:

1 Datex-P

2 Datapac

3 Telenet

4 DDN

5 Other public

6 Other private

7 PSS-1 Extended

8 TRANSPAC

country_prefix
The data country/geographical area codes as defined in CCITT-X.121 Annex D. The *AIX/V3 X.25 Communications Cookbook* also contains a listing of the country codes under Appendix C.

-f *File*
Reads the needed flags from the *File* parameter. The user should not include the **-a** *Attribute=Value* entries within *File*. All attributes should be specified on the command line.

-P
Changes the characteristics of the device permanently in the customized devices object class without actually changing the device. The change can be made to the database with the **-P** flag; and then by restarting the system, the changes will be applied to the port. This flag cannot be used with the **-T** flag.

-T
Changes the characteristics of the device temporarily without changing the customized devices object class for the current start of the system. This flag cannot be used with the **-P** flag.

-h
Displays the command usage message.

-q
Suppresses the command output messages from standard output and standard error.

Examples

To re-initialize X.25 on port sx25a1 with network ID of 5 and a country code of 334, enter:

```
chsx25 -l sx25a1 -a 'network_id=5 country_prefix=334'
```

Related Information

The **chdev** command, **lsdev** command, **lspvc** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, and **rmsx25** command.

lspvc Command

Purpose

Lists the non-default permanent virtual circuits (PVCs) defined on an X.25 port.

Syntax

lspvc -l *Name*[**-n** *Number*] **-O**] **-N**]

lspvc -h

Description

The **lspvc** command displays the non-default PVC attribute information for an X.25 port. If the **-n** flag is used, only the PVC information for that virtual circuit is displayed. If no number is specified, the information for all PVCs defined on that port is listed.

You can display the attribute information in one of two ways. The default output is the *pvc_num rx_win rx_size tx_win tx_size* and *d-bit* attribute information separated by spaces. The **-O** flag displays all the corresponding attribute values separated by colons. Using the **-N** flag will suppress the header information.

Flags

-l *Name*	Specifies the defined port name as found in the ODM customized devices object class.
-n *Number*	Specifies the non-default PVC virtual circuit number.
-O	Displays the attribute information separated by colons for each PVC. The information for each virtual circuit number is displayed on a different line.
-N	Suppress the attribute header information.
-h	Displays the command usage message.

Examples

1. To list the current PVC attribute information for all virtual circuits on port sx25a0, enter:

   ```
   lspvc -l sx25a0
   ```

   The system displays a message similar to the following:

   ```
   pvc_num  rx_win  rx_size  tx_win  tx_size  d_bit
   1        3       128      3       128      1
   2        3       256      3       256      0
   3        7       256      7       256      0
   ```

2. To list the current PVC attribute information for virtual circuit 2 on port sx25a0, in colon format, enter:

   ```
   lspvc -l sx25a0 -n 2 -O
   ```

   The system displays a message similar to the following:

   ```
   pvc_num:rx_win:rx_size:tx_win:tx_size:d_bit
   2:3:256:3:256:0
   ```

3. To list the current PVC attribute information for all virtual circuits on port sx25a0, in colon format and suppressing the header, enter:

   ```
   lspvc -l sx25a0 -O -N
   ```

   The system displays a message similar to the following:

   ```
   1:3:128:3:128:1
   2:3:256:3:256:0
   3:7:256:7:256:0
   ```

Related Information

The **chdev** command, **chsx25** command, **lsdev** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, and **rmsx25** command.

---

lsx25 Command

Purpose

Lists the configuration of the X.25 support on the system.

Syntax

lsx25

Description

The **lsx25** command uses information available from the system's configuration database to display the relationship between adapters, drivers, ports, and so forth, that are configured to use the X.25 support. This command only shows data for devices in the available state.

There are two formats that are used to display the information. The long format displays the information in a number of different ways to make it easier to illustrate a particular relationship or attribute. This format is only available for the supported X.25 microchannel adapters.

The short format is a condensed subset of the long format. This format is only available for the supported X.25 ISA and PCI adapters.

In both the long and short formats, the data is sent to *stdout*.

The long format is divided up into:

- The name of the machine from which the command was entered.
- A list of what adapters are in each slot of the machine and what X.25 ports have been configured to use an adapter.
  - The physical port number as shown on the adapter's breakout cable if applicable. On single port adapters this is always zero.
  - The X.25 port configured. For example, `sx25a3`.
  - The port's network user address (NUA).
- Ports sorted by the X.25 port number, showing information on how the ports are referenced:

X.25 Port	Specifies the port name.
Driver	Specifies the device driver associated with the adapter where the port is configured.
NUA	Specifies the port's network user address.
COMIO	Specifies the port name if a COMIO emulation port is configured to this X.25 port.
TCP/IP	Specifies the TCP/IP name if a TCP/IP interface is configured to this X.25 port.
Logical board	Specifies the logical board number for the adapter. There is NOT a direct correlation between this number and the associated device driver number.
Logical port	Specifies a logical port number associated with each X.25 port. With the varying numbers of ports available on the different supported adapters, a constant logical number must be assumed to allow adapters to be moved physically in the system. Each adapter would have a block of eight ports associated with it, though the number of configured ports may be less than that.

- Ports sorted by the X.25 port number, showing information on the port's physical characteristics:

X.25 Port	Specifies the port name.
Driver	Specifies the device driver associated with the adapter where the port is configured.
Adapter	Specifies the adapter instance that the port is on (`apm` for Portmaster adapters and `ampx` for X.25 Interface Co-Processor/2 adapters).
Slot	Specifies the slot that the adapter is in.
Physical port	Specifies the port being used on the adapter.
Interface	Specifies the electrical interface in use with the given adapter when using the Portmaster adapter. The coprocessor adapters have the electrical interface selectable by the cable used and this cannot be detected by **lsx25**.

- List of X.25 ports, sorted by the network user address, given in alphabetical order.
- List of COMIO emulator ports and the associated X.25 ports.

- List of TCP/IP network interfaces, the associated IP address, and X.25 port number.

The short format displays the following:

- The name of the machine from which the command was entered.
- For each X.25 port, it shows:

ADAPTER	Specifies the adapter over which the X.25 port is configured.
DRIVER	Specifies the driver over which the X.25 port is configured.
PORT	Specifies the port name.
COMIO	Specifies the COMIO port name, if a COMIO emulation port is configured to this X.25 port.
TCPIP	Specifies the TCP/IP name, if a TCP/IP interface is configured to this X.25 port.
NUA	Specifies the port's network user address.
LOGICAL BOARD	Specifies the logical board number for the adapter. There is NOT a direct correlation between this number and the associated device driver number.
LOGICAL PORT	Specifies a logical port number associated with each X.25 port. With the varying number of ports available on the different supported adapters, a constant, logical number must be assumed to allow adapters to be moved physically in the system. Each adapter has a block of eight ports associated with it, though the number of port available to be configured may be less than that.
PHYSICAL PORT	Specifies the port being used on the adapter.

Examples

1. Typical output for the **lsx25** command printed in the long format is :

```
*****************************************************************
* Configuration report for X.25 LPP ports configured            *
*****************************************************************
Machine gladstone



*****************************************************************
* Report by slot number - bus 0                                 *
*****************************************************************
Slot 1, ppr0  POWER Gt4 Midrange Graphics Adapter
Slot 4, ampx0  X.25 CoProcessor/2 Adapter
Slot 4, twd0  X.25 Streams driver
        Physical port 0 is x25 port sx25a0 [11111]
Slot 5, apm0  8-Port Portmaster Adapter/A RS-232
Slot 5, twd1  X.25 Streams driver
        Physical port 3 is x25 port sx25a1 [22222]
Slot 6, tok0  Token-Ring High-Performance Adapter
Slot 7, ricio0  IBM ARTIC960 Adapter
Slot 7, twd2  X.25 Streams driver
        Physical port 2 is x25 port sx25a3 [44444]
        Physical port 5 is x25 port sx25a2 [33333]
Slot 8, ascsi0  Wide SCSI I/O Controller Adapter


*****************************************************************
* Report by logical location of X.25 port                       *
*****************************************************************
X.25                                    Logical Logical
Port    Driver  NUA     COMIO   TCP/IP  Board   Port
sx25a0 twd0 11111 x25s0 n/a 0 0
sx25a1 twd1 22222 n/a xs0 1 11
sx25a2 twd2 33333 x25s1 xs1 2 21
sx25a3 twd2 44444 x25s2 n/a 2 18


*****************************************************************
* Report by physical location of X.25 port                      *
```

```
*****************************************************************
X.25                              Phys.
Port    Driver  Adapter Slot    Port    Interface
sx25a0 twd0 ampx0 4 0 cable selectable
sx25a1 twd1 apm0 5 3 RS-232 (V.24)
sx25a2 twd2 ricio0 7 5 V.36
sx25a3 twd2 ricio0 7 2 V.36


*****************************************************************
* Report by Network User Address (NUA)                          *
*****************************************************************
NUA                    X.25 Port
11111                   sx25a0
22222                   sx25a1
33333                   sx25a2
44444                   sx25a3


*****************************************************************
* Report of COMIO emulators                                     *
*****************************************************************
COMIO                   X.25 Port
x25s0   sx25a0
x25s1   sx25a2
x25s2   sx25a3


*****************************************************************
* Report of X.25 TCP/IP (xs) interfaces                         *
*****************************************************************
TCP/IP          Address                 X.25 Port
xs0  1.1.1.1          sx25a1
xs1  1.1.1.2          sx25a2
```

2. Typical output for the **lsx25** command printed in the short format is:

```
POWER-based Desktop X.25 system configuration
Machine: gladstone

ADAPTER DRIVER PORT COMIO TCPIP NUA LOGICAL LOGICAL PHYSICAL
                                        BOARD PORT PORT
dpmpa0 hdlc0 sx25a2 x25s1  33333 2 16 0
dpmpa0 hdlc1 sx25a3 x25s2 xs1 44444 2 17 1
riciop0 twd0 sx25a0 x25s0 xs0 1234 0 0 0
ampx0 twd1 sx25a1   22222 1 8 0
```

Related Information

The **chdev** command, **chsx25** command, **lsdev** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, and **rmsx25** command.

---

mkpvc Command

Purpose

Creates or modifies a non-default permanent virtual circuit (PVC) on an X.25 port.

Syntax

mkpvc [**-U**] **-l***Name* **-n** *Number* [**-r** *RxWindow*] [**-s** *RxSize*] [**-t** *TxWindow*] [**-u** *TxSize*] [**-d** *Dbit*]

mkpvc -h

Description

The **mkpvc** command creates or modifies a non-default PVC for the virtual circuit number on the port specified. Not all of the PVC attribute values need to be used. If an attribute value is not entered by the user, the command will use the default PVC value found in the ODM for this port.

If the non-default PVC already exists, its values can be updated by issuing the **-U** flag along with the new attribute values.

Flags

-l *Name*	Specifies the defined port name as found in the ODM customized devices object class.
-n *Number*	Specifies the non-default PVC virtual circuit number.
-r *RxWindow*	Specifies the receive window size for the non-default PVC. The window size can range from 1-127.
-s*RxSize*	Specifies the receive packet size for the non-default PVC. The packet size can range from 16-4096 for powers of 2.
-t *TxWindow*	Specifies the transmit window size for the non-default PVC. The window size can range from 1-127.
-u*TxSize*	Specifies the transmit packet size for the non-default PVC. The packet size can range from 16-4096 for powers of 2.
-d *Dbit*	Specifies how the packet layer will handle the D bit. A value of 0 indicates the packet layer will reject the D bit, and a value of 1 indicates the packet layer will allow the use of the D bit.
-U	This flag allows the user to update the attribute values for a non-default PVC.
-h	Displays the command usage message.

Examples

1. To create a non-default PVC for virtual circuit 2, on port sx25a0, with receive and transmit window sizes of 7, and the D bit enabled, enter:

   ```
   mkpvc -l sx25a0 -n 2 -r 7 -t 7 -d 1
   ```

2. To change the receive window and packet sizes for the non-default PVC defined on virtual circuit 2 of port sx25a0, enter:

   ```
   mkpvc -l sx25a0 -n 2 -r 127 -s 2048 -U
   ```

Related Information

The **chdev** command, **chsx25** command, **lsdev** command, **lspvc** command, **mkdev** command, **mksx25** command, **rmdev** command, and **rmsx25** command.

---

mksx25 Command

Purpose

Initializes the attributes of an X.25 port.

Syntax

mksx25 { [**-c***Class*] [**-s** *Subclass*] [**-t** *Type*] } [**-l** *Name*][**-a** *Attribute* **=***Value...*] [**-d** | **-S**] [**-p** *ParentName*] [**-q**] [**-w** *ConnectionLocation*] [**-f** *File*]

mksx25 -l *Name* [**-S**] [**-f** *File*]

mksx25 -h

Description

The **mksx25** command initializes the attributes of the X.25 port specified. The System Management Interface Tool (SMIT) fast path select may be used to run this command. To add a port to a parent configured on a Portmaster adapter, using SMIT, enter:

```
smit mksx25pm
```

To add a port to a parent configured on an X.25 CoProcessor adapter, using SMIT, enter:

```
smit mksx25c
```

This is a fast path to the SMIT screens that allow the port to be added to the system.

Flags

-a *Attribute=Value*
Specifies the X.25 port attributes and their values. You can either use one **-a** flag for a string of *Attribute=Value* pairs and enclose the string in single quotation marks, or use one **-a** flag for each pair. Four attributes are accepted by the **mksx25** command:

- **local_nua**
- **network_id**
- **country_prefix**
- **nddname**, the ″Network Device Driver″ name for the port driver. This attribute is used when the X.25 port is managed by the **hdlc** device driver.

local_nua
The port's network user address is required.

network_id
Network identifier. This attribute may have one of the following values:

1	Datex-P
2	Datapac
3	Telenet
4	DDN
5	Other public
6	Other private
7	PSS-1 Extended

country_prefix
The data country/geographical area codes as defined in CCITT-X.121 Annex D.

-c *Class*
Specifies the device class from the predefined devices object class.

-d
Defines the device in the customized devices object class. If you specify the **-d** flag, the port will only be in the defined state and not available. This flag cannot be used with the **-S** flag.

-f *File*
Reads the needed flags from the *File* parameter. The user should not include the **-a** *Attribute=Value* entries with in *File*. All attributes should be specified on the command line.

-h
Displays the command usage message.

-l *Name*
(lowercase L) Specifies the already defined device, indicated by the *Name* parameter, in the customized devices object class when not used with the **-c**, **-s**, and **-t** flags. The **-a**, **-p**, and **-w** flags cannot be used in this case.

-p*ParentName*
Specifies the parent device logical name from the customized devices object class.

-q
Suppresses the command output messages from standard output and standard error.

-S
Prevents the device from being set to the available state. This flag is only meaningful for those devices that support the stopped state. This flag cannot be used with the **-d** flag.

-s*Subclass*
Specifies the device subclass from the predefined devices object class.

-t*Type*
Specifies the device type from the predefined devices object class.

-w*ConnectionLocation* Specifies the connection location on the parent where this child device is defined.

Note: The **-p***ParentName* and **-w** *ConnectionLocation* parameters are used when configuring a port which uses the **twd** driver. The *ConnectionLocation* value corresponds to the port number where X.25 is configured, when the port is managed by the **twd** driver.

See Chapter 4, ″Managing X.25 Ports″ for more information.

Examples

To configure X.25 on port 2 of adapter driver twd0, with an NUA of 123456789, network ID of 5, and country code of 334, enter:

```
mksx25 -c port -s star -t stx25 -p twd0 -w
2 \
>  -a 'local_nua=123456789 network_id=5 country_prefix=334'
```

Related Information

The **chdev** command, **chsx25** command, **lsdev** command, **lspvc** command, **mkdev** command, **mkpvc** command, **rmdev** command, and **rmsx25** command.

---

removex25 Command

Purpose

Removes all instances of x.25 ports.

Syntax

removex25 [**-q**] [**-d** *Directory*] [**-v**] [**-f**]

Description

The **removex25** command removes all configured information relating to the X.25 LPP.

Flags

-d *Directory* Specifies the name of the directory holding backup files. Defaults to current directory.
-f Forces removal of existing backup files if they are already present in the save directory.
-q Runs the command without asking for verifications or displaying messages.
-v Runs the command in the verbose mode, displaying messages as necessary. Default if the **-q** flag is not used.

Security

Access Control: You must have root authority to run this command.

Example

To remove configuration information for the X.25 LPP in the directory **/tmp/backupx25** and force removal of existing backup files if they are already present, enter:

```
removex25 -d /tmp/backupx25 -f
```

Files

/usr/bin/removex25	Contains the **removex25** command for AIX Version 4.
/usr/lpp/sx25/bin/removex25	Contains the **removex25** command for AIX 3.2.5.

Related Information

The **restorex25** command,**lsdev** command, **lspvc** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, **rmsx25** command.

---

restorex25 Command

Purpose

Restores the X.25 LPP configuration information from files saved when using **backupx25**.

Syntax

restorex25 [**-d** *Directory*] [**-v**]

restorex25 -h

Description

The **restorex25** command will use the information saved by the backupx25 command to restore your configuration after a fresh X.25 LPP installation.

Flags

-d *Directory*	Specifies the name of save directory. Defaults to current directory.
-h	Displays the command usage.
-v	Runs the command in the verbose mode, displaying messages as necessary.

Security

Access Control:	You must have root authority to run this command.

Examples

To restore configuration information for the X.25 LPP in the directory **/tmp/x25setup** in a verbose manner, enter:

```
restorex25 -d /tmp/x25setup -v
```

Files

/usr/bin/restorex25	Contains the **restorex25** command for AIX Version 4.

Related Information

The **backupx25** command,**lsdev** command, **lspvc** command, **mkdev** command, **mkpvc** command, **mksx25** command, **rmdev** command, **rmsx25** command.

rmsx25 Command

Purpose

Removes an X.25 port.

Syntax

rmsx25 -l *Name* [**-d**] [**-q**] [**-f** *File*]

rmsx25 -h

Description

The **rmsx25** command either configures or unconfigures and undefines the X.25 port specified by the -l flag. The default action is to unconfigure the device while retaining its device definition in the customized devices object class.

Flags

-l *Name*	Specifies the device name, indicated by the *Name* parameter, in the customized devices object class, for the port to be removed.
-d	Indicates the the port is undefined and its device information is to be removed for the customized devices object class.
-f *File*	Reads the needed flags from the *File* parameter.
-h	Displays the command usage message.
-q	Suppresses the command output messages from standard output and standard error.

Examples

To unconfigure and completely undefine port sx25a0, enter:

```
rmsx25 -l sx25a0 -d
```

sx25debug Command

Purpose

Verifies that the X.25 on-card code is functioning and gathers debug messages from the X.25 on-card code.

Syntax

For ARTIC960Hx:

sx25debug [**-D**] [**-b** *BoardNumber*] [**-i**]

All other adapters:

sx25debug [**-b** *BoardNumber*] [**-i**]

Description

The **sx25debug** command is a debugging aid that has two basic functions. If the **-i** flag is specified, you can enter characters at the keyboard, and this causes the microcode to be polled to make sure that it is operational. A response of "pse_gdebug: Alive" indicates that it is functioning normally.

Note: If the **-i** flag is specified, the **sx25debug** command cannot be run as a background process, since it must be able to access the tty.

The **-D** option for the ARTIC960 and the ARTIC960Hx adapters allows the user to use the **S** key from the keyboard to detect the modem signals and show its status. For more options, type the **?** character.

The second main function is to read debug and error messages from the microcode and print them on stdout. This can aid in debugging microcode problems.

To exit **sx25debug**, enter Ctrl-E, or press the interrupt key (usually Ctrl-C).

Note: This command is not used with the IBM 2-Port Multiprotocol PCI Adapter. See Appendix H for additional information on this adapter.

Flags

-b *BoardNumber*	Specifies the board number to communicate with. If the **-b** flag is not specified, the board number defaults to 0.
-D	Specifies debug mode. Typing specific characters from the keyboard will show debug information. Type **?** for complete information.
-i	Specifies interactive mode (reads characters from the tty). Every time a character is read, a message is sent to the microcode on the board, which should elicit a response indicating that the microcode is running. This allows the you to poll the adapter to make sure that the microcode is operational.

Security

Access Control:	You must have root authority to run this command.

Examples

1. To start an interactive session, enter:

   ```
   sx25debug -i
   ```

   This should produce the following output:

   ```
   Type is BoardType
   ```

   where *BoardType* is either ″PMA″, ″C2X″, ″C1X″, ″ARTIC960″, or ″ARTIC960Hx″. If this is the first time that the **sx25debug** command has been run since the board was configured, you will see:

   ```
   cyc_main: Entering scheduler loop.
   ```

   for a C2X or C1X board, or:

   ```
   rticmain: Entering scheduler loop.
   ```

   for a PMA board, or:

   ```
   a960main: Entering scheduler loop.
   ```

   for an ARTIC960 or an ARTIC960Hx board.

   If you then hit any character on the keyboard, you should see:

   ```
   pse_gdebug: Alive
   ```

   This indicates that the microcode is functioning normally.
2. To start a ″read-only″ session, enter:

   ```
   sx25debug
   ```

   This will print out any debug messages from the microcode.
3. Either of these modes can have their output redirected or piped by the shell. For example, if you want to start an interactive session on board number 1 and save the output to a file, enter:

```
sx25debug -i -b 1 | tee /tmp/sx25debug.out
```

Files

/usr/bin/sx25debug	Contains the **sx25debug** command.

---

x25ip Command

Purpose

Updates or displays translate information in the IP/X.25 translate table.

Syntax

x25ip [**-h** *HostName*] [**-a**] [**-o** *Options*] [**-d**] [**-s**] [**-z**]

Description

The **x25ip** command updates or displays IP/X.25 translate information. The translate table allows Internet addresses used by the Internet Protocol (IP) to be mapped to specific X.25 virtual circuits with specific X.25 circuit characteristics. New entries and updates to existing entries are written to or deleted from the IP/X.25 translate table.

> **Note:** Users must have root authority to issue the **x25ip** command.

The **x25ip** command uses the **gethostbyname** subroutine to obtain the IP address of a specified host. The host name specified should exist in the **/etc/hosts** file or be retrievable from a name server. However, the IP address can be used instead of a host name.

If the command is completed successfully, a value of 0 is returned. If the command is unsuccessful, a value of -1 is returned.

The IP/X25 translate table can be updated using the System Management Interface Tool (SMIT).

Flags

-a	Adds or changes the specified options for the given host name.
-d	Deletes the specified host.
-h *HostName*	Specifies the host name to use for add, delete and show flags.

-o *Options* Specifies options for the given host name. Valid values are:

vc_type
Specifies the X.25 virtual circuit type: switched virtual circuit (SVC), or permanent virtual circuit (PVC). Valid values are 1 for SVC and 2 for PVC. The parameters for SVC and PVC are as follows:

SVC Parameters (vc_type=1)

remote_dte
Specifies the X.25 address of the remote DTE. Valid values consist of 1 to 15 ASCII decimal digits (X.121 address). Valid values are 0 through 9.

rcv_wndsiz
Specifies the maximum receive window size to be used with the virtual circuit. Valid values are 1 through 127.

xmit_wndsiz
Specifies the maximum transmit window size to be used with the virtual circuit. Valid values are 1 through 127.

rcv_pktsiz
Specifies the maximum receive packet size to be used with the virtual circuit. Valid values are 64, 128, 256, 512, 1024, 2048, 4096.

xmit_pktsiz
Specifies the maximum transmit packet size to be used with the virtual circuit. Valid values are 64, 128, 256, 512, 1024, 2048, 4096.

callusr_data
Specifies the optional user-defined facilities to be used in the call request packet. Valid values consist of 1 through 16 HEX digits, with digit values 0 through F.

RPOA_selec
Specifies the data network identification code or codes identifying a requested RPOA transit network. Valid values consist of 4 ASCII digits, with digit values 0 through 9. Up to 10 groups of 4 digits may be specified.

cug_indx
Specifies the closed user group index to be used with the closed user group facility. Valid values are 0 through 9999.

cug_indxout
Specifies the closed user group index to be used with the closed user group outgoing access facility. Valid values are 0 through 9999.

PVC Parameters (vc_type=2)

logical_chann
Specifies the X.25 logical channel to be used for the PVC. Valid values are 1 through 4095.

port_nam
Specifies the port name (sx25a0, sx25a1, ...) that is used by this PVC.

-s Shows current options for the given host or display list of host names.

-z Displays output in dotted decimal format. This flag is not normally used if the command is issued from the command line.

Examples

1. To initialize the IP/X.25 translate table, issue the following command:
   ```
   x25ip
   ```
2. To show the current IP/X25 translate values for host `node1`, issue the following command:
   ```
   x25ip -h node1 -s
   ```
3. To add options to the host `node2`, issue the following command:

```
x25ip -h node2 -a -ovc_type=1,remote_dte=31060164,callusr_data=CC,rcv_pktsize=1024,xmit_pktsize=10
```

Related Information

The **gethostbyname** subroutine.

The **/etc/hosts** file format.

TCP/IP Network Interfaces, TCP/IP Name Resolution, How to Configure a Network Interface, The SMIT Interface for TCP/IP in *AIX 5L Version 5.2 System Management Guide: Communications and Networks*.

Object Data Manager (ODM) Overview for Programmers in *General Programming Concepts*.

---

x25mon Command

Purpose

Traces the packet and frame traffic of an X.25 port.

Syntax

x25mon -n *Port* [**-f** [**-i** *Bytes*]] [**-p** [**-d** *Bytes*]] [**-c**] [**-t**]

Description

The frame or packet activity on the X.25 port is traced, and the report is sent to stdout. If using a COMIO emulation port, or a TCP/IP network interface, the x25 port being used for that interface can be found using the **lsx25** command.

Use the **-p** and **-f** flags to turn on packet and frame layer tracing, respectively. It is possible to turn on either or both. If you don't specify either flag, the **x25mon** command defaults to the **-p** flag.

Use the **-d** and **-i** flags to vary the length of the DATA packet and INFO frame traces, for the packet and frame layer respectively. The ability to manage the number of bytes traced is important because turning on tracing, by invoking the **x25mon** command, impacts memory resources in addition to performance. Because adapters have limited memory resources, it is important to minimize the length of the frame layer INFO frame when using adapters that run the frame layer on the adapter (for example, apm, ampx, and ricio adapter types).

Use the **-t** flag to trace the frame layer timers and packet layer timers.

Use the -c flag to enable control tracing. The use of this flag can provide additional information for problem determination. Some physical and frame layer status will be included in the trace output and the contents of invalid frames and packets received can be displayed. There are 16 types of control frames and 10 types of control packets. The identifiers for these are displayed in the field for the packet or frame type in the trace output (see ″Trace components″ below). The possible control frames are:

BAD ADDR	TOO LONG	T3 TIMER	INV CR
NO CTRL	BAD NS	N2 RETRAN	NOT RDY
UNKNOWN	BAD NR	L1 UP	L1 DOWN
INV SIZE	INV STATE	L2 UP	L2 DOWN

The possible control packets are:

TOO SHORT	INV LCN	INV STATE
TOO LONG	BAD PS	UNKNOWN
BAD LCN	INV PS	

BAD GFI INV PR

Note: You must have root authority to issue the **x25mon** command.

Flags

-c	Enable control tracing.
-d *Bytes*	Specifies the number of bytes to trace for packet layer Data packets [0..4096]. If you do not specify this flag, the trace defaults to 40 bytes.
-f	Trace the frame layer activity
-i *Bytes*	Specifies the number of bytes to trace for frame layer Info frames [0..5003]. If you do not specify this flag, the trace defaults to 40 bytes. **Note:** Keep in mind that running with large traces may result in adapter memory resource constraints.
-n *Port*	Specifies the X.25 port where the traffic should be monitored. For example, sx25a0.
-p	Trace the packet layer activity
-t	Trace the frame layer timers and packet layer timers.

Examples

1. To run a packet layer trace on X.25 port sx25a2, enter:

   ```
   x25mon -p -n sx25a2
   ```

   The following output would be a typical sample.

   ```
            port    type   LCN     packet type  flags/lengths    data

   14:59:03 sx25a2   PR   0x0020   CALL     d:N  la:4 lf:0 ld:0 441965196400CD
   14:59:03 sx25a2   PS   0x0020   CF CALL       d:N
   14:59:03 sx25a2   PR   0x0020   DATA     pr:0 ps:0 dN   mN  qN  l:16  3F2A01
   34256A8492AABC63E5F21D1C2C
   14:59:03 sx25a2   PS   0x0020   DATA     pr:2 ps:0 dN   mN  qN  l:2   C301
   ```

2. To run a packet layer trace on X.25 port sx25a2 limiting the DATA packet trace, enter:

   ```
   x25mon -p -d 4 -n sx25a2
   ```

   The following output would be a typical sample.

   ```
             port  type LCN       packet  type/flags/lengths        data

   16:58:34   sx25a0 PS   0x002a   CALL    dN la:4 lf:0 ld:1         441964196500FD
   16:58:36   sx25a0 PR   0x002a   CF CALL dN
   16:57:12   sx25a0 PR   0x002a   DATA    pr:0 ps:0 dN mN qN l:64   68656C6C..
   16:57:15   sx25a0 pS   0x002a   RR      pr:1
   16:57:40   sx25a0 PR   0x002a   DATA    pr:0 ps:1 dN mN qN l:64   68656C6C..
   16:57:43   sx25a0 PS   0x002a   RR      pr:2
   ```

3. To run a frame layer trace on the same port, limiting the INFO frame trace to 8 bytes, enter:

   ```
   x25mon -f -i 8 -n sx25a2
   ```

   from which the following out would be a typical sample.

   ```
             X.25  type physical frame  type/flags/lengths     data
             port       port

   17:15:02   sx25a0 FS  0x0000   INFO  a:3 p:0 ns:5 nr:3 l:10  102A0B441964196500FD
   17:15:03   sx25a0 FR  0x0000   RR    a:3 p:0 nr:6
   17:15:05   sx25a0 FR  0x0000   INFO  a:1 p:0 ns:3 nr:6 l:3   102A0F
   17:15:32   sx25a0 FS  0x0000   INFO  a:3 p:0 ns:6 nr:4 l:16  102A0068656C6C6F..
   17:15:33   sx25a0 FR  0x0000   RR    a:3 p:0 nr:7
   17:15:35   sx25a0 FR  0x0000   INFO  a:1 p:0 ns:4 nr:7 l:3   102A21
   ```

Trace components

port	Specifies the X.25 Port being traced.
type	Specifies the type of data. The first character is P for packets and F for frames. The second character is S for data sent and R for data received.
LCN	Specifies the logical channel number for the packet. (Only valid for packet tracing.)
Physical Port	Specifies the physical port on the adapter associated with this X.25 port.
packet type	Specifies the type of X.25 packet or the type of control packet. For example, DATA, CALL, CF CONFIRM, or BAD LCN, INV PS, INV PR .
frame type	Specifies the type of LAP-B frame, or the type of control frame. For example, SABM, UA, INFO, or L2 DOWN, BAD ADDR, T3 TIMER.
flags	Specifies the different flags. For packets d, q, m represent the D, Q, M bits and are set to Y if the flag is set, and to N if it is not.
For frames:	
a	Gives the frame address DCE (3) or DTE (1)
p	Gives the setting of the poll/final bit
n	Gives the settings of the send and receive counters.
lengths	Specifies the lengths are given in the number of bytes they take up.
la	Total length of NUAs.
lf	Length of the facilities.
ld	Length of the call user data.
l	Length of data in data packets.
data	Shows the initial part of the data portion of the packet is shown.

Note: Data is displayed in hexadecimal. Lengths and so forth are displayed in decimal.

---

x25sessions Command

Purpose

Provides current activity status for the COMIO Emulator.

Syntax

x25sessions

Description

The **x25sessions** command provides current status information for each COMIO emulated AIXlink/X.25 port in the system. The command lists the available ports, followed by information for each listen and In/Out session on each available port. If no COMIO emulated ports are identified, **x25sessions** lists an error message and ends with a nonzero exit code.

Exit Status

This command returns the following exit values:

```
0    Successful completion.
>0   An error occurred.
```

Security

Access control: Any user.

Example

To display the current COMIO session information, and prevent the output from scrolling off the top of the display, enter:

```
x25sessions | pg
```

Files

/usr/bin/x25sessions

Related Information

Chapter 10. Common Input/Output Emulation

X25status Command

Purpose Provides current Packet level status for each AIXlink/X.25 port.

Syntax

x25status

Description

The **x25status** command provides current status information for each AIXlink/X.25 port in the system. In addition to providing the hostname, port names and current packet state for each port, **x25status** provides the number of active switched virtual circuits (SVCs), permanent virtual circuits (PVCs), and listens, for each port. The report ends with details regarding each port's active listens:

- Calling address
- Called address
- Call user data (CUD)

Exit Status

This command returns the following exit values:

```
0    Successful completion.
>0   An error occurred.
```

Security

Access control: Any user.

Example

To display the current Packet level status information, and prevent the output from scrolling off the top of the display, enter:

```
x25status | pg
```

Files

/usr/bin/x25status

Related Information

Chapter 1. X.25 Network Communications Overview

xroute Command

Purpose

Enables the routing of calls that use the COMIO emulation.

Syntax

xroute [**-s**]

Description

The **xroute** command enables the routing of calls that use COMIO emulation for X.25 support. For applications that do not use this emulation interface, the **xroute** command is unnecessary. Applications such as **xtalk**, or SNA-based applications, have their calls routed by the table maintained by the **xroute** command.

Note: The **xroute** command works with X.25 ports that have COMIO emulation configured. Refer to Managing COMIO Emulation for further information.

All applications that use the COMIO emulation and wish to receive incoming calls, submit listen requests. These requests identify table entries in the emulator's routing table. Calls that are received are passed up from the X.25 port to the emulator and it determines if any application is listening for that call. The emulator is one of a number of applications, such as TCP/IP or NPI, that will be listening for incoming calls. The emulator provides extra routing capability which allows it to route the calls that its applications are expecting. Calls for applications like TCP/IP are not affected by the **xroute** command. If the incoming call matches the criteria defined in the routing table for a specific application, the call will be routed to that application. A routing table is not needed if you are using only permanent virtual circuits (PVCs).

An application listening for an incoming call is associated with an entry name in the routing table. This entry specifies the criteria that must be satisfied for the application program to receive an incoming call.

The call user data (CUD) field of the call packet is generally used for routing but other conditions may also be tested, such as:

- The network address of the caller.
- A subaddress in the called data terminal equipment (DTE) address.
- The attachment that has received the call.

Flags

-s Download the table to the emulator.

Updating the X.25 Routing Table with the xroute Command

The default routing table is stored in the **/etc/xrt.names** file. You can update this table with the **xroute** command by logging on as the root user and entering the following command:

```
xroute -s
```

The default routing table has predefined entries for SNA, the sample program, and the **xtalk** command. The first six entries correspond to X.25/SNA protocols:

IBMELLC Identifies the SNA enhanced logical link control (LLC) used for peer-to-peer communications.
IBMPSH Identifies the physical-services header LLC used with the IBM-5793 network interface adapter (NIA).
IBMQLLC Identifies the SNA qualified logical link control (QLLC).

The two CUDs associated with each of these protocols identify their version.

To change a routing list entry move the cursor to the entry name you would like to change and type `c` for CHANGE.

The dialog screen contains the following fields:

Entry Name Specifies the name of the entry the listening program uses to find the routing criteria.

Call User Data
Indicates the part of the data received in an incoming call packet. It can be used for any purpose, but it often specifies the protocol. The **xroute** command uses only the first 64 bytes of the CUD.

For example, the CCITT defines the first four bytes of user data in a call from a packet assembler/disassembler (PAD) as hex 01000000. Enter this hex value in the CUD field of the routing table entry associated with an incoming PAD. The AIXlink/X.25 Licensed Program Product PAD does not use the xroute table.

User Name
Specifies the login name of the user who is allowed to start applications listening for incoming calls associated with this entry. An * (asterisk) indicates that any user can listen for calls using this entry. A user whose login name does not match this entry is not allowed to listen for calls corresponding to it.

For example, your routing list entry contains the user name `marcel`, as in the following entry:

```
Entry Name   ==>IBMXTALK
User Name    ==>marcel
```

If you try to start the command **xtalk -l IBMXTALK** as the user `root`, the system displays an error message like the following:

```
CIO Status 79 - X25_AUTH_LISTEN
You cannot listen to this name, because the routing list entry
has a userid which excludes the user running the application.
```

X.25 Port
Specifies the name of the X.25 port associated with the application for which the call is intended. Use an * (asterisk) to indicate any port.

Calling Address
Specifies the network user address (NUA) from which the application will receive incoming calls. An * (asterisk) at the end of this entry indicates that any digits are acceptable in the remainder of the address. Another NUA trying to call this application will be rejected.

Called Subaddress
Specifies the NUA subaddress used to route a call internally within a node. You can add additional digits to the end of an NUA up to the 15-digit limit. Use an * (asterisk) for this parameter to indicate any subaddress.

Calling Address Ext
Specifies the address extension for the calling DTE. The 1984 version of the X.25 protocol allows you to specify up to 40 additional digits of address. Use an * (asterisk) in this field to indicate any address extension.

Called Address Ext
Specifies the address extension for the called DTE. The 1984 version of the X.25 protocol allows you to specify up to 40 additional digits of address. Use an * (asterisk) in this field to indicate any address extension.

Priority (1-3)
Indicates the field that specifies when an application receives a call. If two applications need to listen to the same routing information, the priority field specifies which application receives the call if both are listening. For example, IBMSAMP is defined as listening to any condition (* in all fields), but with the lowest priority (3). Thus a background daemon could listen to IBMSAMP and log any incoming call that is not received by another application.

Action (R,F)
Indicates the field that specifies whether to forward or reject an incoming call. The field values are `R` to reject the incoming call, or `F` to forward the incoming call. The forward or reject conditions apply when no application is listening for the routing list entry which is the best match for the incoming call.

If this field is set to `R` and your application is not running when you try to establish a call, the call will be cleared with cause 0 and diagnostic 0. If the field is set to `F`, the next best match is selected, and so on until an entry that specifies `R` is selected and the call is rejected.

If a call does not match any entry in the routing table, it is cleared with cause 0 and diagnostic 0.

xspad Command

Purpose

Starts a terminal PAD (Packet Assembler/Disassembler) session.

Syntax

xspad -l*Port* [**-s***OptFile*] | [**?**] [**help**] [**-a***Dest*[**,** *Source*]] [**-c**] [**-e***Add*] [**-f**] [**-g***IDX*] [**-o***IDX*] [**-p***ATT*] [**-q**] [**-r**] [**-t***ID*] [**-u***Data*] [**-x**]

Description

The **xspad** commmand starts a terminal PAD (Packet Assembler/Disassembler) session. A PAD is a protocol converter interfacing asynchronous terminals with an X.25 network or an X.25 network with applications written for asynchronous terminals. See Packet Assembler/Disassembler for more information.

Flags

-a *Dest* [**,** *Source*] Specifies the destination X.25 address and optionally a source address. *Dest* and *Source* are ASCII strings of digits corresponding to BCD digits as described by the CCITT X.121 standard. Use of this flag is equivalent to interactively issuing the CALL command.

-c Adds the X.25 Charging facility.

-e *Add* Adds the X.25 Called Address Extension facility, where *ADD* is an ASCII string of digits corresponding to BCD digits. The X.213 format is not used since the PAD directly accesses X.25.

-f Adds the X.25 Fast Select With No Restriction facility.

-g *IDX* Adds the X.25 CUG Selection facility, where *IDX* is a 2 digit ASCII string corresponding to BCD digits of a user group index (Basic Format Only).

-l *Port* (Lowercase L) Specifies the name of a logical port to connect PAD to a host. For example, sx25a2 may represent a physical line connected to an X.25 network switch.

-o *IDX* Adds the X.25 CUG with Outgoing Access Selection facility, where *IDX* is a 2 digit ASCII string corresponding to BCD digits of a user group index (Basic Format Only).

-p *ATT* Selects an initial X.28/X.3 profile. *ATT* identifies a profile containing a set of X.3 parameter values. It can have one of the following formats:

- The # (number sign) character followed by a decimal number corresponding to a standard CCITT or custom profile, no spaces.
- An ASCII string which does not begin with # or **:**(colon) and specifies the profile name.

-q Adds the X.25 Fast Select With Restriction Of Response facility.

-r Adds the X.25 Reverse Charging facility.

-s *OptFile* Specifies that *OptFile* contains the remainder of the command-line arguments. The syntax and semantics of the arguments are the same as if they were passed on the command line, except that they do not all need to be on the same line. You should not pass other flags on the command line with this option. In *OptFile*, blank lines are ignored and lines beginning with a # character are interpreted as comments.

-t *ID* Adds the X.25 RPOA facility, where *ID* is a 4 digit ASCII string corresponding to BCD digits of network identification code as described by X.25. (i.e. Basic Format Only.)

-u *Data* Adds user data to the X.25 Call Packet, where *Data* is an ASCII string as in the D option described by the interactive documentation.

-x Specifies to exit PAD after termination of a connection, typically after issuing CLEAR or ICLEAR command or reception of a CLEAR. The default behavior is that execution continues and establishment of another connection is possible without exiting or reinvoking the PAD.

Options

? Explains command line argument use.

help Displays the PAD help menu when invoking xspad and causes ignores all flags **-l**.

Exit Status

This command returns the following exit values:

0 Successful completion.
>0 An error occurred.

Security

Access Control: Any user.

Auditing Events: N/A

Examples

To start a PAD session:

1. Ensure the PAD is configured on the system.
2. Run **xspad -l sx25a**# where # is the port number.
3. Issue the call to the remote X.25 host at the PAD prompt.
4. Log on to the X.25 host and run the desired application.

Files

/usr/bin/xspad	Contains the **xspad** command.

Related Information

Packet Assembler/Disassembler in *AIXlink/X.25 Version 2.0 for AIX: Guide and Reference*

---

xtalk Command

Purpose

Enables you to initiate or receive calls over SVCs and communicate with another user by typing messages or sending and receiving files.

Syntax

xtalk [**-l** *Name*] [**-n**] [**-s**] [**-q**]

Flags

-l*Name*	Specifies the name in the **xroute** table that should be listened for.
-n	Runs the **xtalk** process in the background to listen for calls and notify you when they arrive. When a call arrives, you can start the **xtalk** process again in the foreground and then choose whether to accept or reject the call.
-s	Bypasses (does not produce) the initial copyright screen.
-q	Removes copyright screen after showing it. User not required to press Enter.

Description

The **xtalk** command enables you to initiate or receive calls over SVCs and communicate with another user by typing messages or sending and receiving files. **xtalk** can be used on any port that has the COMIO emulation enabled.

Note: The **xtalk** command works over X.25 ports that have COMIO emulation configured.

The **xtalk** command has some features that are analogous to using a telephone. To have a conversation or transfer files, one party must first make a call and the other party must receive and accept the call. The **xtalk** command enables you to do the following:

- Listen for calls.
- Make a call.
- Receive a call.
- Have a conversation.
- Transfer files.
- End a call.

You can store the details of the systems you want to communicate with under a symbolic name in an address list. The **xtalk** command allows you to view, change, add, or delete entries in this list.

The **xtalk -n** command runs the **xtalk** process in the background to listen for calls and notify you when they arrive. When a call arrives, you can start the **xtalk** process again in the foreground and then choose whether to accept or reject the call.

The **xtalk -l** *EntryName* command listens for calls for the routing list entry specified by the *EntryName* parameter.

Two other flags enable you to manipulate the display of the title screen for the **xtalk** menu program. The **xtalk -q** command displays the title screen for two seconds; the **xtalk -s** command suppresses the display of the title screen.

To converse, each user types messages. Your messages and the other user's messages appear on your display. You can record in a log file the messages you exchange during a conversation.

Starting the xtalk Command

Both hosts must have the **xtalk** command running in order to communicate. The calling host runs the **xtalk** command to initiate a call. The called host runs the **xtalk** command, either in the foreground or the background, to listen for incoming calls from other users. The same command can be used for both the calling and called host. Enter the following:

```
xtalk -s -l IBMXTALK
```

where **IBMXTALK** is the identification of a record in the routing table giving the characteristics of the incoming calls that must be routed to the **xtalk** application. The **IBMXTALK** entry is a default entry in the routing table supplied with the X.25 program.

If you want to run the **xtalk** command in the background to listen for incoming calls, enter the following:

```
xtalk -n -l IBMXTALK
```

The **xtalk** command displays a screen with the following options:

TALK	Allows you to make calls to another user.
ADD	Permits the addition of new entries in the system address list or in your local address list.
BROWSE	Enables you to see all of the data associated with a name in either address list.
CHANGE	Allows you to change the information for an entry in the address lists.
DELETE	Allows you to delete an entry in the address lists.
QUIT	Ends an **xtalk** session.

The **xtalk** screen also displays the first few names in the system address list and the local address list (if one exists).

Making a Call

The **TALK** option on the **xtalk** main screen allows you to exchange messages with another user. You can specify the user you want to call in one of the following ways:

- Use the Up and Down arrow keys to scroll to the desired name in the local or system address list. To switch between the two lists, use the

F2: **TALK**

- Use the Left and Right arrow keys to select the **TALK** option, and then press

Enter:

Note: If you type a name and an X.25 port, the **xtalk** command uses the port you type instead of the port specified in the address list entry for that name.

- Use the Left and Right arrow keys to select the **TALK** option, and then press

Enter:

Accepting a Call

If you have the **xtalk** command running in the background, the command sends you a message when there is an incoming call. You must have the **xtalk** command running in the foreground to accept a call. When the **xtalk** command is running in the foreground, the command displays a screen listing the caller, the caller's address, and the COMIO emulator port. The screen gives you the option to accept or reject the call. Select **ACCEPT** to receive the incoming call.

Exchanging Messages

When a user accepts an incoming call, the **xtalk** command displays a commands screen on both the calling and the called host. The options on the commands screen are as follows:

TRANSFER FILE
BEGIN LOGGING
END LOGGING
CHANGE LOG FILENAME
QUIT CALL

With the commands screen options, you can choose to transfer files or log the messages between hosts. To exchange messages, press the F2 key to switch to the message screen. The messages typed by both users appear on the message screen of each host.

Transferring Files

To transfer a file with the **xtalk** command, one user sends the file, and the other user can choose to accept or reject it. If a file of the same name already exists on the recipient's system, the recipient can choose to append or overwrite the existing file, or save the transferred file under a new name.

To transfer files, you must first make or receive a call by using the **xtalk** command. When a call is accepted, the **xtalk** command displays the commands screen or the message screen on both hosts.

If you are sending a file to another user and the message screen is displayed on your machine, press the F2 key to display the commands screen. Use the Up or Down arrow key to select the **FILE TRANSFER** option, and then press the

Enter: If you are receiving a file and the commands screen is displayed on your machine, press the F2 key to display the message screen. To receive the file, use the Left or Right arrow key to select the

ACCEPT option. The **xtalk** command displays a screen containing the name of the file. You can change the name of the file being transferred by typing a new name in this screen.

Once a file transfer has been accepted by the called user, either user can cancel the file transfer by pressing the Esc (Escape) key.

Creating Address List Entries

Each user that you can talk to over an X.25 network has a network user address (similar to a telephone number). So that users can make calls without knowing another user's network address, the **xtalk** command maintains a system address list that is available to all users. The system address list is stored in the **/etc/xtalk.names** file. You must have root user permissions to change the entries in the system address list.

Each user can also choose to keep a local address list containing modifications and additions to the system address list. An entry for a user in your local address list overrides the entry for the same user in the system address list. The **xtalk** command stores entries for a local address list in the **$HOME/xtalk.names** file.

To make an entry in an address list, select the **ADD** option from the **xtalk** Main menu screen. The Add screen contains the following fields:

Name	Enter a name of up to 15 characters. The name must start with a letter and consist of letters and digits.
Port	Enter a valid COMIO emulation port name of the form x25s*nn*, where *nn* is the port number. The port name must be the name of one of the COMIO emulators configured onto one of the X.25 ports.
Address	Enter a valid network user address (NUA) of up to 15 decimal digits.
Extended Address	Enter a valid extension of up to 32 digits. The extension address is valid only on networks implementing the 1984 version of X.25. This field is optional.
Facilities	Enter up to 32 hexadecimal characters that represent 16 bytes transmitted in the facilities field in the call request packet. This field is optional.

Making a Call

This procedure uses the **xtalk** command to make a call on an SVC from a DTE named Host1 to a DTE named Host2.

Note: In the following procedure, **IBMXTALK** identifies a record in the routing table giving the characteristics of the incoming calls that must be routed to the **xtalk** application.

1. On Host2, load **xtalk** to listen to the **IBMXTALK** entry in the routing table. Enter the following:
   ```
   xtalk -s -l IBMXTALK
   ```
2. On Host1:
   a. Load the **xtalk** command. Enter the following**:**
      ```
      xtalk -s -l IBMXTALK
      ```
   b. Use the **ADD** option to create an entry in the address list for Host2.
   c. Select the **TALK** option to call Host2.
3. On Host2, accept the incoming call, and then press the F2 key.
4. On Host1, press the F2 key and enter a message. It is transmitted on the network then displayed on Host 2.

   Notes:

   a. If you have only one system, open two windows (or two sessions). In the first window, start one copy of the **xtalk** command that corresponds to the called machine and that will listen for incoming calls whose characteristics are defined in the routing table entry **IBMXTALK**:

```
xtalk -s -l IBMXTALK
```

In the other window, start the **xtalk** command without a routing parameter:

```
xtalk -s
```

Ignore the warning message `You cannot receive incoming calls` by pressing the Esc key. You can now proceed from this window the same way as explained for the system Host1.

b. The **xtalk** command cannot be used to test permanent virtual circuits (PVCs). Use the sample programs instead. These programs are located in the **/usr/lpp/bosext2/x25app/samples** directory and in **/usr/samples/sx25/comio** for AIX Version 4 users.

Problems

The **xtalk** command indicates a device driver problem by displaying a message that begins with `CIO Status`, as in the following message:

```
CIO Status 68 - X25_NAME_USED
The name is already being listened to.
```

These messages correspond to device driver return codes. If you have already tested the connection to the network, the most likely error messages are those related to the routing table:

`CIO Status 68 - X25_NAME_USED`	The name is already being listened to. Another copy of the **xtalk** command is probably running in the background and listening to **IBMXTALK**. Remember to always use the **QUIT** option to end the **xtalk** session and not the F3 key.
`CIO Status 77 - X25_TABLE`	Could not update routing list. A copy of the **xroute** command is probably still running or a lock file **xroute.lck** has been left in the **/etc/locks** directory.
`CIO Status 73 - X25_NO_NAME`	There is no such name in the routing list. The **-l** parameter does not match an entry in the routing list.

Two other problems are also related to the routing:

- The call has been cleared with cause 00 and diagnostic 00.

  Means that the call has reached the remote system but has not been transmitted to an application. Either the application (the **xtalk** command) is not loaded, or it has not been started with the **-l IBMXTALK** flag, or the **IBMXTALK** entry in the routing table doesn't exist or doesn't contain a CUD matching the one in the incoming call packet (**FD**).

- The call has been cleared with cause 00 and diagnostic F4.

  This is a diagnostic generated by the **xtalk** command on the called DTE meaning that the **xtalk** command received the call but was not ready to accept it (not on the main menu).

If you get a message like the following, you have an X.25 protocol problem.

```
The call has been cleared with cause 13
and diagnostic 43.
```

The cause and diagnostic codes are hexadecimal values indicating the cause of the problem. These codes may be generated by the local system, the PSDN or the remote DTE.

Appendix B. COMIO Emulator

The library API is provided for applications that were written to the user space API provided with the base AIX Version 3 X.25 support. The API is not intended for new program development. To allow the use of this API, a COMIO emulation port must be configured on to the X.25 ports to be used. Applications running on AIX Version 3 with the base X.25 product must be recompiled on AIX Version 4 with the AIXlink/X.25 product before they can be run over a COMIO emulation port.

- x25_ack Subroutine
- x25_call Subroutine
- x25_call_accept Subroutine
- x25_call_clear Subroutine
- x25_circuit_query Subroutine
- x25_ctr_get Subroutine
- x25_ctr_remove Subroutine
- x25_ctr_test Subroutine
- x25_ctr_wait Subroutine
- x25_deafen Subroutine
- x25_device_query Subroutine
- x25_init Subroutine
- x25_interrupt Subroutine
- x25_link_query Subroutine
- x25_listen Subroutine
- x25_pvc_alloc Subroutine
- x25_pvc_free Subroutine
- x25_receive Subroutine
- x25_reset Subroutine
- x25_reset_confirm Subroutine
- x25_send Subroutine
- x25_term Subroutine

x25_ack Subroutine

Purpose

Acknowledges data received with the D-bit set.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_ack(conn_id)
int  conn_id;
```

Description

The **x25_ack** subroutine sends an acknowledgment for the data packet most recently received with the D-bit set for the call specified by the *conn_id* parameter.

Control is returned to the calling application when the adapter has queued the packet for transmission.

Parameters

conn_id Connection identifier of the call.

Return Values

If successful, the **x25_ack** subroutine returns a value of 0. If an error occurs, the **x25_ack** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADCONNID

X25NOACKREQ

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25SYSERR

X25RESETCLEAR

X25TRUNCTX

If the **x25_errno** global variable is set to a **X25SYSERR** value, the **errno** global variable is set to one of the following values:

ENOSPC

EINTR

EIO

Examples

Acknowledge data received with the D-bit set: example program **svcrcv**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_send** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags in *AIX 5L Version 5.2 Communications Programming Concepts*.

x25_call Subroutine

Purpose
Makes an X.25 call by setting up a switched virtual circuit (SVC).

Library
X.25 Communications Library (**libx25s.a**)

C Syntax
```
int x25_call(cb_call, ctr_id)
struct cb_call_struct * cb_call,
int  ctr_id;
```

Description
The **x25_call** subroutine sets up a switched virtual circuit (SVC) for the X.25 port specified in **cb_call_struct** for an X.25 call between the calling address and called address, also specified in the **cb_call_struct** structure.

Control is returned to the application as soon as the call-request packet has been transmitted, but the SVC is not established until a call-connected packet is received (using the **x25_receive** subroutine).

Optional facilities, such as fast-select calls, can be requested by entering the correct values in the **cb_fac_struct** structure. If the facilities requested are not allowed by the network, the call is cleared and an appropriate error code is made available in the **cb_clear_struct** structure, which can be received using the **x25_receive** subroutine.

Parameters

cb_call	Pointer to the **cb_call_struct** structure.
ctr_id	Identifier of a counter allocated by a previous **x25_ctr_get** subroutine.

Return Values
If successful, the **x25_call** subroutine returns the connection identifier to be used by other subroutines for the duration of the call. If an error occurs, or the call is cleared, the **x25_call** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes
On failure, the **x25_errno** global variable is set to one of the following:

X25CALLED

X25CALLING

X25INVCTR

X25INVFAC

X25LONG

X25NOCARD

X25NOLINK

X25NOSUCHLINK

X25NOTINIT

X25PROTOCOL

X25SYSERR

X25TOOMANYVCS

X25TRUNCTX

If **x25_errno** is set to **X25SYSERR**, **errno** is set to one of the following values:

EINTR

EIO

ENOSPC

Examples
Make a call: example program **svcxmit**.

Implementation Specifics
This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information
The **x25_call_accept** subroutine, **x25_call_clear** subroutine, **x25_ctr_get** subroutine, **x25_receive** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_call_accept Subroutine

Purpose
Accepts an incoming call.

Library
X.25 Communications Library (**libx25s.a**)

C Syntax
```
int x25_call_accept(conn_id, cb_call, ctr_id)
int  conn_id;
struct cb_call_struct * cb_call;
int  ctr_id;
```

Description

The **x25_call_accept** subroutine accepts an incoming call by generating and sending a call-accepted packet. It then returns control to the application. If the facilities requested are not allowed by the network, the call is cleared and an appropriate error code is made available in a later **cb_clear_struct** control block.

Parameters

conn_id	Connection identifier of the call.
cb_call	Pointer to the call control block, the **cb_call_struct** structure.
ctr_id	Identifier of a counter allocated by a previous **x25_ctr_get** subroutine, to be associated with this call.

Return Values

If successful, the **x25_call_accept** subroutine returns a value of 0. If an error occurs, the **x25_call_accept** subroutine returns a value of -1 and sets the **x25_errno** global value to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global value is set to one of the following values:

X25BADCONNID

X25CALLED

X25CALLING

X25INVCTR

X25INVFAC

X25LONG

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25RESETCLEAR

X25SYSERR

X25TRUNCTX

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

ENOSPC

Examples

Accept an incoming call: example program **svcrcv**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_call** subroutine, **x25_call_clear** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_call_clear Subroutine

Purpose

Clears a call.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_call_clear (conn_id, cb_clear, cb_msg)
int  conn_id;
struct cb_clear_struct * cb_clear;
struct cb_msg_struct * cb_msg;
```

Description

The **x25_call_clear** subroutine clears a call by generating and sending a clear request packet. Control is not returned to the application until a clear confirmation or a clear indication packet has been received.

A call is cleared by disconnecting a connected call or rejecting a call that has not been accepted.

Parameters

conn_id	Specifies the Connection identifier of the call.
cb_clear	Indicates the pointer to the clear structure, **cb_clear_struct**.
cb_msg	Indicates the pointer to the message structure, **cb_msg_struct**. This structure is used to return information from the clear confirmation packet. The application must interpret the appropriate structure to access the message. This structure is allocated by the API. (It is the responsibility of the application to free this memory). If the **cb_msg** value is set to a null value, no clear confirmation information is returned.

Return Values

If successful, the **x25_call_clear** subroutine returns a value of 0. If an error occurs, the **x25_call_clear** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADCONNID

X25CALLED

X25CALLING

X25LONG

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25SYSERR

X25RESETCLEAR

X25TRUNCTX

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

ENOSPC

Examples

Terminate (clear) a call: example program **svcxmit**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_call** subroutine, **x25_call_accept** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_circuit_query Subroutine

Purpose

Returns configuration information about a virtual circuit.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
struct cb_circuit_info_struct *x25_circuit_query(conn_id)
int  conn_id;
```

Description

The **x25_circuit_query** subroutine returns the current information about the specified virtual circuit in **cb_circuit_info_struct**.

Parameters

conn_id Connection identifier of the call currently using the virtual circuit.

Return Values

If successful, the **x25_circuit_query** subroutine returns a pointer to **cb_circuit_info_struct**, the structure containing the information. Storage for this structure is allocated by the API; it is the responsibility of the application to free it. If an error occurs, the **x25_circuit_query** subroutine returns a NULL value and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADCONNID

X25NOLINK

X25NOTINIT

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to **ENOMEM**.

Examples

The following code structure prints current information for the virtual circuit identified by **conn_id**:

```
struct cb_circuit_info_struct *cct_ptr;
cct_ptr = x25_circuit_query(conn_id);
if (cct_ptr == NULL)
  (void)printf("Error %d from x25_circuit_query.",x25_errno);
else
{
  if (cct_ptr -> flags & X25FLG_LCN)
   (void)printf("Logical Channel Number (LCN) : %d\n",cct_ptr -> lcn);
  if (cct_ptr -> flags & X25FLG_INCOMING_PACKET_SIZE)
    (void)printf("Incoming Packet Size : %d\n",
                 cct_ptr -> incoming_packet_size);
  if (cct_ptr -> flags & X25FLG_OUTGOING_PACKET_SIZE)
    (void)printf("Outgoing Packet Size : %d\n",
                 cct_ptr -> outgoing_packet_size);
  if (cct_ptr -> flags & X25FLG_INCOMING_THROUGHPUT_CLASS)
    (void)printf("Incoming throughput class : %d\n",
                 cct_ptr -> incoming_throughput_class);
```

```
  if (cct_ptr -> flags & X25FLG_OUTGOING_THROUGHPUT_CLASS)
    (void)printf("Outgoing throughput class : %d\n",
                 cct_ptr -> outgoing_throughput_class);
  if (cct_ptr -> flags & X25FLG_INCOMING_WINDOW_SIZE)
    (void)printf("Incoming window size : %d\n",
                 cct_ptr -> incoming_window_size);
  if (cct_ptr -> flags & X25FLG_OUTGOING_WINDOW_SIZE)
    (void)printf("Outgoing window size : %d\n",
                 cct_ptr -> outgoing_window_size);
  free(cct_ptr);
}
```

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_device_query** subroutine, the **x25_link_query** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_ctr_get Subroutine

Purpose

Gets a counter.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_ctr_get(void)
```

Description

The **x25_ctr_get** subroutine allocates a counter whose value is incremented when a message associated with it arrives. The counter is decremented when a message associated with it is received by an application.

Return Values

If successful, the **x25_ctr_get** subroutine returns the counter identifier. If an error occurs, the **x25_ctr_get** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25NOCTRS

X25NOTINIT

X25SYSERR

Examples

Get a counter: example program **svcxmit**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_ctr_remove** subroutine, **x25_ctr_test** subroutine, **x25_ctr_wait** subroutine.

Processing Calls with the X.25 API in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_ctr_remove Subroutine

Purpose

Removes a counter.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_ctr_remove(ctr_id)
int  ctr_id;
```

Description

The **x25_ctr_remove** subroutine removes the specified counter from the system. Only the application that requested the counter can remove it from the system. The counter cannot be removed if it has a non-zero value, which indicates that data is waiting to be read from an associated call. The counter identifier may be reused by a future call to the **x25_ctr_get** subroutine.

Parameters

ctr_id Contains the counter identifier allocated by a previous **x25_ctr_get** subroutine call.

Return Values

If successful, the **x25_ctr_remove** subroutine returns a value of 0. If an error occurs, the **x25_ctr_remove** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following value:

X25AUTHCTR

X25CTRUSE

X25INVCTR

X25NOTINIT

X25SYSERR

Examples

See the example program **svcxmit** for a code sample that removes a counter.

Implementation Specifics

This command is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_ctr_get** subroutine, **x25_ctr_test** subroutine, **x25_ctr_wait** subroutine.

Processing Calls with the X.25 API in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_ctr_test Subroutine

Purpose

Returns the current value of a counter.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_ctr_test(ctr_id)
int  ctr_id;
```

Description

The **x25_ctr_test** subroutine returns the current value of an active counter for testing.

Parameters

ctr_id Contains the Counter identifier allocated by a previous **x25_ctr_get** subroutine.

Return Values

If successful, the **x25_ctr_test** subroutine returns the current value of the counter. If an error occurs, the **x25_ctr_test** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25INVCTR

X25NOTINIT

X25SYSERR

Examples

The following code structure determines how many messages for a call are waiting to be received, assuming an array of calls information is available:

```
ctr_id = calls[i].counter_id;
number_of_messages = x25_ctr_test(ctr_id);
if (number_of_messages == 0)
    (void)printf("There are no messages waiting\n");
else if (number_of_messages > 0)
    (void)printf("The number of messages waiting is %d\n",
                 number_of_messages);
```

Note: The array used here is not part of the X.25 API.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_ctr_get** subroutine, **x25_ctr_remove** subroutine, **x25_ctr_wait** subroutine.

Processing Calls with the X.25 API, X.25 Overview for Programming in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_ctr_wait Subroutine

Purpose

Waits for counters to change in value.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_ctr_wait(ctr_num, ctr_array)
int  ctr_num;
struct ctr_array_struct  ctr_array[ ];
```

Description

The **x25_ctr_wait** subroutine waits for the values of active counters to change. The process is suspended until the value of one of the counters is greater than the specified value. Setting this value in the application is optional, but recommended.

Parameters

ctr_num	Indicates the number of elements in the **ctr_array_struct** structure.
ctr_array	Indicates the contents of an array of structures.
ctr_id	Contains the counter identifier allocated by a previous **x25_ctr_get** subroutine.
ctr_value	Specifies the value that must be exceeded by this counter.

Return Values

If successful, the **x25_ctr_wait** subroutine returns the counter identifier (*ctr_id)* of the counter that satisfied the condition by exceeding the specified value. (If more than one counter exceeded its specified value, only one of the counter identifiers is returned.) If an error occurs, the **x25_ctr_wait** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25INVCTR

X25NOTINIT

X25SYSERR

Examples

1. See the example program **svcxmit** for an illustration of code statements that wait for a call to be connected or cleared.
2. See the example program **svcrcv** for an illustration of codes statements that wait for an incoming call.
3. See the example program **svcrcv** for an illustration of code statements that wait for data or some other message.

Implementation Specifics

This command is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_ctr_get** subroutine, **x25_ctr_remove** subroutine **x25_ctr_test** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_deafen Subroutine

Purpose

Turns off listening for incoming calls.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_deafen(listen_id)
int  listen_id;
```

Description

The **x25_deafen** subroutine turns off listening for incoming calls. It stops routing the calls that the application was listening for using the specified the *listen_id* parameter.

Parameters

listen_id Contains the listen identifier returned from a previous **x25_listen** subroutine.

Return Values

If successful, the **x25_deafen** subroutine returns a value of 0. If an error occurs, the **x25_deafen** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADLISTENID

X25NOTINIT

X25SYSERR

X25TIMEOUT

Examples

Stop listening: example program **svcrcv**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_listen** subroutine.

Processing Calls with the X.25 API in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_device_query Subroutine

Purpose

Returns configuration information about a device.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
struct cb_dev_info_struct *x25_device_query(link_name)
struct cb_link_name_struct * link_name;
```

Description

The **x25_device_query** subroutine returns information about the X.25 adapter in the **cb_dev_info_struct** structure.

The information entered when the adapter was configured is returned. Changes made to a particular switched virtual circuit (SVC) by requests entered in the facilities fields of X.25 API structures are not reflected by this subroutine; these values can be obtained using the **x25_circuit_query** subroutine.

Parameters

link_name Specifies a pointer to the **cb_link_name_struct** structure, which gives the name of the X.25 port.

Return Values

If successful, the **x25_device_query** subroutine returns a pointer to **cb_dev_info_struct**, structure containing the information. The storage for this structure is allocated by the API; it is the responsibility of the application to free it. If an error occurs, the **x25_device_query** subroutine returns a null value and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25NOTINIT

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to **ENOMEM**.

Examples

The following code sample prints out the number of PVCs and the default and maximum packet sizes for an X.25 port:

```
struct cb_dev_info_struct *dev_ptr;
dev_ptr = x25_device_query(&link_name);
if (dev_ptr == NULL)
  (void)printf("Error %d from x25_device_query.",x25_errno);
else
{
  if (dev_ptr -> flags & X25FLG_NUA)
  {
    (void)printf("NUA : %s\n",dev_ptr -> nua);
    free(dev_ptr -> nua);
  }
  if (dev_ptr -> flags & X25FLG_NO_OF_VCS)
    (void)printf("Number of PVCs : %d\n",dev_ptr -> no_of_vcs);
  if (dev_ptr -> flags & X25FLG_MAX_RX_PACKET_SIZE)
    (void)printf("Max receive pkt size : %d\n",
      dev_ptr -> max_rx_packet_size);
  if (dev_ptr -> flags & X25FLG_MAX_TX_PACKET_SIZE)
    (void)printf("Max transmit pkt size : %d\n",
      dev-ptr -> max_tx_packet_size);
  if (dev_ptr -> flags & X25FLG_DEFAULT_SVC_RX_PACKET_SIZE)
    (void)printf("Default receive pkt size : %d\n",
      dev_ptr -> default_svc_rx_packet_size);
  if (dev_ptr -> flags & X25FLG_DEFAULT_SVC_TX_PACKET_SIZE)
    (void)printf("Default transmit pkt size : %d\n",
      dev_ptr -> default_svc_tx_packet_size);

  free(dev_ptr);
}
```

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_circuit_query** subroutine, **x25_link_query** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_init Subroutine

Purpose

Initializes the X.25 application programming interface (API).

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

int x25_init(*link_name***)**
struct cb_link_name_struct * *link_name***;**

Description

The **x25_init** subroutine establishes communications with the X.25 device driver to set up X.25 communications with the X.25 port named by the *link_name* parameter. The application must invoke the **x25_init** subroutine before any other X.25 subroutines.

> **Note:** Initializing a port does not guarantee that the port is connected (see the **x25_link_query** subroutine).

Parameters

link_name Indicates a pointer to the **cb_link_name_struct** structure, which gives the name of the X.25 port.

Return Values

If successful, the **x25_init** subroutine returns a value of 0. If an error occurs, the **x25_init** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25BADDEVICE

X25INIT

X25MAXDEVICE

X25NOSUCHLINK

X25SYSERR

Examples

Initialize the API for an X.25 port: example program **svcxmit**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_term** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_interrupt Subroutine

Purpose

Sends an interrupt packet.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_interrupt(conn_id, cb_int)
int  conn_id;
struct cb_int_struct * cb_int;
```

Description

The **x25_interrupt** subroutine sends an interrupt message. Control is returned to the application when the message has been received by the adapter.

Parameters

conn_id	Contains the connection identifier of the call.
cb_int	Contains a pointer to the **cb_int_struct** structure, which contains the interrupt data.

Return Values

If successful, the **x25_interrupt** subroutine returns a value of 0. If an error occurs, the **x25_interrupt** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25BADCONNID

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25RESETCLEAR

X25SYSERR

X25TRUNCTX

If the **x25_errno** global variable is set to a **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

ENOSPC

Examples

The following code statement ends an interrupt:

```
struct cb_int_struct int_data;
int_data.flags = X25FLG_INT_DATA;
int_data.data_len = 20;
int_data.int_data = "This is an interrupt";
```

```
rc = x25_interrupt(conn_id,&int_data);
if (rc < 0)
  (void)printf("Error %d from x25_interrupt.",x25_errno);
```

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

Processing Calls with the X.25 API , Using the X.25 Structures and Flags ,

in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_link_query Subroutine

Purpose

Returns information about the current status of an X.25 port.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

int x25_link_query(*link_name***)**
struct cb_link_name_struct * *link_name***;**

Description

The **x25_link_query** subroutine returns the status of the X.25 port as an integer.

Parameters

link_name Contains a pointer to the **cb_link_name_struct** structure, which gives the name of the X.25 port.

Return Values

If successful, the **x25_link_query** subroutine returns an integer that indicates a status of **X25_LINK_CONNECTED**, **X25_LINK_DISCONNECTED**, or **X25_LINK_CONNECTING**. If an error occurs, the **x25_link_query** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25NOCARD

X25NOTINIT

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

Examples

To find out whether port x25s1 is connected, disconnected, or connecting, use this code sample:

```
struct cb_link_name_struct link_name;
link_name.flags = X25FLG_LINK_NAME;
link_name.link_name = "x25s1";
rc = x25_link_query(&link_name);
switch (rc)
{
  case X25_LINK_CONNECTED:
    (void)printf("Link is connected\n");
    break;
  case X25_LINK_DISCONNECTED:
    (void)printf("Link is disconnected\n");
    break;
  case X25_LINK_CONNECTING:
    (void)printf("Link is connecting\n");
    break;
    case -1;
      switch (x25_errno);
      {
        case X25SYSERR:
          (void)printf("System error : errno = %d\n",errno);
           perror();
           break;
        case X25NOCARD:
          (void)printf("The X.25 adapter is either not\n");
          (void)printf("installed or not functioning:");
          (void)printf("Call your system administrator.\n");
           break;
        case X25NOTINIT:
          (void)printf("The application has not initialized\n",
          (void)printf("X.25 communications:");
          (void)printf("Call your system administrator.\n");
           break;
        }
    break;
}
```

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_circuit_query** subroutine, **x25_device_query** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_listen Subroutine

Purpose

Starts listening for incoming calls.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_listen(name, ctr_id)
NLchar * name;
int  ctr_id;
```

Description

The **x25_listen** subroutine starts listening for incoming calls that fit the criteria in the routing list entry with the specified *name* parameter. It also tells the API to associate the calls with the counter identifier specified. It returns a listen identifier to be used by the **x25_receive** subroutine.

Parameters

name	Contains a pointer to a name that is specified in the routing list.
ctr_id	Identifies a counter, allocated by a previous **x25_ctr_get** subroutine.

Return Values

If successful, the **x25_listen** subroutine returns the listen identifier. If an error occurs, the **x25_listen** subroutine returns a value of -1 and sets the **x25_errno** global variable global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25AUTHLISTEN

X25INVCTR

X25NAMEUSED

X25NOLINK

X25NONAME

X25NOTINIT

X25SYSERR

X25TABLE

X25TIMEOUT

Examples

Start listening for incoming calls: example program **svcrcv**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_deafen** subroutine.

Processing Calls with the X.25 API in *AIX 5L Version 5.2 Communications Programming Concepts*.

x25_pvc_alloc Subroutine

Purpose
Allocates a permanent virtual circuit (PVC) for use by an application.

Library
X.25 Communications Library (**libx25s.a**)

C Syntax
```
int x25_pvc_alloc(pvc_ptr, ctr_id)
struct cb_pvc_alloc_struct * pvc_ptr;
int  ctr_id;
```

Description
The **x25_pvc_alloc** subroutine reserves the use of the specified permanent virtual circuit (PVC) for an application.

Parameters
pvc_ptr	Contains a pointer to the **cb_pvc_alloc_struct** structure, which contains the name of the X.25 port and the logical channel number of the PVC to be used. Together, the port and the logical number, identify the PVC.
ctr_id	Identifies a counter allocated by a previous **x25_ctr_get** subroutine.

Return Values
If successful, the **x25_pvc_alloc** subroutine returns the connection identifier to be used by other subroutines. If an error occurs, the **x25_pvc_alloc** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes
On failure, the **x25_errno** global variable is set to one of the following:

X25INVCTR

X25NOCARD

X25NOLINK

X25NOSUCHLINK

X25NOTINIT

X25NOTPVC

X25PVCUSED

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

Examples

The example program, **pvcxmit**, illustrates code statements that allocate a PVC.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_pvc_free** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_pvc_free Subroutine

Purpose

Frees a permanent virtual circuit (PVC).

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_pvc_free(conn_id)
int  conn_id;
```

Description

The **x25_pvc_free** subroutine frees the permanent virtual circuit (PVC) used for the specified connection for use by another application. Any data queued for the **x25_receive** subroutine is lost. It is the responsibility of the application to check the counter identifier for queued data before freeing the PVC.

Parameters

conn_id Contains the connection identifier, returned by the previous **x25_pvc_alloc** subroutine.

Return Values

If successful, the **x25_pvc_free** subroutine returns a value of 0. If an error occurs, the **x25_pvc_free** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADCONNID

X25NOCARD

X25NOLINK

X25NOTINIT

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

Examples
Free a PVC: example program **pvcxmit**.

Implementation Specifics
This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information
The **x25_pvc_alloc** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_receive Subroutine

Purpose
Receives an incoming packet and indicates the packet type.

Library
X.25 Communications Library (**libx25s.a**)

C Syntax
```
int x25_receive(conn_id, cb_msg)
int * conn_id;
struct cb_msg_struct * cb_msg;
```

Description
The **x25_receive** subroutine is used to receive incoming calls and messages and monitor data for connected calls. One **x25_receive** subroutine call receives a complete packet sequence. In the event that an interrupt packet is received, an interrupt confirmation is sent automatically by the system.

Parameters

conn_id Contains a pointer to an integer that contains the listen identifier.

> **Note:** If the call is successfully received, an open file is created in the current process that is used by the API library functions for subsequent communication on this connection. The file is closed when the call is cleared.

To receive a message for any connected call, a pointer to an integer that contains a value of 0.

To receive a message for a specific connected call, a pointer to an integer that contains the connection identifier of the call.

To receive monitoring data for a call, a pointer to an integer that contains the connection identifier returned by the **x25_link_monitor** subroutine.

On return from this subroutine, in all cases, a pointer to an integer that now contains the actual connection identifier.

cb_msg Specifies a pointer to the message structure, **cb_msg_struct**, which includes the **msg_type** field. This structure is allocated by the API; it is the responsibility of the application to free this memory.

Return Values

If successful, the **x25_receive** subroutine returns a nonnegative value. If an error occurs, the **x25_receive** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following values:

X25BADID

X25NOACKREQ

X25NOCARD

X25NODATA

X25NOLINK

X25NOTINIT

X25RESETCLEAR

X25SYSERR

X25TRUNCTX

If the **x25_errno** global variable is set to a **X25SYSERR** value, the **errno** global variable is set to an **EINTR** value.

Examples

1. See the example program **svcrcv** for a code sample that receives an incoming call.
2. See the example program **svcrcv** for a code sample that receives data or some other message.
3. See the example program **svcxmit** for a code sample that receives an acknowledgment that data has been received.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_send** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_reset Subroutine

Purpose

Resynchronizes communications on a virtual circuit.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_reset(conn_id, cb_res)
int  conn_id;
struct cb_res_struct * cb_res;
```

Description

The **x25_reset** subroutine sends a reset-indication packet to reset the virtual circuit using the specified connection identifier.

If the application is sending data at the time this subroutine is called, the data is flushed from the system, and the **x25_send** subroutine returns an appropriate error code. Incoming data not already passed to the application will be flushed. Because resets can cause data to be lost, it is the responsibility of the application to provide higher-level protocol for data protection.

Parameters

conn_id	Contains the connection identifier of the call.
cb_res	Specifies a pointer to the **cb_res_struct** structure, which is used to pass the reset cause and diagnostic codes.

Return Values

If successful, the **x25_reset** subroutine returns a value of 0. If an error occurs, the **x25_reset** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following value:

X25BADCONNID

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25RESETCLEAR

X25SYSERR

If the **x25_errno** global variable is set to the **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

ENOSPC

Examples
See the example program **pvcxmit** for a code sample that resets a call.

Implementation Specifics
This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information
The **x25_reset_confirm** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_reset_confirm Subroutine

Purpose
Confirms that a reset-indication packet has been received.

Library
X.25 Communications Library (**libx25s.a**)

C Syntax
```
int x25_reset_confirm(conn_id)
int  conn_id;
```

Description
The **x25_reset_confirm** subroutine sends a reset-confirmation packet. After the reset-indication packet has been received by the **x25_receive** subroutine, no further data can be sent or received until the reset-confirmation has been sent. Any data currently in transmission is discarded with an appropriate return code.

Parameters
conn_id Contains the Connection identifier of the call.

Return Values

If successful, the **x25_reset_confirm** subroutine returns a value of 0. If an error occurs, the **x25_reset_confirm** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25BADCONNID

X25NOACKREQ

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25RESETCLEAR

X25SYSERR

X25TRUNCTX

If the **x25_errno** global variable is set to a **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EINTR

EIO

ENOSPC

Examples

The example program **pvcrcv** illustrates how to confirm that a reset indication has arrived.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_reset** subroutine.

Processing Calls with the X.25 API in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_send Subroutine

Purpose

Sends a data packet.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_send(conn_id, cb_data)
int  conn_id;
struct cb_data_struct * cb_data;
```

Description

The **x25_send** subroutine transfers the data packet to the adapter for transmission across the network. Control is returned to the calling application when the device driver indicates successful data transfer to the adapter. If there is no room for the packet in adapter memory, the **x25_send** subroutine waits until memory becomes available. The amount of memory available depends on the transmit packet window and the transmit packet size. More memory generally becomes available after the X.25 network sends a Receiver Ready signal for the connection identifier.

Parameters

conn_id	Contains the connection identifier of the call.
cb_data	Specifies a pointer to data structure, **cb_data_struct**.

Return Values

If successful, the **x25_send** subroutine returns a value of 0. If an error occurs, the **x25_send** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25BADCONNID

X25NOACKREQ

X25NOCARD

X25NOLINK

X25NOTINIT

X25PROTOCOL

X25RESETCLEAR

X25SYSERR

X25TRUNCTX

If the **x25_errno** global variable is set to a **X25SYSERR** value, the **errno** global variable is set to one of the following values:

EFAULT

EINTR

EIO

ENOSPC

Examples

1. For code statements that send data without the D-bit set; see the example program **svcxmit**.
2. For code statements that send data with the D-bit set to request acknowledgment, see the example program **svcxmit**.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_ack** subroutine, **x25_receive** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

---

x25_term Subroutine

Purpose

Terminates the X.25 API for a specified X.25 port.

Library

X.25 Communications Library (**libx25s.a**)

C Syntax

```
int x25_term(link_name)
struct cb_link_name_struct * link_name;
```

Description

The **x25_term** subroutine terminates communication with the X.25 device driver to stop X.25 communications with the X.25 port named by the **link_name** subroutine. If this is the last X.25 port open for this process, X.25 resources are freed.

The **x25_term** subroutine clears any virtual circuits still being used by the application. However, it is recommended that you clear virtual circuits in an orderly manner before invoking the **x25_term** subroutine.

Parameters

link_name Contains the pointer to the **cb_link_name_struct** structure, which gives the name of the X.25 port.

Return Values

If successful, the **x25_term** subroutine returns a value of 0. If an error occurs, the **x25_term** subroutine returns a value of -1 and sets the **x25_errno** global variable to one of the error codes shown below.

Error Codes

On failure, the **x25_errno** global variable is set to one of the following:

X25BADDEVICE

Examples

The example program **svcxmit** illustrates code statements that terminates an API.

Implementation Specifics

This subroutine is part of X.25 Application in Base Operating System (BOS) Extensions 2.

Related Information

The **x25_init** subroutine.

Processing Calls with the X.25 API, Using the X.25 Structures and Flags, in *AIX 5L Version 5.2 Communications Programming Concepts*.

Appendix C. Device Handler API

Device Driver Emulation

The emulation of the old driver provides the following programming interface routines:

- x25sclose X.25 Device Handler Entry Point
- x25sioctl X.25 Device Handler Entry Point
- CIO_GET_STAT (Get Status) x25sioctl X.25 Device Handler Operation
- CIO_HALT (Halt Session) x25sioctl X.25 Device Handler Operation
- CIO_QUERY (Query Device) x25sioctl X.25 Device Handler Operation
- CIO_START (Start Session) x25sioctl X.25 Device Handler Operation
- IOCINFO (Identify Device) x25sioctl X.25 Device Handler Operation
- X25_ADD_ROUTER_ID (Add Router ID) x25sioctl X.25 Device Handler Operation
- X25_COUNTER_GET (Get Counter) x25sioctl X.25 Device Handler Operation
- X25_COUNTER_READ (Read Counter) x25sioctl X.25 Device Handler Operation
- X25_COUNTER_REMOVE (Remove Counter) x25sioctl X.25 Device Handler Operation
- X25_COUNTER_WAIT (Wait Counter) x25sioctl X.25 Device Handler Operation
- X25_DELETE_ROUTER_ID (Delete Router ID) x25sioctl X.25 Device Handler Operation
- X25_LINK_STATUS (Link Status) x25sioctl Operation
- X25_LOCAL_BUSY (Local Busy) x25sioctl Operation
- X25_QUERY_ROUTER_ID (Query Router) ID x25sioctl Operation
- X25_QUERY_SESSION (Query Session) x25sioctl Operation
- X25_REJECT_CALL (Reject Call) x25sioctl Operation
- x25smpx X.25 Device Handler Entry Point
- x25sopen X.25 Device Handler Entry Point
- x25sread X.25 Device Handler Entry Point
- x25sselect X.25 Device Handler Entry Point
- x25swrite X.25 Device Handler Entry Point

x25sclose X.25 Device Handler Entry Point

Purpose

Closes an X.25 device handler channel.

Syntax

```
int x25sclose (devno, chan, ext)
dev_t  devno;
int  chan,  ext;
```

Parameters

devno	Specifies major and minor device numbers.
chan	Identifies the channel number assigned by the **x25smpx** entry point.
ext	Not used by the **x25sclose** entry point.

Description

The **x25sclose** entry point closes an X.25 device handler channel. For each channel opened by the **x25sopen** entry point, there must be a corresponding **x25sclose** entry point. When the X.25 device handler receives an **x25sclose** entry point, the device handler frees all internal data areas associated with the corresponding **x25sopen** entry point. In addition, any receive data for the indicated channel is purged.

> **Note:** The **x25sclose** entry point does not free the channel itself. The channel is freed by the **x25smpx** entry point, which the kernel calls immediately after the **x25sclose** entry point.

If the channel being closed is the only open channel for the minor device, the X.25 device handler does the following as well:

- Frees the interrupt level.
- Resets all static data to its initial state.

Before issuing the **x25sclose** entry point, the caller should issue a call to the **CIO_HALT** operation for each successful **CIO_START** operation. If the user does not call the **CIO_HALT** operation (for example, the call was invoked by the kernel after a user process ended abnormally), the X.25 device handler performs the **CIO_HALT** operation on all open sessions on the channel before continuing with the **x25sclose** function. The close purges all data waiting on the channel. No special clear data can be sent and any clear confirm data is lost.

> **Attention:** If the user does not call a **CIO_HALT**, it is possible data could be lost on the channel's open sessions.

Execution Environment

An **x25sclose** entry point can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to one of the following values:

EINTR Indicates that the close call was interrupted.
ENXIO Indicates that the channel was invalid.

Implementation Specifics

The **x25sclose** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25smpx** entry point, **x25sopen** entry point.

The **CIO_HALT** x25sioctl X.25 Device Handler Operation, **CIO_START** x25sioctl X.25 Device Handler Operation.

---

x25sioctl X.25 Device Handler Entry Point

Purpose

Provides various functions for controlling the X.25 device.

Syntax

```
int x25sioctl (devno, cmd, arg, devflag, chan, ext)
dev_t  devno;
int  cmd,  arg;
ulong  devflag;
int  chan,  ext;
```

Parameters

devno Specifies major and minor device numbers.

cmd Specifies which of the available **x25sioctl** operations is to be performed.

arg Identifies the address of the **x25sioctl** parameter block. The meaning of this parameter depends on the value of the *cmd* parameter.

devflag Indicates how the device was opened and whether the caller is a user- or kernel-mode process. This parameter accepts the following flags:

DAPPEND
Specifies open for appending. The X.25 handler ignores this flag.

DKERNEL
Indicates a kernel-mode process called the entry point. This flag is clear if a user-mode process called the entry point.

DNDELAY
Specifies the X.25 device handler performs nonblocking reads and writes. If this flag is not set, the X.25 device handler performs blocking reads and writes.

DREAD
Specifies open for reading. This is the default for the X.25 handler regardless of whether this flag is set.

DWRITE
Specifies open for writing. This is the default for the X.25 handler regardless of whether this flag is set.

chan Identifies the channel number assigned by the **x25smpx** entry point.

ext Specifies the extended system call parameter. The meaning of this parameter depends on the value of the *cmd* parameter.

Description

The **x25sioctl** X.25 device handler entry point provides various functions for controlling the X.25 device. The following are valid operations for the X.25 device:

CIO_DNLD Downloads tasks to the kernel.
CIO_GET_STAT Returns the next status block.
CIO_HALT Halts a session.
CIO_QUERY Returns the current RAS counter values.
CIO_START Starts a session and registers a network ID.
IOCINFO Returns a structure that describes the device.

In addition to the above standard operations, the **x25sioctl** operation supports the following X.25 specific commands:

X25_ADD_ROUTER_ID Adds a router ID.
X25_COUNTER_GET Gets a counter for asynchronous notification.
X25_COUNTER_READ Reads the value of a counter.
X25_COUNTER_REMOVE Removes a counter from the system.
X25_COUNTER_WAIT Waits for the contents of counters to change.

X25_DELETE_ROUTER_ID	Deletes a router ID.
X25_DIAG_IO_READ	Reads from an I/O register on the adapter.
X25_DIAG_IO_WRITE	Writes to an I/O register on the adapter.
X25_DIAG_MEM_READ	Reads memory from the adapter into a user's buffer.
X25_DIAG_MEM_WRITE	Writes to memory on the adapter from a user's buffer.
X25_DIAG_TASK	Downloads the diagnostics task onto the card.
X25_LINK_CONNECT	Connects a link.
X25_LINK_DISCONNECT	Disconnects a link.
X25_LINK_STATUS	Queries the status of a link.
X25_LOCAL_BUSY	Enables or disables receiving data packets on a port.
X25_QUERY_SESSION	Queries a session.
X25_QUERY_ROUTER_ID	Queries router ID.
X25_REJECT	Rejects a call.

Execution Environment

The **x25sioctl** entry point can be called from the process environment only.

Implementation Specifics

The **x25sioctl** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25smpx** entry point.

CIO_DNLD (Download Task) x25sioctl X.25 Device Handler Operation

Purpose

Downloads tasks to the kernel.

Description

Note: The **CIO_DNLD** operation can be called only by a user-mode process. The **DKERNEL** flag must be clear to run this operation.

The **CIO_DNLD** x25sioctl operation downloads tasks to the kernel. This routine is used to pass microcode between the configuration program and the device driver. Each call to this routine completely replaces any previous version of microcode stored in the device driver.

Notes:

1. The **CIO_DNLD** operation does not download the microcode to the card. It transfers the microcode into kernel memory so that the microcode is available when needed.
2. If the microcode for real-time control microcode (RCM), X.25, or diagnostics is not available, the code pointer should be set to null and the code length set to 0.

For the **CIO_DNLD** operation, the *arg* parameter points to an **x25_task** structure. This structure contains the following fields:

`x25_code`	Points to X.25 code.
`rcm_code`	Points to RCM.
`diagnostic_code`	Points to diagnostic code.
`x25length`	Specifies the length of the X.25 code.

`rcm_length`	Specifies the length of the RCM code.
`diagnostic_length`	Specifies the length of the diagnostic code.

Execution Environment

The **CIO_DNLD** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to the following value:

EFAULT Indicates that an invalid address was specified.

Implementation Specifics

The **CIO_DNLD** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** device handler entry point.

CIO_GET_STAT (Get Status) x25sioctl X.25 Device Handler Operation

Purpose

Returns the next status block.

Description

Note: Only a user-mode process can call the **CIO_GET_STAT** operation.

The **CIO_GET_STAT** x25sioctl operation reads the next status block available on the channel. This operation does not block. To call the **CIO_GET_STAT** operation, the **DKERNEL** flag must be clear, indicating a user-mode process. If a new status block is available for a kernel-mode process, the **stat_fn** kernel procedure defined by the **x25sopen** entry point is called.

For the **CIO_GET_STAT** operation, the *arg* parameter returns a pointer to a **status_block** structure as defined in the **/usr/include/sys/comio.h** file.

Status Blocks for the X.25 Device Handler

Status blocks are provided to user-mode and kernel-mode processes differently. Kernel-mode processes receive a status block when they are called using the **stat_fn** kernel procedure. This procedure is indicated when a channel is opened by the **x25sopen** entry point.

User-mode processes receive a status block whenever they issue a **CIO_GET_STAT** operation. In addition, a user-mode process can wait for the next available status block by issuing an **x25sselect** entry point with the **DPOLLPRI** event specified.

Status blocks can be solicited to indicate a completion of a previous command (such as a **CIO_START** operation). Status blocks can also be unsolicited to indicate some asynchronous event.

Status blocks contain a code field and possible options. The code field indicates the type of status block code (for example, **CIO_START_DONE**). The types of status blocks are:

- **CIO_HALT_DONE**
- **CIO_NULL_BLK**
- **CIO_START_DONE**
- **CIO_TX_DONE**
- **X25_REJECT_DONE**

CIO_HALT_DONE Status Block

This block is provided by the X.25 device handler when the **CIO_HALT** operation is complete:

`option[0]` Specifies the status or exception code. This option is set to **CIO_OK** if the start is successful. Otherwise, it is set to one of the status returns defined by the **CIO_HALT** operation.

`option[1]` Specifies the session ID and the network ID. The 2 high-order bytes contain the session ID, and the 2 low-order bytes contain the network ID.

`option[2]` Valid only if the **CIO_HALT** was used to issue a Clear Request. For a user-mode process, this option contains a pointer to the first byte of the process's data buffer.

For kernel-mode process, this option contains the **mbuf** structure passed with the **CIO_HALT** operation. The buffer described by the **mbuf** structure represents a Clear Confirm received in response to the Clear Request. If there was a clear collision, the buffer represents a Clear Indication that should be treated as an acknowledgment of the Clear Request as no Clear Confirm is subsequently received.

This option is null for other **CIO_HALT** operation types.

`option[3]` Not used.

CIO_NULL_BLK Status Block

This status block is returned whenever a status block is requested but none are available.

`option[0]` Not used
`option[1]` Not used
`option[2]` Not used
`option[3]` Not used

CIO_START_DONE Status Block

This block is provided by the X.25 device handler when the **CIO_START** operation is complete:

`option[0]` Specifies the status or exception code. This option is set to **CIO_OK** if the start is successful. Otherwise, it is set to one of the status returns defined by the **CIO_START** operation.

`option[1]` Specifies the session ID and the network ID. The 2 high-order bytes contain the session ID and the 2 low-order bytes contain the network ID.

`option[2]` Valid only for a session of type **SESSION_SVC_OUT**. For a user-mode process, this option contains a pointer to the first byte of the process's data buffer**.**

For kernel-mode process, this option contains the **mbuf** structure passed with the **CIO_START** operation. The **mbuf** structure describes a buffer that represents either a Call Connected or a Clear Indication received in response to a Call Request.

This option is null for other session types.

`option[3]` Not used.

CIO_TX_DONE Status Block

This block is provided by the X.25 device handler when a transmit request for which an acknowledgment was requested is complete:

`option[0]` Specifies the status or exception code. This option is set to **CIO_OK** if the start is successful. Otherwise, it is set to one of the status returns defined by the **x25swrite** entry point.

`option[1]` Specifies the write ID in the **write_extension** structure. This structure is passed by the **x25swrite** entry point.

`option[2]` Points to the first byte of a user-mode process data buffer or to the **mbuf** structure for a kernel-mode process. The **mbuf** structure is passed with the **x25swrite** entry point.

`option[3]` Specifies the session ID in the 2 high-order bytes and the network ID in the 2 low-order bytes.

X25_REJECT_DONE Status Block

This status block indicates that the **CIO_REJECT** operation is complete:

`option[0]` Specifies the status or exception code. This option is set to **CIO_OK** if the start is successful. Otherwise, it is set to one of the status returns defined by the **CIO_REJECT** operation.

`option[1]` Specifies the session ID in the 2 high-order bytes and the network ID in the 2 low-order bytes.

`option[2]` Identifies the call ID of the incoming call that is being rejected.

`option[3]` Not used.

Execution Environment

The **CIO_GET_STAT** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to the following value:

EFAULT Indicates an invalid address was specified.

Implementation Specifics

The **CIO_GET_STAT** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

Common Communications Status/Exception Codes in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

The **x25sioctl** entry point **x25sopen** entry point, **x25sselect** entry point, **x25swrite** entry point.

The **CIO_HALT** x25sioctl X.25 Device Handler Operation, **CIO_START** x25sioctl X.25 Device Handler Operation, **X25_REJECT** x25sioctl X.25 Device Handler Operation.

CIO_HALT (Halt Session) x25sioctl X.25 Device Handler Operation

Purpose

Ends a session with the X.25 device handler.

Description

The **CIO_HALT** x25sioctl operation ends a session with the X.25 device handler. A session identified by a particular `session_id` field can be terminated in any of the following ways:

- Clear request on an SVC
- Clear confirm on an SVC
- Request for deallocation of a PVC
- Request to stop listening for incoming calls
- Turning off the monitoring facilities on the card.

If the **CIO_HALT** operation is the last for this port, appropriate termination (such as automatic disconnection, if configured) is done.

The **CIO_HALT** operation returns immediately to the caller before the halt completes. If the return does not indicate an error, the X.25 device handler builds a **CIO_START_DONE** status block on completion of the operation. For kernel-mode processes the status block is passed to the associated status function, as specified by the **x25sopen** entry point. For user-mode processes, the block is placed in the associated status-and-exception queue.

Parameter Block

The parameter block for the **CIO_HALT** operation is the **x25_halt_data** structure. This structure contains the following fields:

`sb`	Indicates that the **session_blk** structure is defined in the **/usr/include/sys/comio.h** file. The `status` field in this structure has meaning when returned only if the return code is **EIO**.
`session_id`	Identifies the ID of the session to halt.

If the **CIO_HALT** operation is issued to send a Clear Request packet on a session of type **SESSION_SVC_OUT** or **SESSION_SVC_IN**, the **CIO_HALT** operation *ext* parameter is used. The *ext* parameter points to a buffer containing the data required for the Clear Request packet. This data is in the form described in the **mbuf** structure.

For a kernel-mode process, the data passed in the *ext* parameter is an **mbuf** pointer. Only the calling process can free the mbuf data returned in the **CIO_HALT_DONE** status block. The mbuf data returned by this status block is not the same as the data passed down.

For a user-mode process, the data passed in the *ext* parameter is a pointer to a buffer of the same format in user space. If the pointer is null, then the Clear Request packet is sent with default cause-and-diagnostic codes (0, 0), but with no facilities or user data. When the **CIO_HALT_DONE** status block is received, the buffer is filled with the contents of the Clear Confirm packet.

Execution Environment

The **CIO_HALT** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to one of the following values:

EFAULT	Indicates that an invalid address was specified.
EINVAL	Indicates invalid values in the *ext* parameter buffer.

EIO Indicates that an error has occurred. The `status` field in the **status_block** structure indicates one of the following exception codes:

- **CIO_HARD_FAIL**
- **CIO_LOST_DATA**
- **CIO_NOMBUF**
- **CIO_NOT_STARTED**
- **CIO_TIMEOUT**

In addition, one of the following X.25-specific codes may be returned:

X25_BAD_PKT_TYPE	Indicates that the packet type passed in the *ext* parameter is invalid. For session types **SESSION_SVC_OUT** or **SESSION_SVC_IN**, the packet type should be either **PKT_CLEAR_REQ** or **PKT_CLEAR_CONFIRM**.
X25_NO_LINK	Indicates that the link is not connected.
X25_PROTOCOL	Indicates that a protocol error occurred.

Implementation Specifics

The **CIO_HALT** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult the adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** entry point, **x25sopen** entry point.

The **CIO_START** x25sioctl X.25 Device Handler Operation.

Status Blocks for the X.25 Device Handler.

CIO_QUERY (Query Device) x25sioctl X.25 Device Handler Operation

Purpose

Returns device statistics and device-dependent information.

Description

Note: The counters and profile information can only be cleared by a system user.

The **CIO_QUERY** x25sioctl operation returns the `ras_log` field of the define device structure (DDS).

query_params Parameter Block

For the **CIO_QUERY** operation, the *arg* parameter returns a pointer to the **query_params** structure. This structure contains the following fields:

`status`	Contains the returned value if the return code is **EIO**.
`bufptr`	Points to an **x25_query_data** structure. This structure contains the following fields:
	cc Contains a **cio_stats** structure as defined in the **/usr/include/sys/comio.h** file.
	ds Contains an **x25_stats** structure identifying X.25-specific statistics. This structure is found in the **/usr/include/sys/x25user.h** file.
`buflen`	Specifies the length of the buffer.
`clearall`	Clears the statistics when set to **CIO_QUERY_CLEAR**. Any other setting leaves the statistics unchanged.

x25_stats Structure

The **x25_stats** structure identifies X.25-specific statistics. Information in this structure includes the `transmit_profile` field. This field provides a profile of the transmission packet sizes in use on a port and can be used for configuration of adapter buffers. The `transmit_profile` field contains a count of the number of packets sent since the device was last configured. The size of these packets must be in the range specified.

Index	Size
0	Packet size not known
1	Reserved
2	Reserved
3	Reserved
4	0 - 15
5	16 - 31
6	32 - 63
7	64 - 127
8	128 - 255
9	256 - 511
10	512 - 1023
11	1024 - 2047
12	2048 - 4095
>12	Reserved

Execution Environment

The **CIO_QUERY** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation and the kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates an invalid address was specified.

EIO Indicates an error has occurred. The `arg->status` field contains one of the following exception codes:

- **CIO_BAD_MICROCODE**
- **CIO_HARD_FAIL**
- **CIO_NOT_STARTED**
- **CIO_TIMEOUT**
- **CIO_LOST_DATA**

EMSGSIZE Indicates the statistical data was greater than the length of the buffer specified by the `buflen` field. The data in the buffer is truncated.

ENOBUFS Indicates no buffers are available.

ENXIO Indicates the device has not been completely configured.

Implementation Specifics

The **CIO_QUERY** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** device handler entry point.

CIO_START (Start Session) x25sioctl X.25 Device Handler Operation

Purpose

Starts an X.25 device handler session.

Description

The **CIO_START** x25sioctl operation starts an X.25 session. Only one X.25 session is associated with each **CIO_START** operation. An X.25 session can be initiated by any of the following:

- A call request on a switched virtual circuit (SVC).
- A call accepted on an SVC in response to an incoming call received on some other (listening) session.
- A request for allocation of a permanent virtual circuit (PVC).
- A request to listen for incoming calls satisfying a named specification in the routing table.

If the **CIO_START** operation is the first one issued for a port, the operation also does the appropriate initialization (for example, downloading the microcode).

The **CIO_START** operation returns immediately to the caller, before the command completes. If the operation completes successfully, the X.25 device handler builds a **CIO_START_DONE** status block on completion. For kernel-mode processes, the status block is passed to the associated status function specified at **x25sopen** time. For user-mode processes, the block is placed in the associated status exception queue indicated by a / (slash).

If the immediate return indicates an error, there is no need to halt the operation. However, if the status block indicates an error, the calling process must issue a halt. An X.25 **CIO_HALT** operation can be called before a **CIO_START_DONE** status block is received. In this case, it is undefined whether or not the session generates a **CIO_START_DONE** status block.

> **Note:** Read or write operations should not be performed until the **CIO_START_DONE** status block is received.

Parameter Block

For the **CIO_START** operation, the *arg* parameter points to an **x25_start_data** structure as defined in the **/usr/include/sys/comio.h** file. This structure contains the following fields:

`sb` — Defines a **session_blk** structure as described in the **/usr/include/sys/comio.h** file. This structure contains the following fields:

- **`netid`** — Identifies the network ID. This field can be set by the caller to a correlator returned with any data received on this session.
- **`status`** — Identifies return values. This field is meaningful only if the return code is **EIO**.

`session_name` — Specifies an ASCII name supplied by the caller for RAS purposes. This field is null-terminated if less than 16 characters.

`session_id` — Specifies a unique ID for this session returned by the X.25 device handler. The caller must use this ID to identify the session on all subsequent calls.

`session_type` Specifies the type of session required.

The X.25 device handler permits a process to start a session of type **SESSION_SVC_IN** only if its UID is the same as that of the process that owns the session of type **SESSION_SVC_LISTEN** that received the incoming call.

If the session type is **SESSION_SVC_OUT** or **SESSION_SVC_IN**, then the **CIO_START** operation *ext* parameter is used. The *ext* parameter points to the data required for the Call Request and Call Accepted packets issued by an out or in session. This data is in the form described in the **mbuf** structure (found in the **/usr/include/sys/x25user.h** file). For a kernel-mode process, the data is an mbuf pointer. For user-mode, the data is a pointer to a buffer in user space of the same format.

For a **SESSION_SVC_OUT** session, the `option[2]` field of the status block points to the packet that completed the **CIO_START** operation. This is either a **PKT_CALL_CONNECTED** or **PKT_CLEAR_INDICATION** packet.

`session_protocol` Specifies the protocol for this session. This field is set by the caller and is valid only for a **SESSION_SVC_OUT** or **SESSION_SVC_IN** session. The protocol for **SESSION_PVC** is set in the configuration.

The `session_protocol` field accepts the following values:

PROTOCOL_ELLC
Reserved.

PROTOCOL_ISO8208
No specific action. This value is used whenever no other specific protocol is wanted.

PROTOCOL_QLLC_1980
Selects SNA 1980 cause-and-diagnostic codes instead of CCITT.

PROTOCOL_QLLC_1984
Selects SNA 1984 cause-and-diagnostic codes instead of CCITT.

PROTOCOL_TCPIP
No specific action.

PROTOCOL_YBTS
Yellow Book Transport Service.

For this protocol, the X.25 device handler does not handle X.25 packet sequences on behalf of the user. Instead, incoming packets with the M bit set are passed to the user without waiting for the sequence to complete.

`counter_id` Specifies the counter to increment for any incoming data on this session. This field is set by the calling process. This field set to -1 indicates that counters are not used on this session.

Note: Counter functions are available only to user-mode processes.

`session_type_data` Contains additional data set by the caller. The data returned in this field depends on the value of the `session_type` field. Following are the possible data types:

listen_name
Identifies the nickname of an entry (or collection of entries) in the router table. This must be set by the caller with the **CIO_START** operation when the `session_type` field is set to **SESSION_SVC_LISTEN**.

call_id Contains the incoming call ID supplied to a listening session by the device handler with an incoming call from remote data terminal equipment (DTE). This value must be set by the caller with the **CIO_START** operation when the `session_type` field is **SESSION_SVC_IN**.

logical_channel
Specifies the logical channel number of the PVC to be acquired. This field must be set by the caller with the **CIO_START** operation when the `session_type` field is set to **SESSION_PVC**.

Note: When the session type is **SESSION_SVC_OUT**, no additional data is required.

Execution Environment

The **CIO_START** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates that an invalid address was specified.

EIO Indicates that an error has occurred. The error is returned in the `sb.status` field of the **CIO_START** parameter block and takes any one of the following four exception codes:

- **CIO_BAD_MICROCODE**
- **CIO_HARD_FAIL**
- **CIO_NOMBUF**
- **CIO_TIMEOUT**

In addition, the `sb.status` field may take any of the following X.25-specific codes:

X25_AUTH_LISTEN
Indicates the UID in the router table entry that corresponds to the `listen_name` field does not match the calling UID.

X25_BAD_CALL_ID
Indicates the `call_id` field specified on a **SESSION_SVC_IN** session is not valid.

X25_BAD_PKT_TYPE
Indicates the packet type passed by the *ext* parameter is not valid.

X25_CLEAR
The session has been cleared.

X25_INV_CTR
The counter specified in the `x25_start_data` field does not exist.

X25_NAME_USED
Indicates the `listen_name` field specified on a **SESSION_SVC_LISTEN** session is in use by another application.

X25_NO_LINK
Indicates could not connect to the link.

X25_NO_NAME
Indicates the `listen_name` field specified on an **SESSION_SVC_LISTEN** session is not in the router table.

X25_NOT_PVC
Indicates the channel is not defined as a PVC.

X25_PROTOCOL
Indicates a protocol error occurred. For example, a **SESSION_SVC_IN** session was cleared by the remote DTE before it could be accepted. The clear packet can be read using the **x25sread** operation before issuing the halt.

X25_PVC_USED
Indicates the PVC is in use by another application.

X25_TOO_MANY_VCS
Indicates too many virtual circuits have been opened.

EINVAL Indicates that any of the following errors may have occurred:

- The `session_type` field is not valid. This field must be set to **PKT_CALL_REQ** for a **SESSION_SVC_OUT** session or to **PKT_CALL_ACCEPT** for a **SESSION_SVC_IN** session.
- The `session_protocol` field is not valid.
- The *chan* parameter was not opened in the correct mode. For a **SESSION_MONITOR** session, the channel must be opened in M mode. For sessions of type **SESSION_SVC_IN**, **SESSION_SVC_OUT**, and **SESSION_SVC_LISTEN**, the channel must be opened without a mode.

EINTR Indicates that a signal was received during the call.

ENOBUFS Indicates that there are no spare buffers in the pool.

EBUSY Indicates that the number of starts for this device was exceeded. This occurs with a monitor device that can only support one start.

ENXIO Indicates that the device was not completely configured. Initial configuration must be completed before any starts can be issued.

Implementation Specifics

The **CIO_START** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** entry point, **x25sopen** entry point, **x25sread** entry point.

The **CIO_HALT** x25sioctl X.25 Device Handler Operation.

Status Blocks for the X.25 Device Handler.

IOCINFO (Identify Device) x25sioctl X.25 Device Handler Operation

Purpose

Returns I/O character information for an X.25 device.

Description

The **IOCINFO** x25sioctl operation returns I/O character information for an X.25 device. The parameter block for this operation is defined in the **/usr/include/sys/definfo.h** file by the **devinfo** structure. This structure contains the following fields:

`devtype`	Identifies the type of device. This is set to the **DD_X25** value (defined as the ASCII character x).
`flags`	Undefined for X.25 devices.
`devsubtype`	Undefined for X.25 devices.

In addition to the above members, the **/usr/include/sys/devinfo.h** file also contains an **x25** structure (found in the **/usr/include/sys/x25user.h** file). This structure defines the X.25 device and contains the following members:

`support_level`	Identifies a support level of 1980 or 1984.
`nua`	Contains a null-terminated ASCII string that represents the network-user address.
`subscription_facilities_supported`	Contains device-dependent information.
`network_id`	Specifies the identification code for the network. The range and default value for this code is defined by the device configuration.
`max_tx_packet_size`	Specifies the maximum size of a transmitted data packet. This packet is encoded in the manner of the ISO 8208 definition.
`max_rx_packet_size`	Specifies the maximum size of a received data packet. This packet is encoded in the manner of the ISO 8208 definition.
`default_svc_tx_packet_size`	Specifies the default transmit packet size for a switched virtual circuit (SVC). This packet is encoded in the manner of the ISO 8208 definition.
`default_svc_rx_packet_size`	Specifies the default received packet size for an SVC. This packet is encoded in the manner of the ISO 8208 definition.

Permanent Virtual Circuit (PVC) Packets

PVC packet sizes are configured on a per-PVC basis. To determine the packet size on a PVC you can use either of the following operations:

- **CIO_QUERY** operation
- **CIO_START** operation followed by a **X25_QUERY_SESSION** operation

Execution Environment

The **IOCINFO** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to the following value:

EFAULT Indicates an invalid address.

Implementation Specifics

The **IOCINFO** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **CIO_QUERY** x25sioctl X.25 Device Handler Operation, **CIO_START** x25sioctl X.25 Device Handler Operation, **X25_QUERY_SESSION** x25sioctl X.25 Device Handler Operation.

X25_ADD_ROUTER_ID (Add Router ID) x25sioctl X.25 Device Handler Operation

Purpose

Registers a new routing name and routing specification.

Description

> **Note:** Only a process that has opened the router special file can call the **X25_ADD_ROUTER_ID** operation.

The **X25_ADD_ROUTER_ID** operation registers a new route name and routing specification in the router table. For this operation, the *arg* parameter points to an **x25_router_add** structure. This structure contains the following fields:

`router_id` Specifies the unique identifier for the entry.

`listen_name` Specifies the nickname identifier for the entry. The nickname need not be unique.

`priority` Identifies the integer priority to attach to the routing request. A priority of 1 is high; 3 indicates a low priority.

`action` Specifies the action to take if the name is not being listened to. This field takes the following values:

0 Forwards the incoming call so that it can match other listening specifications.

1 Rejects the incoming call with cause 0, diagnostic 0.

`uid` Identifies the user ID allowed to receive these incoming calls. This field can be the user ID number. A value of -1 indicates that any user ID can receive the calls. Any attempt by a user with insufficient authority to listen on a name is rejected with the **EACCES** return value.

`call_user_data` Contains the call user data to match with an incoming call. The last character can be an * (asterisk). The format of this data is a string of hexadecimal characters and an optional * (asterisk); for example, `C3*`. The call user data is null-terminated if it is less than the maximum length.

Additionally, the **x25_router_add** structure contains the following address fields which are defined in the glossary:

- called_subaddress[20]
- calling_address[20]
- extended_calling_address[41]
- extended_called_address[41]

These addresses are set to match with an incoming call. The last character of an address can be an * (asterisk). The addresses are null-terminated if less than the maximum length.

Execution Environment

The **X25_ADD_ROUTER_ID** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the ioctl operation was issued on an channel that was not opened in Router mode.

EFAULT Indicates a specified address was not valid.

EINVAL Indicates one of the following occurred:

- The specified router ID already exists. (Router IDs must be unique.)
- The `action` field passed was neither 0 nor 1.

ENOMEM Indicates the operation ran out of memory.

Implementation Specifics

The **X25_ADD_ROUTER_ID** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** handler entry point.

X25_COUNTER_GET (Get Counter) x25sioctl X.25 Device Handler Operation

Purpose

Gets a counter for asynchronous notification.

Description

Note: Only user-mode processes can use counter operations.

The **X25_COUNTER_GET** x25sioctl operation uses the *arg* parameter to return a counter ID. The ID can be used to wait and test for incoming X.25 data.

Execution Environment

The **X25_COUNTER_GET** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates a specified address is not valid.
ENOSPC Indicates there are no counters available to allocate.

Implementation Specifics

The **X25_COUNTER_GET** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** entry point.

The **X25_COUNTER_READ** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_REMOVE** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_WAIT** x25sioctl X.25 Device Handler Operation.

Using Counters to Correlate Messages in *AIX 5L Version 5.2 Communications Programming Concepts*.

X25_COUNTER_READ (Read Counter) x25sioctl X.25 Device Handler Operation

Purpose

Reads the value of a counter.

Description

Note: Only user-mode processes can use counter operations.

The **X25_COUNTER_READ** x25sioctl operation reads the value of a counter. For this operation, the *arg* parameter points to the **x25_counter_info** structure. This structure contains the following fields:

`counter_id` Identifies a counter to read.
`counter_value` Holds the current value of the counter on return of the **X25_COUNTER_READ** operation.

Execution Environment

The **X25_COUNTER_READ** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates a specified address is not valid.
EINVAL Indicates the counter ID does not exist.

Implementation Specifics

The **X25_COUNTER_READ** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_COUNTER_GET** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_REMOVE** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_WAIT** x25sioctl X.25 Device Handler Operation.

The **x25sioctl** device handler entry point.

X25_COUNTER_REMOVE (Remove Counter) x25sioctl X.25 Device Handler Operation

Purpose

Removes a counter from the system.

Description

Note: Only user-mode processes can use counter operations.

The **X25_COUNTER_REMOVE** x25sioctl operation removes the specified counter from the system. For this operation, the *arg* parameter indicates what ID is to be removed. An error code is returned if there is outstanding data to be read associated with this counter.

Execution Environment

The **X25_COUNTER_REMOVE** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the application did not get the counter. The counter is not deleted.

EBUSY Indicates one of the following errors occurred:

- Some packets are still waiting to be read.
- The counter is being waited on by another process.

EINVAL Indicates the counter ID specified does not exist.

Implementation Specifics

The **X25_COUNTER_REMOVE** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_COUNTER_GET** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_READ** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_WAIT** x25sioctl X.25 Device Handler Operation.

The **x25sioctl** device handler entry point.

X25_COUNTER_WAIT (Wait Counter) x25sioctl X.25 Device Handler Operation

Purpose

Waits for the contents of counters to change.

Description

Note: Only user-mode processes can use counter operations.

The **X25_COUNTER_WAIT** x25sioctl operation waits for the contents of a counter to change. The process that called this operation is suspended until the value of one of its counters exceeds the value specified by the `counter_value` field.

For the **X25_COUNTER_WAIT** operation, the *arg* parameter points to the **x25_counter_list** structure. This structure contains the following fields:

`counter_num` Identifies the number of elements in the counter array.

`counter_array` Specifies an array of the following:

`flags` Indicates if the counter information was successfully matched. If successful, the top bit of the `flags` field is set on the return of the Counter Wait operation.

`counter_id`
Identifies the counter to wait on.

`counter_value`
Specifies the value the counter must exceed in order for the counters to match successfully.

Execution Environment

The **X25_COUNTER_WAIT** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates a specified address is not valid.
EIDRM Indicates the counter has been removed.
EINVAL Indicates one or more of the counters in the list does not exist.
ENOMEM Indicates the operation ran out of memory.

Implementation Specifics

The **X25_COUNTER_WAIT** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_COUNTER_GET** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_READ** x25sioctl X.25 Device Handler Operation, **X25_COUNTER_REMOVE** x25sioctl X.25 Device Handler Operation.

The **x25sioctl** entry point.

Using Counters to Correlate Messages in *AIX 5L Version 5.2 Communications Programming Concepts*.

X25_DELETE_ROUTER_ID (Delete Router ID) x25sioctl X.25 Device Handler Operation

Purpose

Removes a routing name.

Description

Note: Only a process that has opened the router special file can call the X.25 **X25_DELETE_ROUTER_ID** operation.

The **X25_DELETE_ROUTER_ID** x25sioctl operation removes a routing name from the router table. For this operation, the *arg* parameter points to the **x25_router_del** structure. This structure contains the following fields:

`router_id` Specifies the unique ID for the entry.

`override` Indicates how listening is handled. If set to 0, the routing entry is not deleted if any process is listening for it. If set to a non-zero value, outstanding listens are canceled. No notification is given to the listening applications if the outstanding listens are canceled.

Execution Environment

The **X25_DELETE_ROUTER_ID** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the ioctl was issued on a channel that was not opened in Router mode.

EBUSY Indicates the router ID was being listened to and the override option was not set.

EFAULT Indicates a specified address was not valid.

EINVAL Indicates the router ID cannot be deleted because it does not exist.

Implementation Specifics

The **X25_DELETE_ROUTER_ID** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** entry point.

X25_DIAG_IO_READ (Read Register) x25sioctl X.25 Device Handler Operation

Purpose

Reads from an I/O register on the X.25 Interface Co-Processor/2.

Description

Note: Only a process that has opened the device for diagnostics can issue this call.

The **X25_DIAG_IO_READ** x25sioctl operation reads from an I/O register on the X.25 Interface Co-Processor/2. Both direct and indirect registers can be read since the card pointer register is adjusted by this operation.

For this operation, the *arg* parameter returns a pointer to an **x25_diag_io** structure. The value this operation reads is placed in the `value` field of the **x25_diag_io** structure.

Execution Environment

The **X25_DIAG_IO_READ** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES The device was not opened in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_IO_READ** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_WRITE** x25sioctl X.25 Device Handler Operation, **X25_DIAG_MEM_READ** x25sioctl X.25 Device Handler Operation, **X25_DIAG_MEM_WRITE** x25sioctl X.25 Device Handler Operation, **X25_DIAG_TASK** x25sioctl X.25 Device Handler Operation.

The **x25sioctl** entry point.

X25_DIAG_IO_WRITE (Write to Register) x25sioctl Operation

Purpose

Writes to an I/O register on the X.25 Interface Co-Processor/2.

Description

Note: Only a process that has opened the device for diagnostics can call this process.

The **X25_DIAG_IO_WRITE** x25sioctl operation writes to an I/O register on the X.25 Interface Co-Processor/2. Both direct and indirect registers can be written to as the card pointer register is adjusted by this operation.

Execution Environment

The **X25_DIAG_IO_WRITE** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES The channel was not opened in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_IO_WRITE** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_READ** x25sioctl operation, **X25_DIAG_MEM_READ** x25sioctl operation, **X25_DIAG_MEM_WRITE** x25sioctl operation, **X25_DIAG_TASK** x25sioctl operation.

The **x25sioctl** entry point.

X25_DIAG_MEM_READ (Read Memory) x25sioctl Operation

Purpose

Reads memory from the X.25 Interface Co-Processor/2 into a user buffer.

Description

Note: Only a process that has opened the device for diagnostics can call this process.

The **X25_DIAG_MEM_READ** x25sioctl operation reads memory from the X.25 Interface Co-Processor/2 into a user's buffer. For this operation, the *arg* parameter points to a **x25_diag_mem** structure. This structure provides the following:

- Page and offset of card memory to start from
- Number of bytes to read
- Pointer to a buffer into which the data is read

The read operation can cover more than one page of card memory.

Execution Environment

The **X25_DIAG_MEM_READ** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES The channel was not opened in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_MEM_READ** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_READ** x25sioctl operation, **X25_DIAG_IO_WRITE** x25sioctl operation, **X25_DIAG_MEM_WRITE** x25sioctl operation, **X25_DIAG_TASK** x25sioctl operation.

The **x25sioctl** entry point.

X25_DIAG_MEM_WRITE (Write Memory) x25sioctl Operation

Purpose

Writes memory to the X.25 Interface Co-Processor/2 from a user buffer.

Description

Note: Only a process that has opened the device for diagnostics can issue this call.

The **X25_DIAG_MEM_WRITE** x25sioctl operation writes memory to the X.25 Interface Co-Processor/2 from a user buffer. For this operation, the *arg* parameter points to a **x25_diag_mem** structure. This parameter provides the following:

- Page and offset of card memory to start from
- Number of bytes to write
- Pointer to the user buffer containing the data to write

The write can cover more than one page of card memory.

Execution Environment

The **X25_DIAG_MEM_WRITE** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES The channel was not opened in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_MEM_WRITE** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_READ** x25sioctl operation, **X25_DIAG_IO_WRITE** x25sioctl operation, **X25_DIAG_MEM_READ** x25sioctl operation, **X25_DIAG_TASK** x25sioctl operation.

The **x25sioctl** entry point.

X25_DIAG_RESET (Reset Adapter) x25sioctl Operation

Purpose

Resets the X.25 Interface Co-Processor/2.

Description

Note: Only a process that has opened the device for diagnostics can call this process.

The **X25_DIAG_RESET** x25sioctl operation completely resets the X.25 Interface Co-Processor/2.

Execution Environment

The **X25_DIAG_RESET** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EINVAL The channel was not opened in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_RESET** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_READ** x25sioctl operation, **X25_DIAG_IO_WRITE** x25sioctl operation, **X25_DIAG_MEM_READ** x25sioctl operation, **X25_DIAG_MEM_WRITE** x25sioctl operation, **X25_DIAG_TASK** x25sioctl operation.

The **x25sioctl** entry point.

X25_DIAG_TASK (Download Diagnostics) x25sioctl Operation

Purpose

Provides the means to download the diagnostics task on to the card.

Description

The **X25_DIAG_TASK** x25sioctl operation provides the means to download the diagnostics task on to the X.25 Interface Co-Processor/2. The task microcode must have been previously downloaded to the device handler using the **CIO_DNLD** operation.

For the **X25_DIAG_TASK** operation, the *arg* parameter points to a **x25_diag_addr** structure that is used to return the load page and offset.

Execution Environment

The **X25_DIAG_TASK** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EINVAL The microcode was not available to download.
ENACCES The channel was not opened in diagnostic mode. You must have appropriate authority to open a channel in diagnostic mode.
ENXIO The operation attempted to read a card that was not configured.

Implementation Specifics

The **X25_DIAG_TASK** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_DIAG_IO_READ** x25sioctl operation, **X25_DIAG_IO_WRITE** x25sioctl operation, **X25_DIAG_MEM_READ** x25sioctl operation, **X25_DIAG_MEM_WRITE** x25sioctl operation.

The **x25sioctl** entry point.

X25_LINK_CONNECT (Connect Link) x25sioctl Operation

Purpose

Connects the link to the data circuit-terminating equipment (DCE).

Description

> **Note:** Only a process that has opened the router special file can issue the **X25_LINK_CONNECT** operation.

The **X25_LINK_CONNECT** x25sioctl operation connects the X.25 link to the network. If required, the connection is made using the automatic calling unit (ACU). If the link is already connected, no action is taken.

For the **X25_LINK_CONNECT** operation, the *arg* parameter points to the **x25_connect_data** structure. This structure contains only a `status` field. This field has meaning only when the return code is **EIO**.

Execution Environment

The **X25_LINK_CONNECT** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the calling application does not have root authority.

EIO Indicates an I/O error occurred. The `status` field in the **x25_connect_data** structure contains one of the following values:

CIO_BAD_MICROCODE
Indicates that the microcode download failed.

CIO_HARD_FAIL
Indicates that a hardware failure was detected.

CIO_TIMEOUT
Indicates that a time out occurred.

ENOBUFS Indicates no buffers are available.

Implementation Specifics

The **X25_LINK_CONNECT** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_LINK_DISCONNECT** x25sioctl operation, **X25_LINK_STATUS** x25sioctl operation.

The **x25sioctl** entry point.

X25_LINK_DISCONNECT (Disconnect Link) x25sioctl Operation

Purpose

Disconnects the link to the data circuit-terminating equipment (DCE).

Description

Note: This command is restricted to user programs that have root authority.

The **X25_LINK_DISCONNECT** x25sioctl operation disconnects the X.25 link from the network. If the link is already disconnected, no action is taken. If there are virtual calls in progress on the link, disconnection takes place only if the *override* parameter is nonzero.

The **X25_LINK_DISCONNECT** operation returns synchronously. The **X25_LINK_STATUS** operation is used to determine if the disconnect operation is complete.

For the **X25_LINK_DISCONNECT** operation, the *arg* parameter points to the **x25_disconnect_data** structure. This structure contains the following fields.

`status` Holds values supplied by the ioctl operation if there is an **EIO** error**.**

`override` Specifies how disconnection occurs. If this parameter is 0, the disconnection takes place only if there are no virtual calls in progress. Otherwise, the disconnection is forced. This disconnects the link layer only, not the physical layer.

Execution Environment

The **X25_LINK_DISCONNECT** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the calling application does not have root authority.

EBUSY Indicates there are active circuits on the link.

EIO Unable to disconnect due to an I/O error. The `status` field in the **x25_disconnect_data** structure contains one of the following common exception codes:

CIO_HARD_FAIL
Indicates that a hardware failure was detected.

CIO_TIMEOUT
Indicates that a time out occurred.

ENOBUFS Indicates no available buffers.

Implementation Specifics

The **X25_LINK_DISCONNECT** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_LINK_CONNECT** x25sioctl operation, **X25_LINK_STATUS** x25sioctl operation.

The **x25sioctl** entry point.

X25_LINK_STATUS (Link Status) x25sioctl Operation

Purpose

Returns the status of the link.

Description

The **X25_LINK_STATUS** x25sioctl operation returns the status of a link. This operation returns the last known status of the link to the calling program.

For the **X25_LINK_STATUS** operation, the *arg* parameter points to a buffer. On return of this operation, the buffer is filled with a **x25_link_status** structure. This structure contains the following five fields:

`status` Supplied by an X.25 device handler with a return value when the return code is **EIO**.

`packet` Identifies the status of the packet layer. This field has the following possible values:

- **0** Link disconnected
- **1** Connecting link
- **2** Link connected

`frame` Specifies the status of the frame layer. This field has the same values as the `packet` field.

`physical` Specifies the status of the physical layer. This field has the same values as the `packet` field.

`no_of_vcs_in_use` Identifies the number of virtual circuits currently in use on the link.

Execution Environment

The **X25_LINK_STATUS** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT Indicates a specified address is not valid.

EIO Indicates an error occurred. The `status` field of the **x25_link_status** structure contains one of the following exception codes:

CIO_BAD_MICROCODE
Indicates that the microcode download failed.

CIO_HARD_FAIL
Indicates that a hardware failure was detected.

ENOBUFS Indicates no available buffers.

Implementation Specifics

The **X25_LINK_STATUS** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **X25_LINK_CONNECT** x25sioctl operation, **X25_LINK_DISCONNECT** x25sioctl operation.

The **x25sioctl** entry point.

X25_LOCAL_BUSY (Local Busy) x25sioctl Operation

Purpose

Enables or disables receiving data packets on a port.

Description

> **Note:** Only the user who called the **CIO_START** operation can call the **X25_LOCAL_BUSY** operation.

The **X25_LOCAL_BUSY** x25sioctl operation enables or disables receiving data and interrupt packets on a given session. This operation can be used to slow down large blocks of received data or reduce the number of buffers required. However, clear-and-reset packets are still passed on.

The effects of disabling received packets do not take place immediately after calling the **X25_LOCAL_BUSY** operation. Data packets that arrived before the call or packets currently being read off the card are passed on.

The **X25_LOCAL_BUSY** operation does not affect the outcome of the **x25read** or **x25select** entry points. These operations continue to wait for received packets. To query the status of a local busy on a session, use the **X25_QUERY_SESSION** operation.

Parameter Block

For the **X25_LOCAL_BUSY** operation, the *arg* parameter points to a buffer that contains the **x25_local_busy** structure. This structure contains the following fields:

`busy_mode` Specifies the handling of data packets in the session. This field accepts one of the following values:

- **0** Enables the receiving of data on this session.
- **1** Disables the receiving of data on this session.

`session_id` Identifies the session to which this operation applies.

Execution Environment

The **X25_LOCAL_BUSY** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the call must be made by the user who issued the **CIO_START** operation.
EFAULT Indicates a specified address was not valid.
EINVAL Indicates the specified session ID was not valid, or the `busy_mode` field was illegal.

Implementation Specifics

The **X25_LOCAL_BUSY** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Data Transmission and Reception for the X.25 Device Handler in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

The **CIO_START** x25sioctl operation, **X25_QUERY_SESSION** x25sioctl operation.

Related Information

The **x25sioctl** entry point.

X25_QUERY_ROUTER_ID (Query Router) ID x25sioctl Operation

Purpose

Queries an entry in the routing table.

Description

Note: This operation is restricted to user programs that have root authority.

The **X25_QUERY_ROUTER_ID** x25sioctl operation queries an entry in the routing table. For this operation, the *arg* parameter points to the **x25_router_query** structure. This structure contains the following fields:

`router_id`	Specifies what entry to query.
`pid`	Is set on return of the query to the process ID of the listening process. A value of 0 indicates that no process is listening.

Execution Environment

The **X25_QUER_ROUTER_ID** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT	Indicates the specified address was not valid.
EINVAL	Indicates the specified router ID was not in the router table.

Implementation Specifics

The **X25_QUER_ROUTER_ID** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sioctl** entry point.

X25_QUERY_SESSION (Query Session) x25sioctl Operation

Purpose

Queries the status of an open X.25 session.

Description

Note: This call only succeeds on SVCs and PVCs.

The **X25_QUERY_SESSION** x25sioctl operation supplies the information for a session in the user data area. The packet size, window size, and throughput class values are not available until the session is completely established. To query the static configuration, use the **CIO_QUERY** operation.

x25_query_session_data Parameter Block

For the **X25_QUERY_SESSION** operation the *arg* parameter points to the **x25_query_session_data** structure. Within this structure, the session to be queried is identified either by a nonzero session ID or a nonzero logical channel number. If both the `session_id` and `logical_channel` fields are nonzero, the `session_id` field is used.

The fields in the **x25_query_session_data** structure are set on return. All the X.25 facilities specified by the structure's field are encoded as in the ISO 8208 definition. The **x25_query_session_data** structure contains the following fields:

`netid`	Identifies the user-defined correlator set by the **CIO_START** operation.
`session_name`	Identifies the user-defined name set by the **CIO_START** operation.
`session_id`	Identifies the device handler correlator returned from the **CIO_START** operation.
`local_busy`	Contains a value of 1 if the session is in Local Busy mode, or a 0, if not.
`session_protocol`	Specifies the higher-level protocol specified by the user for this session in the **CIO_START** operation.
`logical_channel`	Identifies the X.25 logical channel number used by the session.
`tx_tclass`	Specifies the transmit throughput class facility in use on the session. If the call has not been established, this is 0.
`rx_tclass`	Specifies the receive throughput class facility in use on the session. If the call has not been established, this is 0.
`tx_packet_size`	Identifies the outbound packet size in use on the session. If the call has not been established, this field is set to 0.
`rx_packet_size`	Identifies the inbound packet size in use on the session. If the call has not been established, this field is set to 0.
`tx_window_size`	Identifies the outbound window size in use on the session. If the call has not been established, this field is set to 0.
`rx_window_size`	Identifies the inbound window size in use on the session. If the call has not been established, this field is set to 0.

Execution Environment

The **X25_QUERY_SESSION** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EFAULT	Indicates a specified address was not valid.
EINVAL	Indicates the session ID was not valid, or the `logical_channel` field was not valid.

Implementation Specifics

The **X25_QUERY_SESSION** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **CIO_START** x25sioctl operation.

The **x25sioctl** entry point.

X25_REJECT_CALL (Reject Call) x25sioctl Operation

Purpose

Provides the means to reject an incoming X.25 call.

Description

Note: A call can be rejected only by the process that called the **CIO_START** operation.

The **X25_REJECT_CALL** x25sioctl operation is used to reject an X.25 incoming call that was forwarded to a session of type **SESSION_SVC_LISTEN**. This operation causes a clear request to be issued in response to the incoming call.

The **X25_REJECT_CALL** operation returns immediately to the caller, before the command completes. If the immediate return indicates no error, the X.25 device handler builds a status block of type **X25_REJECT_DONE** on receipt of a clear confirm or clear indication. For kernel mode processes, the status block is passed to the associated status function. The status function is specified when the X.25 channel is opened. For user-mode processes, the block is placed in the associated status and exception queue.

The x25_reject_data Parameter Block

For the **X25_REJECT_CALL** call operation, the *arg* parameter points to a **x25_reject_data** structure. The `sb.status` field of this structure is meaningful on return only if the return code is **EIO**.

For the **X25_REJECT_CALL** operation, the *ext* parameter optionally points to a buffer containing the data required for a clear request packet. This data is in the form described in the **mbuf** structure. For a kernel-mode process, this parameter points to the **mbuf** structure. For a user-mode process, it points to a buffer of the same format in user space. If the pointer is a null character, the clear request is sent with default cause-and-diagnostic codes and no facilities or user data.

Execution Environment

The **X25_REJECT_CALL** operation can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. If -1 is returned, the kernel sets the **errno** global variable to one of the following values:

EACCES Indicates the reject must be performed by the same process that called the X.25 **CIO_START** operation.
EFAULT Indicates an invalid address was specified.

EINVAL Indicates one of the following occurred:

- A reject was issued on a session that was not started in **SESSION_SVC_LISTEN** mode.
- The *ext* parameter points to a buffer that does not have a packet type of **PKT_CLEAR_REQ**.

In addition, the `arg->status` field may return one of three X.25-specific codes:

X25_BAD_CALL_ID
The `call_id` field specified is invalid.

X25_CLEAR
Indicates the session has been cleared.

X25_PROTOCOL
Indicates a protocol error occurred.

EIO Indicates an error has occurred. The `arg->status` field in the **x25_reject_data** structure contains one of four exception codes:

CIO_HARD_FAIL
Indicates that a hardware failure was detected.

CIO_NOMBUF
Indicates that the operation was unable to allocate **mbuf** structures.

CIO_NOT_STARTED
Indicates that the command could not be accepted because the device has not yet been started by the first call to **CIO_START** operation.

CIO_TIMEOUT
Indicates that a time out occurred.

Implementation Specifics

The **X25_REJECT_CALL** operation functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **CIO_START** x25sioctl operation.

The **x25sioctl** entry point.

The X.25 **mbuf** structure in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

---

x25smpx X.25 Device Handler Entry Point

Purpose

Provides the means to allocate and deallocate a channel into X.25 device handler.

Syntax

```
int x25xmpx (devno, chan, channame)
 dev_t devno;
 int *chan;
 char *channame;
```

Parameters

devno Specifies the major and minor device numbers.

chan Specifies the channel ID. If the *channame* parameter is a null character, the *chan* parameter identifies the channel to be deallocated. Otherwise, the **x25smpx** entry point returns the ID of the allocated channel to the *chan* parameter.

channame Points to the remaining path name describing the channel to allocate. The *channame* parameter accepts the following values:

null Deallocates the channel.

Pointer to a null string
Allows a normal open sequence of the X.25 device on the channel ID generated by the **x25smpx** entry point.

Pointer to a "D"
Allows the X.25 device to be opened in diagnostic mode on the channel ID generated by the **x25smpx** entry point.

Pointer to an "M"
Allows the X.25 device to be opened in monitor mode on the channel ID generated by the **x25smpx** entry point.

Pointer to an "R"
Allows the X.25 device to be opened in router mode on the channel ID generated by the **x25smpx** entry point.

Description

Note: This entry point is called by the kernel. It cannot be called directly by a user- or kernel-mode process.

The **x25smpx** entry point provides the means for allocating and deallocating a channel into the X.25 device handler. This entry point is called by the kernel in response to an **open** subroutine (before calling the **x25sopen** entry point) or in response to a **close** subroutine (after calling the **x25sclose** entry point).

Execution Environment

An **x25smpx** entry point can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to one of the following values:

EINVAL Indicates an invalid parameter was specified.
EPERM Indicates an open in the specified mode is denied.
EBUSY Indicates the device is already open in diagnostic, monitor, or router mode.

Implementation Specifics

The **x25smpx** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sclose** entry point, **x25sopen** entry point.

The **close** subroutine, **open** subroutine.

X.25 Device Handler Modes in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

x25sopen X.25 Device Handler Entry Point

Purpose

Initializes a channel into the X.25 device handler.

Syntax

```
int x25sopen (devno,
devflag, chan, ext)
dev_t devno;
ulong devflag;
int chan;
struct kopen_ext *ext;
```

Parameters

devno Specifies major and minor device numbers.

devflag Indicates how the device was opened and whether the caller is a user- or kernel-mode process. This parameter accepts the following flags:

DKERNEL
Specifies a kernel-mode process called the entry point. This flag is clear if a user-mode process called the entry point.

DREAD
Specifies open for reading. This is the default for the X.25 handler regardless of whether this flag is set.

DWRITE
Specifies open for writing. This is the default for the X.25 handler regardless of whether this flag is set.

DAPPEND
Specifies open for appending. The X.25 handler ignores this flag.

DNDELAY
Specifies that the X.25 device handler performs nonblocking reads and writes. If this flag is not set, the X.25 device handler performs blocking reads and writes.

chan Identifies the channel number assigned by the **x25smpx** routine.

ext Specifies the extended system call. This parameter is required for kernel-mode processes and ignored for user-mode processes.

Description

The **x25sopen** entry point performs data-structure allocation and initialization. Time-consuming tasks, such as port initialization and connection establishment, are deferred until the first **CIO_START** operation is issued. This call is synchronous and does not return until the **x25open** entry point is complete.

> **Note:** If this is the first open request to the X.25 device handler, the interrupt level and interrupt handler entry point are registered.

Parameter Block

For the **x25sopen** entry point, the *ext* parameter can be a pointer to the **kopen_ext** structure defined in the **/usr/include/sys/comio.h** file. This structure contains the following fields:

`status` Identifies the status of the open process. This value is meaningful only if the **EIO** code is returned.

`open_id` Specifies the channel correlator for kernel mode processes. This value is passed to kernel functions to identify which channel an event occurred on.

`rx_fn` Specifies the address of a kernel procedure. This procedure is called by the X.25 device handler whenever received data is to be processed. This kernel procedure must be defined as follows:

```
void rx_fn (open_id,read_ext, mbufptr)
ulong open_id;
struct x25_read_ext *read_ext;
struct mbuf *mbufptr;
```

The parameters in this kernel procedure are defined as follows:

open_id
Specifies the ID of this instance of the **x25sopen** entry point. The device handler sets this parameter to the ID originally passed to the X.25 device handler with the **x25sopen** entry point.

read_ext
Contains the status of the **x25sopen** entry point. Currently, this parameter accepts a value of **CIO_OK** or **CIO_BUF_OVFLW**.

mbufptr Points to received data. This data is in the form described by the **mbuf** structure.

The kernel-mode process making the call to the **x25sopen** entry point is responsible for pinning the **rx_fn** kernel procedure before making the call. When the X.25 device handler calls the kernel procedure, the X.25 device handler pins the **mbuf** structure. It is the responsibility of the **rx_fn** kernel procedure to free the pinned **mbuf** structure.

`tx_fn` Identifies the address of a kernel procedure. The X.25 device handler calls this procedure when both the following conditions are true:

- The most recent **x25swrite** entry point for this channel was unsuccessful with a return code of **EAGAIN**, indicating the write request was not performed.
- The **x25sopen** entry point, or the most recent **x25sioctl** operation for this channel, indicates the nonblocking mode (**DNDELAY**) is set.

The **tx_fn** kernel process should be defined as follows:

```
void tx_fn (open_id)
ulong open_id;
```

The parameter in this kernel process is defined as follows:

open_id
Identifies the ID of the **x25sopen** entry point. The device handler sets this value to the ID passed with the **x25sopen** entry point.

The kernel-mode process making the call to the **x25sopen** entry point is responsible for pinning the **tx_fn** kernel procedure before making the call.

`stat_fn` The address of a kernel procedure to be called by the X.25 device handler whenever a status block becomes available. The kernel procedure should have the following structure:

```
void stat_fn (open_id,
sblk_ptr)
ulong open_id;
struct status_block *sblk_ptr;
```

The kernel procedure parameters have the following values:

open_id
Identifies the ID of the open entry point. The device handler sets this value to the ID passed with the **x25sopen** entry point.

sblk_ptr
Points to a status block.

The kernel-mode process that calls the **x25sopen** entry point is responsible for pinning the **stat_fn** kernel procedure before making the open call.

The **rx_fn**, **tx_fn** and **stat_fn** kernel procedures are all made synchronously at high priority. It is therefore imperative that the called kernel procedure return quickly. Until the return, the kernel procedure cannot call any other device entry point.

Note: Entry points are associated with a channel initialized by the **x25sopen** entry point. Sessions are initialized by the **CIO_START** operation. A single channel can support numerous sessions.

Execution Environment

An **x25sopen** entry point can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to one of the following values:

EINVAL Indicates a kernel user passed an invalid function.

EIO Indicates that an error has occurred. The `sb.status` field contains the **CIO_HARD_FAIL** return value, indicating a hardware failure was detected.

EINTR Indicates that the **open** subroutine was interrupted.

ENODEV Indicates that the device requested does not exist.

EBUSY Indicates that the maximum number of opens was exceeded. This error results from an attempt to open a channel in diagnostic mode while other channels on the minor device number are open. This error can also result from an attempt to open a channel while another channel is already open and running in monitor or router mode.

ENOMEM Indicates that the X.25 device handler was unable to allocate space required for the open.

ENXIO Indicates that one of the following occurred:

- An attempt was made to open the X.25 device handler before it was configured.
- The interrupt could not be registered.

Implementation Specifics

The **x25sopen** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **CIO_START** x25ioctl X.25 Device Handler Operation.

The **x25sioctl** entry point, **x25smpx** entry point, **x25swrite** entry point.

The **open** subroutine.

X.25 Device Handler Modes in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

---

x25sread X.25 Device Handler Entry Point

Purpose

Provides the means to receive data from the X.25 adapter.

Syntax

```
int x25sread (devno, uiop, chan, ext)
dev_t devno;
struct uio *uiop;
int chan;
struct x25_read_ext *ext;
```

Parameters

devno	Specifies major and minor device numbers.
uiop	Points to a **uio** structure.
chan	Identifies the channel number assigned by the **x25smpx** routine.
ext	Points to the **x25_read_ext** structure. This structure is found in the **/usr/include/sys/x25user.h** file and it contains a `call_id` field and a `re.status` field. The `call_id` field is only valid on sessions of type **SESSION_SVC_LISTEN**. The `re.status` field is meaningful only if the return value is **EIO**.

Description

Note: This entry point can only be called by user-mode processes. Data received for a kernel-mode process is passed to the **rx_fn** kernel procedure specified by the **x25sopen** entry point.

The **x25sread** entry point provides the means to receive incoming data on the session specified by `session_id` field. If the `session_id` field is 0 (zero) and the device was opened in normal mode, data for any session started by this channel is returned, and the `session_id` field is filled in accordingly. The X.25 device handler copies the data to the user buffer and decrements the `uiop->resid` field by the number of bytes moved.

X.25 data is made up of an M-bit sequence. This sequence is consolidated before it is made available for read operations. The exception are sessions of type **X25_SESSION_YBTS**. For these sessions, each packet is available as a separate data block.

Notes:

1. The order of incoming data is preserved for each session, but is not guaranteed across sessions.
2. The **x25_packet_data** common data structure describes the buffering of incoming X.25 packet sequences. This structure is found in the **/usr/include/sys/x25user.h** file.

The **x25sread** entry point can be a blocking or nonblocking read. The type of read is determined by flags specified by the **x25sopen** entry point when the channel is opened. If the read is blocking, and no data is available, the **x25sread** entry point blocks until data is received. If the read is nonblocking and no data is available, the entry point returns an error code.

If the current session was initialized for listening, the only data that can be read on the session is an incoming call. The user process should respond by issuing a **X25_REJECT** operation on the current session or by starting a new session with a Start Session **CIO_START** operation to accept the call.

When a **PKT_CLEAR_IND** packet is received, the user must respond with a **CIO_HALT** operation. As a result, no further **x25swrite** entry points are accepted. If the session is a **SESSION_MONITOR** type, then the data buffer contains monitor control sequences.

Parameter Block

For the **x25sread** entry point, the *arg* parameter returns a pointer to the **uio** structure. This structure specifies the location and length of the caller's data area to transfer information. The **uio** structure is defined in the **/usr/include/sys/uio.h** file.

The data is in the form described in the **mbuf** structure. The value for the `packet_type` field for **SESSION_SVC_LISTEN** sessions is **PKT_INCOMING_CALL**. For other sessions, the possible packet types are the following:

- **PKT_DATA**
- **PKT_INT**
- **PKT_INT_CONFIRM**
- **PKT_RESET_IND**
- **PKT_RESET_CONFIRM**
- **PKT_D_BIT_ACK**
- **PKT_CLEAR_IND** (except for sessions of type **SESSION_PVC**)

Execution Environment

The **x25sread** entry point can be called from the process environment only.

Return Values

EFAULT Indicates a buffer area was not valid.

EINVAL Indicates a parameter was not valid.

EIO Indicates an error has occurred. The `ext->status` field in the **x25_read_ext** structure contains one of the following values:

- **CIO_NOT_STARTED**
- **CIO_HARD_FAIL**
- **CIO_LOST_DATA**

EMSGSIZE Indicates that the buffer was not large enough to receive the packet data. The receiver data is preserved within the device driver until a read is issued with a large enough buffer.

EAGAIN Indicates that there were no packets to be read and the device was opened with the **DNDELAY** flag set.

Implementation Specifics

The **x25sread** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25smpx** entry point, **x25sopen** entry point, **x25swrite** entry point.

The **CIO_HALT** x25sioctl X.25 Device Handler Operation, **CIO_START** x25sioctl X.25 Device Handler Operation, **X25_REJECT** x25sioctl X.25 Device Handler Operation.

The **mbuf** structure in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

Data Transmission and Reception for the X.25 Device Handler in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

Sessions with the X.25 Device Handler in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

---

x25sselect X.25 Device Handler Entry Point

Purpose

Determines whether a specified event occurred on a device.

Syntax

```
int x25sselect (devno, events, reventp, chan)
dev_t devno;
ushort events;
ushort *reventp;
int chan;
```

Parameters

devno Specifies major and minor device numbers.

events Identifies the events to check. The *events* parameter is indicated by a bitwise OR using the following flags:

DPOLLIN
Checks if the receive data is available.

DPOLLOUT
Checks if transmission is possible. For the X.25 device handler, this event is always true.

DPOLLPRI
Checks if status is available.

DPOLLSYNC
Indicates the request is synchronous. The **x25sselect** entry point should *not* perform a **selnotify** kernel service if the events occur later.

reventp Returns the events pointer. The **x25sselect** entry point uses this parameter to indicate which of the selected events are true at the time of the call. The *reventp* parameter is indicated by a bitwise OR of the **DPOLLIN**, **DPOLLOUT**, or **DPOLLPRI** flag, as appropriate.

chan Identifies the channel number assigned by the **x25smpx** entry point.

Description

Note: This entry point should only be called by user-mode processes using the **select** or **poll** subroutine.

The **x25sselect** entry point determines if a specified event occurred on a device. If one or more events specified by the *events* parameter are true, this entry point updates the *reventp* parameter by setting the corresponding bits.

If none of the events are true, the *reventp* parameter is set to 0 (zero) and the entry point checks the **DPOLLSYNC** flag. If this flag is set, the request is synchronous and the entry point simply returns. If this flag is false, the **x25sselect** entry point records which events were requested. When one or more of the events subsequently becomes true, the **x25sselect** entry point calls the **selnotify** kernel subroutine to notify the user process.

When the X.25 device handler is in a state that prevents any of the events from being satisfied (such as an adapter failure), the **x25sselect** entry point sets the *reventp* parameter to 1 for the appropriate event. This prevents the **select** or **poll** subroutine from waiting indefinitely.

> **Note:** Unless the session protocol is **PROTOCOL_YBTS**, an X.25 packet sequence can not satisfy a **x25sselect** entry point until the final packet of the sequence is received or the sequence is otherwise terminated (for example, by the arrival of a clear indication).

Execution Environment

An **x25sselect** entry point can be called from the process environment only.

Return Values

A return code of -1 indicates an unsuccessful operation. The kernel sets the **errno** global variable to the following value:

EINVAL Indicates an invalid argument was specified or that the **x25sselect** entry point was called by a kernel-mode user.

Implementation Specifics

The **x25sselect** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

Select /Poll Logic for ddwrite and ddread Routines.

The **CIO_GET_FASTWRT** ddioctl Communications PDH entry point.

The **selnotify** kernel service.

The **poll** subroutine, **select** subroutine.

---

x25swrite X.25 Device Handler Entry Point

Purpose

Provides the means to send data to the X.25 adapter.

Syntax

```
int x25swrite (devno,uiop, chan, ext)
dev_t devno;
struct uio *uiop;
int chan;
struct x25_write_ext *ext;
```

Parameters

devno	Specifies major and minor device numbers.
uiop	Points to a **uio** structure.
chan	Identifies the channel number assigned by the **x25smpx** routine.
ext	Points to the **struct x25_write_ext** structure. This structure is found in the **/usr/include/sys/x25user.h** file.

Description

Note: Call-establishment or termination packets cannot be sent using this entry point.

The **x25swrite** entry point provides the means to send an X.25 data packet to the adapter.

uio Structure

For the **x25swrite** entry point, the *uiop* parameter is a pointer to a **uio** structure. This structure is described in the **/usr/include/sys/uio.h** file. The **uio** structure specifies the location and length of the caller's data.

This routine checks the `uiop->segflag` field to determine whether the data is in user space or kernel space. If the data is in kernel space, the `uiop->uio_ iov ->uio_base` field points to an **mbuf** structure chain containing the data for transmission. If the data is in user space, then the `uiop->uio_iov` field points to an array of **iovec** structures.

The data is in the form described by the **mbuf** structure. For a kernel-mode process, the **mbuf** structure containing the data should be pinned before making this call.

For session types of **SESSION_SVC_OUT**, **SESSION_SVC_IN**, or **SESSION_PVC**, the possible values for the `packet_type` field are the following:

- **PKT_DATA**
- **PKT_INT**
- **PKT_INT_CONFIRM**
- **PKT_RESET_REQ**
- **PKT_RESET_CONFIRM**
- **PKT_D_BIT_ACK**

> **Note:** For a **SESSION_MONITOR** session, the `packet_type` field must have a value of **PKT_MONITOR**.

If the value of the `packet_type` field is **PKT_DATA** and the data buffer is larger than the packet size, the data is transmitted as an X.25 packet sequence.

If a previous incoming data packet has been received with the D bit set, the incoming packet must be acknowledged with a **PKT_D_BIT_ACK** packet type before any further packets can be accepted by the device handler for this session.

x25_write_ext Parameter Block

For the **x25swrite** entry point, the *ext* parameter points to the **x25_write_ext** parameter block. The **x25_write_ext** structure contains a **write_extension** (**we**) structure and a `session_id` field.

The `we.flag` field consists of the bitwise OR of one or more of the following values:

CIO_NOFREE_MBUF Setting this bit causes the X.25 device handler to retain the **mbuf** structure after transmission is complete. If a kernel-mode process sets this bit, it must also do the following:

1. Determine when the X.25 device handler is finished with the **mbuf** structure.
2. Free the **mbuf** structure.

For a user-mode process, the device handler always frees the **mbuf** structure.

CIO_ACK_TX_DONE Setting this bit causes the X.25 device handler to acknowledge completion by building a **CIO_TX_DONE** status block for the caller when the write is complete.

The `we.status` field is meaningful only if the return value is **EIO**.

Execution Environment

The **x25swrite** entry point can be called from the process environment only.

Return Values

EINVAL Indicates that an invalid parameter was used or a write was made on a listen session.

EIO Indicates an error has occurred. The `ext->status` field contains one of the following common exception codes:

- **CIO_NOT_STARTED**
- **CIO_HARD_FAIL**
- **CIO_NOMBUF**
- **CIO_TIMEOUT**

Otherwise, the field contains one of the following X.25 specific codes:

X25_NO_ACK
A data packet with the D-bit set requires acknowledgment. Data packets cannot be sent until the acknowledgment is completed.

X25_NO_ACK_REQ
A **PKT_D_BIT_ACK** was sent and no packets required a D-bit acknowledgment.

X25_PROTOCOL
A protocol error occurred.

X25_RESET
The session is in reset state. The data packet could not be sent.

X25_CLEAR
The session has been cleared.

X25_NO_LINK
The X.25 link is not established.

X25_BAD_PKT_TYPE
The `packet_type` field passed in the *uiop* parameter block is not valid for the session type.

EAGAIN
Indicates that the transmit queue is full and the **DNDELAY** flag is set. The command was not accepted.

Implementation Specifics

The **x25swrite** entry point functions with an X.25 Interface Co-Processor/2 that has been correctly configured for use on a qualified network. Consult adapter specifications for more information on configuring the adapter and network qualifications.

Related Information

The **x25sread** entry point.

Common X.25 Device Handler Structures in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

Sessions with the X.25 Device Handler in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts*.

Appendix D. X.25 Cables and Connectors

The following information describes X.25 cable and connector configuration.

X.25 Coprocessor 37-Pin Connector Pin Assignments

The following table describes the X.25 coprocessor 37-pin connector pin assignments.

X.25 Coprocessor 37-Pin Connector Pin Assignments					
Pin	**Circuit Description**	**Symbol**	**X.21bis /V24**	**X.21bis /V35**	**X.21**
1	Reserved				
2	Transmitted data	TXD	x		
3	Received data	RXD	x		
4	Request to send	RTS	x	x	
5	Clear to send	CTS	x	x	
6	Data set ready	DSR	x	x	
7	Signal ground	GND	x	x	x
8	Carrier detect	CD	x	x	
9	Cable ID 0	ID0	x		x
10	Transmitted data (A)	T (A)			x
11	Control (A)	C (A)			x
12	Received data (A)	R (A)			x
13	Indication (A)	I (A)			x
14	Transmit clock (A)	S (A)			x
15	Cable ID 1	ID1	x	x	
16	Receive clock (B)	RX CLK (B)		x	
17	Transmitted data (B)	TXD (B)		x	
18	Transmit clock (B)	TX CLK (B)		x	
19	Receive data (B)	RXD (B)		x	
20	Data terminal ready	DTR	x	x	
21	Remote loopback test	RLBT	x		
22	Call indicate	CI	x	x	
23	Reserved				
24	Transmit clock	TX CLK	x		
25	Test indicate	TI	x		
26	Receive clock	RX CLK	x		
27	Local loopback test	LLBT			
28	Transmitted data (B)	T (B)			x

29	Control (B)	C (B)			x
30	Received data (B)	R (B)			x
31	Indication (B)	I (B)			x
32	Transmit clock (B)	S (B)			x
33	Reserved				
34	Receive clock (A)	RX CLK (A)		x	
35	Transmitted data (A)	TXD (A)		x	
36	Transmit clock (A)	RX CLK (A)		x	
37	Received data (A)	RXD (A)		x	
NOTES:					
(a) and (b) indicate pins that are associated to form pairs.					

Modem Attachment Pin Assignments

Supported modem types include the V.11, V.24/X.21bis, and V.35/X.21bis attachments.

X.21 Pin Assignments

The following table describes the pin assignments of the V.11 interface circuits for 15-pin connectors.

Pin Assignments of V.11 Interface Circuits to 15-Pin Connectors		
Pin Number	**Circuit Assignment**	**Circuit Description**
1[1]	-	-
2	T(a)[2]	Transmit
3	C(a)	Control
4	R(a)	Receive
5	I(a)	Indication
6	S(a)	Signal element timing
7	B(a)/X(a)	Byte timing: Modem transmit signal
8	G	Signal ground or common return
9	T(b)[2]	Transmit
10	C(b)	Control
11	R(b)	Receive
12	I(b)	Indication
13	S(b)	Signal element timing
14	B(b)/X(b)	Byte timing: Modem transmit signal
15	Future use	Future use
NOTES:		
[1] Assigned for connecting the shields between tandem sections of shielded interface cables.		
[2] (a) and (b) indicate pins that are associated to form pairs.		

G - Signal ground (or common return)	In the case of interchange circuits according to Recommendation V.11, it interconnects the zero volt reference points of a generator and a receiver to reduce environmental signal interference, if required.
T - Transmit	The binary signals originated by the system to be transmitted during the data transfer phase by way of the data circuit to one or more remote systems are transferred on this circuit to the modem. This circuit also transfers the call control signals originated by the system to be transmitted to the modem in the call establishment and other call control phases as specified by the relevant recommendations for the procedural characteristics of the interface. The modem monitors this circuit to detect electrical circuit fault conditions, according to the specifications of the electrical characteristics of the interface. A circuit fault is interpreted by the modem as defined in the recommendation for the procedural characteristics of the interface.
R - Receive	The binary signals sent by the modem as received during the data transfer phase from a remote system are transferred on this circuit to the system. This circuit also transfers the call control signals sent by the modem as received during the call establishment and other call control phases specified by the relevant recommendation for the procedural characteristics of the interface. The system monitors this circuit to detect electrical circuit fault conditions, according to the specifications of the electrical characteristics of the interface. A circuit fault is to be interpreted by the system as defined in the recommendations for the procedural characteristics of the interface.
C - Control	Signals on this circuit control the modem for a particular signalling process. Representation of a control signal requires additional coding of circuit T-Transmit as specified in the relevant recommendation for the procedural characteristics of the interface. During the data phase, this circuit must remain on. During the call control phases, the condition of this circuit must be as specified in the relevant recommendation for the procedural characteristics of the interface. The modem monitors this circuit to detect electrical circuit fault conditions, according to the specifications of the electrical characteristics of the interface. A circuit fault is to be interpreted by the modem as defined in the recommendation for the procedural characteristics of the interface.
I - Indication	Signals on this circuit indicate the state of the call control process. Representation of a control signal requires additional coding of circuit R-Receive, as specified in the relevant recommendation for the procedural characteristics of the interface. The on condition of this circuit signifies that signals on circuit R contain information from the distant system. The off condition signifies a control signalling condition which is defined by the bit sequence on circuit R as specified by the procedural characteristics of the interface. The system monitors this circuit to detect electrical circuit fault conditions, according to the specifications of the electrical characteristics of the interface. A circuit fault is to be interpreted as defined in the recommendation for the procedural characteristics of the interface.

S - Signal element timing

Signals on this circuit provide the system with signal element timing information from the modem. The condition of this circuit is on and off for nominally equal periods of time. However, for burst asynchronous operations, longer periods of off condition may be permitted equal to an integer odd number of the nominal period of the on condition as specified by the relevant procedural characteristics of the interface.

The system must present a binary signal on circuit T-Transmit and a condition on circuit C-Control, in which the transitions nominally occur prior to the transitions from off to on condition of this circuit.

The modem presents a binary signal on circuit R-Receive and a condition on circuit I-Indication in which the transitions nominally occur at the time of the transitions from off to on condition of this circuit.

The modem transfers signal-element timing information on this circuit across the interface whenever the timing source is capable of generating this information.

V.24/X.21bis Pin Assignments

The following table describes V.24/X.21bis pin assignment for a 25-pin connector (for speeds up to 20 Kbps).

V.24/X.21bis Pin Assignments for a 25-Pin Connector		
Pin	**CCITT Circuit**	**CCITT Circuit Designation**
1		
2	103	Transmitted data
3	104	Received data
4	105	Request to send
5	106	Ready for sending
6	107	Data set ready
7	102	Signal ground
8	109	N
9	-	N
10	-	N
11	-	N
12	-	N
13	-	F
14	-	F
15	114	Tx signal element timing from modem
16	-	F
17	115	Rx signal element timing from modem
18	141	Local loopback
19	-	F
20	108	Data terminal ready: Connect data set to line
21	140	Local loopback
22	125	Calling indicator
23	-	N
24	-	F

25	142	Test indicator
Notes: N=Pin permanently reserved for national use. F=Pin reserved for future international standard and should not be used for national use.		

V.35/X.21bis Pin Assignments

The following table describes the V.35/X.21bis pin assignments for a 34-pin connector (for speeds above 20 kbps).

V.35/X.21bis Pin Assignments for a 34-Pin Connector			
Pin	**CCITT Circuit**	**CCITT Circuit Designation**	**Direction**
A			
B	102	Signal ground or common return	Common
C	105	Request to send	From system
D	106	Ready for sending	To system
E	107	Data set ready	To system
F	109	Data channel received line signal detection	To system
H	108.1	Connect data set to line	From system
H	108.2	Data terminal ready	From system
J	125	Calling indicator	To system
K	141	Local loopback	From system
L	-	F	-
M	-	F	-
N	140	Loopback/Maintenance test	From system
P	103	Transmitted data A-wire	From system
R	104	Received data A-wire	To system
S	103	Transmitted data B-wire	From system
T	104	Received data B-wire	To system
U	113	Tx signal element timing from system A-wire	From system
V	115	Receiver signal element timing A-wire	To system
W	113	Tx signal element timing from system B-wire	From system
X	115	Receiver signal element timing B-wire	To system
Y	114	Tx signal element timing from modem A-wire	To system
Z	-	F	-
AA	114	Tx signal element timing from modem B-wire	To system
BB	-	F	-
CC	-	F	-
DD	-	F	-
EE	-	F	-
FF	-	F	-

HH	-	N	-
JJ	-	N	-
KK	-	N	-
LL	-	N	-
MM	-	F	-
NN	142	Test indicator	To system
Notes: N=Pin permanently reserved for national use. F=Pin reserved for future international standard and should not be used for national use.			

V.36 Pin Assignments

The following table describes the V.36 pin assignments for a 37-pin connector:

ARTIC960 6-Port V.36 37-Pin D-Shell Connector Pins			
Pin	**CCITT Circuit**	**Signal Name**	**Direction**
04	103	TXDnA	From system
22	103	TXDnB	From system
06	104	RXDnA	To system
24	104	RXDnB	To system
05	114	TCLKInA	To system
23	114	TCLKInB	To system
08	115	RCLKInA	To system
26	115	RCLKInB	To system
17	113	TCLKOnA	From system
35	113	TCLKOnB	From system
07	105	RTSn	From system
09	106	CTSn	To system
13	109	DCDn	To system
12	108.2	DTRn	From system
11	107	DSRn	To system
19	102	SigGnd(-)	
01	—	Shield(-)	

X.25 Interconnection Cables

The X.25 protocol supports the X.25 interface cables.

X.21 Interface Cable

D-37 Connector		**D-15 Connector**		**Function**
T(A)	10	2	T(A)	Transmitted data (A)
C(A)	11	3	C(A)	Control (A)
R(A)	12	4	R(A)	Received data (A)
I(A)	13	5	I(A)	Indication (A)

S(A)	14	6	S(A)	Transmit clock (A)
GND	7	8	GND	Signal ground
T(B)	28	9	T(B)	Transmitted data (B)
C(B)	29	10	C(B)	Control (B)
R(B)	30	11	R(B)	Received data (B)
I(B)	31	12	I(B)	Indication (B)
S(B)	32	13	S(B)	Transmit clock (B)
ID0	9*			

* Wired to pin 7 of the same cable

X.21bis/V.24 Interface Cable

D-37 Connector		D-25 Connector		Function
TXD	2	2	TXD	Transmitted data
RXD	3	3	RXD	Received data
RTS	4	4	RTS	Request to send
CTS	5	5	CTS	Clear to send
DSR	6	6	DSR	Data set ready
GND	7	7	GND	Signal ground
CD	8	8	CD	Carrier detect
TX CLK	24	15	TX CLK	Transmit click
RX CLK	26	17	RX CLK	Receive clock
LLBT	27	18	LLBT	Local loopback test
DTR	20	20	DTR	Data terminal ready
RLBT	21	21	RLBT	Remote loopback test
CI	22	22	CI	Call indicated
TI	25	25	TI	Test indicated
ID0	9*			
ID1	15*			

* Wired to pin 7 of the same cable

X.21bis/V.35 Interface Cable

D-37 Connector		M/34 Connector		Function
GND	7	B	GND	Signal ground
RTS	4	C	RTS	Request to send
CTS	5	D	CTS	Clear to send
DSR	6	E	DSR	Data set ready
CD	8	F	CD	Carrier detect
DTR	20	H	DTR	Data terminal ready
CI	22	J	CI	Call indicate
TXD (A)	35	P	TXD (A)	Transmitted data (A)
RXD (A)	37	R	RXD (A)	Receive data (A)

TXD (B)	17	S	TXD (B)	Transmitted data (B)
RXD (B)	19	T	RXD (B)	Receive data (B)
RX CLK (A)	34	V	RX CLK (A)	Receive clock (A)
TX CLK (A)	36	Y	TX CLK (A)	Transmit clock (A)
RX CLK (B)	16	X	RX CLK (B)	Receive clock (B)
TX CLK (B)	18	AA	TX CLK (B)	Transmit clock (A)
ID1	15			

* Wired to pin 7 of the same cable

The X.25 protocol supports the X.21, X.21bis/V.24, and X.21bis/V.35 interface cables.

6-Port X.21 Portmaster Adapter

X.21 Interface Cable

D-25 Connector		D-15 Connector		Function
Signal	**Pin**	**Pin**	**Signal**	
T(A)	2	2	T(A)	Transmitted data (A)
C(A)	4	3	C(A)	Control (A)
R(A)	3	4	R(A)	Received data (A)
I(A)	5	5	I(A)	Indication (A)
S(A)	15	6	S(A)	Receive clock (A)
GND	7	8	GND	Signal ground
T(B)	24	9	T(B)	Transmitted data (B)
C(B)	20	10	C(B)	Control (B)
R(B)	17	11	R(B)	Received data (B)
I(B)	6	12	I(B)	Indication (B)
S(B)	23	13	S(B)	Receive clock (B)
ID0			ID0	
X(A)	8	7	X(A)	Transmit clock (A)
X(B)	22	14	X(B)	Transmit clock (B)

X.21bis/V.24 for 8-Port Portmaster and 8-Port ARTIC960 Adapters

D-25 Connector		D-25 Connector		Function
Signal	**Pin**	**Pin**	**Signal**	
TXT	2	2	TXD	Transmitted data
RXD	3	3	RXD	Received data
RTS	4	4	RTS	Request to send
CTS	5	5	CTS	Clear to send
DSR	6	6	DSR	Data set ready
GND	7	7	GND	Signal ground
CD	8	8	CD	Carrier detect

TX CLK	15	15	TX CLK	Transmit clock (in)
RX CLK	17	17	RX CLK	Receive clock (in)
DTR	20	20	DTR	Data terminal ready
DTECLK	24	24	DTECLK	DTE clock (out)
HRS	23	23	HRS	Half rate select

6-Port /V.35 Portmaster and 6-Port/V.36 ARTIC960 Adapters (V.35 Configuration)

D-25 Connector		D-25 Connector		
Signal	**Pin**	**Pin**	**Signal**	**Function**
GND	7	B	GND	Signal ground
RTS	4	C	RTS	Request to send
CTS	5	D	CTS	Clear to send
DSR	6	E	DSR	Data set ready
CD	8	F	CD	Carrier detect
DTR	20	H	DTR	Data terminal ready
TXD (A)	2	P	TXD (A)	Transmitted data (A)
RXD (A)	3	R	RXD (A)	Receive data (A)
TXD (B)	14	S	TXD (B)	Transmitted data (B)
RXD (B)	16	T	RXD (B)	Receive data (B)
RX CLK (A)	17	V	RX CLK (A)	Receive clock (A)
TX CLK (A)	15	Y	TX CLK (A)	Transmit clock (A)
RX CLK (B)	9	X	RX CLK (B)	Receive clock (B)
TX CLK (B)	12	AA	TX CLK (B)	Transmit clock (B)
DTE CLK (A)	24	U	DTE CLK (A)	DTE clock (A)
DTE CLK (B)	11	W	DTE CLK (B)	DTE clock (B)

X.25 Adapter and Cable Diagnostics Wrap Plugs and Pinouts

Adapter and cable wrap plugs, used for diagnostics, are automatically included with cables and X.25 adapter orders specifying the system order numbers.

The following table lists the wrap plug pin assignments. They are supplied for local loopback tests in accordance with CCITT Recommendation X.150.

Table 1.

X.25 Co-Processor	PORTMASTER	ARTIC960
D-37 wrap plug	6-Port V.21 D-15 wrap plug	8-Port X.21 D-15 wrap plug
D-15 wrap plug	6-Port V.35 wrap plug	8-Port EIA-232E D-25 wrap plug
D-25 wrap plug	6-Port X.21bis/.24 D-25 wrap plug	
M/34 wrap plug	6-Port X.35 D-25 wrap plug	6-Port V.36 DB-37 wrap plug
	6-Port X.21 D-78 wrap plug	8-Port X.21 D-100 wrap plug
	6-Port V.35 D-100 wrap plug	6-Port V.36 D-100 wrap plug

Table 1. (continued)

	6-Port X.21bis/.24 D-100 wrap plug	8-Port X.21bis/EIA-232E D-100 wrap plug

X.25 Co-Processor Adapter D-37 Wrap Plug

The D-37 Wrap plug is used to test local loopback at the D-37 connector on the adapter. It has a cable identifier of ID0=1, and ID1=1. The D-37 Wrap Plug table shows the pin assignments.

D-37 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
T(B)	28	30	R(B)
T(A)	10	12	R(A)
C(B)	29	31	I(B)
C(A)	11	13	I(A)
TXD	2	3	RXD
RTS	4	5	CTS
DTR	20	6	DSR
LLBT	27	25	TI
RBLT	21	22	CI
TXD(A)	35	37	RXD(A)
TXD(B)	17	19	RXD(B)
ID0	9	15	ID1

X.25 Co-Processor Adapter D-15 Wrap Plug

The D-15 wrap plug is used to test loopback at the DCE end of the X.21 interface cable. The D-15 Wrap Plug table shows the pin assignments.

D-15 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
T(B)	9	11	R(B)
T(A)	2	4	R(A)
C(B)	10	12	I(B)
C(A)	3	5	I(A)

X.25 Co-Processor Adapter D-25 Wrap Plug

The D-25 wrap plug is used to test loopback at the DCE end of the X.21bis/V.24 interface cable. The D-25 Wrap Plug table shows the pin assignments.

D-25 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD	2	3	RXD
RTS	4	5	CTS
DTR	20	6	DSR
LLBT	18	25	TI
RBLT	21	22	CI

X.25 Co-Processor Adapter M/34 Wrap Plug

The M/34 wrap plug is used to test loopback at the DCE end of the X.21bis/V.35 interface cable. The M/34 Wrap Plug table shows the pin assignments.

M/34 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD(A)	P	R	RXD(A)
TXD(B)	S	T	RXD(B)
RTS	C	D - F	CTS - DCD
DTR	H	E	DSR
DTECLK(A)	U	V	RXCLK(A)
DTECLK(B)	W	X	RXCLK(B)

6-Port X.21 Portmaster and 8-Port X.21 ARTIC960 D-15 Wrap Plug

6-Port X.21 Portmaster and ARTIC960 D-15 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
T(B)	9	11	R(B)
T(A)	2	4	R(A)
C(B)	10	12	I(B)
C(A)	3	5	I(A)
X(A)	7	6	S(A)
X(B)	14	13	S(B)

8-Port V.24 Portmaster and 8-Port EIA-232E ARTIC960 D-25 Wrap Plug

8-Port V.24 Portmaster and 8-Port EIA-232E ARTIC960 D-25 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD	2	3	RXD
RTS	4	5 - 8 - 15	CTS - DCD - TXCLKIN
DTR	20	6 - 22 - 23	DSR - RI - HRS
TXCLK	24	17	RXCLK

6-Port V.35 Portmaster M/34 Wrap Plug

6-Port V.35 Portmaster M/34 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD(A)	P	R-Y	RXD(A), TXCLK(A)
TXD(B)	S	T-AA	RXD(B), TXCLK(B)
RTS	C	D-F	CTS, DCD
DTR	H	E	DSR
DTE CLK(A)	U	V	RX CLK(A)

DTE CLK(B)	W	X	RX CLK(B)

6-Port X.21 Portmaster D-25 Wrap Plug

6-Port X.21 Portmaster D-25 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
T(B)	2	3	R(B)
T(A)	24	17	R(A)
C(B)	4	5	I(B)
C(A)	20	6	I(A)
X(A)	8	15	S(A)
X(B)	22	23	S(B)

6-Port V.35 Portmaster Wrap Plug

6-Port V.35 Portmaster Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD(A)	2	3-5	RXD(A), TXCLK(A)
TXD(B)	14	16-12	RXD(B), TXCLK(B)
RTS	4	5-8	CTS, DCD
DTR	20	6	DSR
DTE CLK(A)	24	17	RX CLK(A)
DTE CLK(B)	11	9	RX CLK(B)

6-Port V.36 ARTIC960 DB-37 Wrap Plug

6-Port V.36 ARTIC960 DB-37 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD(A)	4	6-5	RXD(A), TCLK(A)
TXD(B)	22	24-23	RXD(B), TCLK(B)
TCLK(A)	17	8	RCLK(A)
TCLK(B)	35	26	RCLK(B)
RTS	7	9-13	CTS, DCD
DTR	12	11	DSR

IBM ARTIC960Hx 4-Port Selectable PCI Adapter V.35 M/34 Wrap Plug

IBM ARTIC960Hx 4-Port Selectable PCI Adapter V.35 M/34 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD(A)	P	R - Y	RXD(A), TXCLK(A)
TXD(B)	S	T - AA	RXD(B), TXCLK(B)
RTS	C	D	CTS
DTR	H	E - F	DSR, DCD

DTE CLK(A)	U	V	RX CLK(A)
DTE CLK(B)	W	X	RX CLK(B)

IBM ARTIC960Hx 4-Port Selectable PCI Adapter EIA-232 D-25 Wrap Plug

IBM ARTIC960Hx 4-Port Selectable PCI Adapter EIA-232 D-25 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
TXD	2	3 - 15	RXD, TXCLKIN
RTS	4	5	CTS
DTR	20	6 - 8	DSR, DCD
TXCLK	24	17	RXCLK

IBM ARTIC960Hx 4-Port Selectable PCI Adapter X.21 D-15 Wrap Plug

IBM ARTIC960Hx 4-Port Selectable PCI Adapter X.21 D-15 Wrap Plug			
Signal	**Pin Number**	**Pin Number**	**Signal**
T(B)	9	11	R(B)
T(A)	2	4	R(A)
C(B)	10	12	I(B)
C(A)	3	5	I(A)
X(A)	7	6	S(A)
X(B)	14	13	S(B)

6-Port X.21 Portmaster D-78 Wrap Plug

6-Port X.21 Portmaster D-78 Wrap Plug		
Port	**Connector K9B Pins**	**Signal Wrapped**
0	40 —>02	T0(A) —> R0(A)
	41 —> 62	T0(B) —> R0(B)
	01 —> 61	C0(A) —> I0(A)
	60 —> 42	C0(B) —> I0(B)
	22 —> 23	X0(A) —> S0(A)
	03 —> 21	X0(B) —> S0(B)
1	04 —>64	T1(A) —> R1(A)
	05 —> 26	T1(B) —> R1(B)
	63 —> 25	C1(A) —> I1(A)
	24 —> 06	C1(B) —> I1(B)
	45 —> 46	X1(A) —> S1(A)
	65 —> 44	X1(B) —> S1(B)
2	66 —>28	T2(A) —> R2(A)
	19 —> 57	T2(B) —> R2(B)

	27 —> 48	C2(A) —> I2(A)
	47 —> 68	C2(B) —> I2(B)
	09 —> 78	X2(A) —> S2(A)
	29 —> 76	X2(B) —> S2(B)
3	69 —>31	T3(A) —> R3(A)
	20 —> 77	T3(B) —> R3(B)
	30 —> 51	C3(A) —> I3(A)
	50 —> 71	C3(B) —> I3(B)
	12 —> 59	X3(A) —> S3(A)
	32 —> 37	X3(B) —> S3(B)
4	73 —>54	T4(A) —> R4(A)
	10 —> 18	T4(B) —> R4(B)
	34 —> 15	C4(A) —> I4(A)
	35 —> 72	C4(B) —> I4(B)
	74 —> 39	X4(A) —> S4(A)
	49 —> 38	X4(B) —> S4(B)
5	55 —>75	T5(A) —> R5(A)
	13 —> 53	T5(B) —> R5(B)
	16 —> 36	C5(A) —> I5(A)
	17 —> 33	C5(B) —> I5(B)
	56 —> 14	X5(A) —> S5(A)
	52 —> 58	X5(B) —> S5(B)

8-Port X.21 ARTIC960 D-100 Wrap Plug

8-Port X.21 ARTIC960 D-100 Wrap Plug		
Port	**Connector K9B Pins**	**Signal Wrapped**
0	01 —> 52	T0(A) —> R0(A)
	27 —> 76	T0(B) —> R0(B)
	03 —> 54	TCLK0(A) —> RCLK0(A)
	29 —> 78	TCLK0(B) —> RCLK0(B)
	02 —> 53	C0(A) —> I0(A)
	28 —> 77	C0(B) —> I0(B)
1	04 —> 55	T1(A) —> R1(A)
	30 —> 79	T1(B) —> R1(B)
	07 —> 57	TCLK1(A) —> RCLK1(A)
	32 —> 81	TCLK1(B) —> RCLK1(B)
	05 —> 56	C1(A) —> I1(A)
	31 —> 80	C1(B) —> I1(B)
2	08 —> 58	T2(A) —> R2(A)
	33 —> 82	T2(B) —> R2(B)
	10 —> 60	TCLK2(A) —> RCLK2(A)

	35 —> 84	TCLK2(B) —> RCLK2(B)
	09 —> 59	C2(A) —> I2(A)
	34 —> 83	C2(B) —> I2(B)
3	11 —> 61	T3(A) —> R3(A)
	36 —> 85	T3(B) —> R3(B)
	13 —> 63	TCLK3(A) —> RCLK3(A)
	39 —> 87	TCLK3(B) —> RCLK3(B)
	12 —> 62	C3(A) —> I3(A)
	37 —> 86	C3(B) —> I3(B)
4	14 —> 64	T4(A) —> R4(A)
	40 —> 88	T4(B) —> R4(B)
	16 —> 66	TCLK4(A) —> RCLK4(A)
	42 —> 90	TCLK4(B) —> RCLK4(B)
	15 —> 65	C4(A) —> I4(A)
	41 —> 89	C4(B) —> I4(B)
5	17 —> 67	T5(A) —> R5(A)
	43 —> 91	T5(B) —> R5(B)
	20 —> 69	TCLK5(A) —> RCLK5(A)
	45 —> 93	TCLK5(B) —> RCLK5(B)
	18 —> 68	C5(A) —> I5(A)
	19 —> 92	C5(B) —> I5(B)
6	21 —> 70	T6(A) —> R6(A)
	46 —> 94	T6(B) —> R6(B)
	23 —> 72	TCLK6(A) —> RCLK6(A)
	48 —> 96	TCLK6(B) —> RCLK6(B)
	22 —> 71	C6(A) —> I6(A)
	47 —> 95	C6(B) —> I6(B)
7	24 —> 73	T7(A) —> R7(A)
	49 —> 97	T7(B) —> R7(B)
	26 —> 99	TCLK7(A) —> RCLK7(A)
	51 —> 100	TCLK7(B) —> RCLK7(B)
	50 —> 74	C7(A) —> I7(A)
	75 —> 98	C7(B) —> I7(B)

6-Port V.35 Portmaster and 6-Port V.36 ARTIC960 D-100 Wrap Plug

6-Port V.35 Portmaster and 6-Port V.36 ARTIC960 D-100 Wrap Plug		
Port	**Connector K9B Pins**	**Signal Wrapped**
0	94 - 08 - 76	TXDA0 - RXDA0 - TXCA0 IN
	70 - 33 - 52	TXDB0 - RXDB0 - TXCB0 IN
	24 - 20	TXCA0 OUT - RXCA0
	49 - 45	TXCB0 OUT - RXCB0

	42 - 15 - 89	RTS0 - CTS0 - DCD0
	18 - 66	DTR0 - DSR0
1	21 - 54 - 06	TXDA1 - RXDA1 - TXCA1 IN
	46 - 78 - 31	TXDB1 - RXDB1 - TXCB1 IN
	73 - 41	TXCA1 OUT - RXCA1
	97 - 16	TXCB1 OUT - RXCB1
	43 - 65 - 40	RTS1 - CTS1 - DCD1
	91 - 90	DTR1 - DSR1
2	47 - 58 - 77	TXDA2 - RXDA2 - TXCA2 IN
	22 - 82 - 53	TXDB2 - RXDB2 - TXCB2 IN
	98 - 38	TXCA2 OUT - RXCA2
	74 - 13	TXCB2 OUT - RXCB2
	92 - 86 - 62	RTS2 - CTS2- DCD2
	69 - 88	DTR2 - DSR2
3	71 - 29 - 56	TXDA3 - RXDA3 - TXCA3 IN
	95 - 04 - 80	TXDB3 - RXDB3 - TXCB3 IN
	25 - 19	TXCA3 OUT - RXCA3
	50 - 44	TXCB3 OUT - RXCB3
	93 - 87 - 61	RTS3 - CTS3 - DCD3
	68 - 64	DTR3 - DSR3
4	72 - 28 - 27	TXDA4 - RXDA4 - TXCA4 IN
	96 - 03 - 02	TXDB4 - RXDB4 - TXCB4 IN
	99 - 32	TXCA4 OUT - RXCA4
	75 - 07	TXCB4 OUT - RXCB4
	37 - 59 - 35	RTS4 - CTS4 - DCD4
	14 - 60	DTR4 - DSR4
5	23 - 57 - 55	TXDA5 - RXDA5 - TXCA5 IN
	48 - 81 -79	TXDB5 - RXDB5 - TXCB5 IN
	26 - 30	TXCA5 OUT - RXCA5
	51 - 05	TXCB5 OUT - RXCB5
	39 - 09 - 84	RTS5 - CTS5 - DCD5
	12 - 85	DTR5 - DSR5

8-Port X.21bis/V.24 Portmaster and 8-Port X.21bis/V.24 ARTIC960 D-100 Wrap Plug

8-Port X.21bis/V.24 Portmaster and 8-Port X.21bis/V.24 ARTIC960 D-100 Wrap Plug		
Port	**Connector K9B Pins**	**Signal Wrapped**
0	51 - 02	TXD0 - RXDA0
	01 - 77 - 28 - 29	RTS0 - CTS0 - DCD0 - TXCLKIN0
	52 - 78	TXCLK0 - RXCLK0
	76 - 53 - 03 - 27	DTR0 - DSR0 - RI0 - HRS0

1	54 - 05	TXD1 - RXDA1
	04 - 80 - 31 - 32	RTS1 - CTS1 - DCD1 - TXCLKIN1
	55 - 81	TXCLK1 - RXCLK1
	79 - 56 - 06 - 30	DTR1 - DSR1 - RI1 - HRS1
2	07 - 83	TXD2 - RXDA2
	82 - 34 - 59 - 60	RTS2 - CTS2 - DCD2 - TXCLKIN2
	08 - 35	TXCLK2 - RXCLK2
	33 - 09 - 84 - 58	DTR2 - DSR2 - RI2 - HRS2
3	10 - 86	TXD3 - RXDA3
	85 - 37 - 63 - 63	RTS3 - CTS3 - DCD3 - TXCLKIN3
	11 - 38	TXCLK3 - RXCLK3
	36 - 12 - 87 - 61	DTR3 - DSR3 - RI3 - HRS3
4	13 - 89	TXD4 - RXDA4
	88 - 40 - 65 - 66	RTS4 - CTS4 - DCD4 - TXCLKIN4
	14 - 41	TXCLK4 - RXCLK4
	39 - 15 - 90 - 64	DTR4 - DSR4 - RI4 - HRS4
5	16 - 92	TXD5 - RXDA5
	91 - 43 - 68 - 69	RTS5 - CTS5 - DCD5 - TXCLKIN5
	17 - 44	TXCLK5 - RXCLK5
	42 - 18 - 93 - 97	DTR5 - DSR5 - RI5 - HRS5
6	94 - 46	TXD6 - RXDA6
	45 - 71 - 21 - 22	RTS6 - CTS6 - DCD6 - TXCLKIN6
	95 - 72	TXCLK6 - RXCLK6
	70 - 96 - 47 - 20	DTR6 - DSR6 - RI6 - HRS6
7	48 - 74	TXD7 - RXDA7
	73 - 24 - 99 - 100	RTS7 - CTS7 - DCD7 - TXCLKIN7
	49 - 25	TXCLK7 - RXCLK7
	23 - 50 - 75 - 98	DTR7 - DSR7 - RI7 - HRS7

IBM ARTIC960Hx 4-Port Selectable PCI Adapter 120-Pin Wrap Plug

The 120-Pin Wrap plug is used to test local loopback at the 120-Pin connector on the adapter. It has a cable identifier of FEh.

IBM ARTIC960Hx 4-Port Selectable PCI Adapter 120-Pin Wrap Plug		
Port	**Connector Pins**	**Signal Wrapped**
0	118 - 96 - 102	TXD(A) - RXD(A) - TXC(A) IN
	119 - 97 - 103	TXD(B) - RXD(B) - TXC(B) IN
	105 - 104 - 91	TXD12V - RXD12V - TXC12V IN
	116 - 108	TXC(A) OUT - RXC(A)
	117 - 109	TXC(B) OUT - RXC(B)
	111 - 106	TXC12V OUT - RXC12V IN
	114 - 100	RTS(A) - CTS(A)

	115 - 101	RTS(B) - CTS(B)
	114 - 100	RTS12V - CTS12V
	112 - 98 - 94	DTR(A) - DSR(A) - DCD(A)
	113 - 99 - 95	DTR(B) - DSR(B) - DCD(B)
	112 - 98 - 94	DTR12V - DSR12V - DCD12V
1	58 - 36 - 42	TXD(A) - RXD(A) - TXC(A) IN
	59 - 37 - 43	TXD(B) - RXD(B) - TXC(B) IN
	45 - 44 - 31	TXD12V - RXD12V - TXC12V IN
	56 - 48	TXC(A) OUT - RXC(A)
	57 - 49	TXC(B) OUT - RXC(B)
	51 - 46	TXC12V OUT - RXC12V IN
	54 - 40	RTS(A) - CTS(A)
	55 - 41	RTS(B) - CTS(B)
	54 - 40	RTS12V - CTS12V
	52 - 38 - 34	DTR(A) - DSR(A) - DCD(A)
	53 - 39 - 35	DTR(B) - DSR(B) - DCD(B)
	52 - 38 - 34	DTR12V - DSR12V - DCD12V
2	2 - 24 - 18	TXD(A) - RXD(A) - TXC(A) IN
	3 - 25 - 19	TXD(B) - RXD(B) - TXC(B) IN
	17 - 16 - 30	TXD12V - RXD12V - TXC12V IN
	4 - 12	TXC(A) OUT - RXC(A)
	5 - 13	TXC(B) OUT - RXC(B)
	11 - 1	TXC12V OUT - RXC12V IN
	6 - 20	RTS(A) - CTS(A)
	7 - 21	RTS(B) - CTS(B)
	6 - 20	RTS12V - CTS12V
	8 - 22 - 26	DTR(A) - DSR(A) - DCD(A)
	9 - 23 - 27	DTR(B) - DSR(B) - DCD(B)
	8 - 22 - 26	DTR12V - DSR12V - DCD12V
3	62 - 84 - 78	TXD(A) - RXD(A) - TXC(A) IN
	63 - 85 - 79	TXD(B) - RXD(B) - TXC(B) IN
	77 - 76 - 90	TXD12V - RXD12V - TXC12V IN
	64 - 72	TXC(A) OUT - RXC(A)
	65 - 73	TXC(B) OUT - RXC(B)
	71 - 61	TXC12V OUT - RXC12V IN
	66 - 80	RTS(A) - CTS(A)
	67 - 81	RTS(B) - CTS(B)
	66 - 80	RTS12V - CTS12V
	68 - 82 - 86	DTR(A) - DSR(A) - DCD(A)
	69 - 83 - 87	DTR(B) - DSR(B) - DCD(B)
	68 - 82 - 86	DTR12V - DSR12V - DCD12V

Note: The DTR(B) signal is pulled to ground with a 10k Ohm resistor in the wrap plug to allow testing of both 5-volt balanced and 12-volt signals

Appendix E. CCITT Causes and Diagnostics

The following information includes the X.25 clear and reset causes and diagnostic codes as defined by the CCITT.

X.25 Clear and Reset Codes

Each clear-indication or reset-indication packet includes a 1-byte cause code and a 1-byte diagnostic code in the received data. The API subroutines take no specific action on any of the cause or diagnostic codes.

Origins of the Clear or Reset

X.25 clear-indication and reset-indication packets may be generated by the X.25 code, the remote data terminal equipment (DTE), or the X.25 network itself. The relationship between cause codes and diagnostic codes is shown in a table in the annexes to ISO 8208. The following sections discuss X.25 clear and reset codes:

- CCITT Clear and Reset Cause for X.25
- X.25 Logical Channel States .
- X.25 Diagnostic Codes .
- SNA Diagnostic Codes .

Notes:

1. When Communications Server (SNA) is being used, use the SNA diagnostic codes; otherwise, use the CCITT/ISO diagnostic codes.
2. **x25mon** shows codes in decimal.

CCITT Restart Causes

The CCITT meanings for the restart cause codes are:

Hex	Meaning
00	Originated by the remote X.25 data terminal equipment (DTE).
01	Local procedure error.
03	Network congestion.
07	Network up.

X.25 Logical Channel States

This information lists the CCITT logical channel states referred to in X.25 Diagnostic Codes .

State	Meaning
d1	Flow control ready.
d2	DTE reset request.
d3	DCE reset indication.
p1	Channel ready.
p2	DTE call request.
p3	DCE incoming call.
p4	Data transfer.
p5	Call collision.
p6	DTE clear request.
p7	DCE clear indication.
r1	Packet level ready.

State	Meaning
r2	DTE restart request.
r3	DCE restart indication.

X.25 Diagnostic Codes

Diagnostic codes give additional information about the reason for sending a clear-indication or reset-indication. (The reason is also indicated in the cause code.) The meaning of each diagnostic code depends on whether X.25 is being used as a medium for SNA communications, by means of qualified logical link control (QLLC), or being used directly. If SNA is being used, refer to the SNA Diagnostic Codes; if X.25 is being used directly, refer to the CCITT/ISO Diagnostic Codes sections that follow.

In addition, some diagnostic codes are used by the **xtalk** command.

CCITT Clear and Reset Causes for X.25

This section gives the CCITT meanings for cause codes given in clear-indication and reset-indication packets:

- CCITT clear causes.
- CCITT reset causes.
- CCITT restart causes.

> **Note:** A RESET procedure is invoked to recover a single virtual circuit. A RESTART procedure is invoked when all virtual circuits on a link are to be reinitialized.

The CCITT defines the X.25 clear cause codes as follows:

List of CCITT Clear Causes		
Hex	**Dec**	**Meaning**
00	00	Originated by the remote X.25 data terminal equipment (DTE).
01	01	Number busy.
03	03	Incorrect facility request.
05	05	Network congestion.
09	09	Out of order.
0B	11	Access barred.
0D	13	Not obtainable.
11	17	Remote procedure error.
13	19	Local procedure error.
15	21	RPOA out of order.
19	25	Reverse charging acceptance not subscribed.
21	33	Incompatible destination.
29	41	Fast select acceptance not subscribed.
80	128	80 through FF not defined by CCITT, but used by SNA.

The CCITT defines the X.25 reset cause codes as follows:

List of CCITT Reset Causes

Hex	Dec	Meaning
00	00	Originated by the remote X.25 data terminal equipment (DTE).
01	01	Out of order.
03	03	Remote procedure error.
05	05	Local procedure error.
07	07	Network congestion.
09	09	Remote DTE operational.
0F	15	Network operational.
11	17	Incompatible destination.
1D	19	Network out of order.
80	128	Originated through FF DTE.

Diagnostic Codes for X.25 and Communications Server (SNA)

The CCITT has defined diagnostic codes for X.25 communications and Communications Server (SNA).

CCITT Diagnostic Codes

The CCITT defines the X.25 diagnostic codes as follows:

List of CCITT Diagnostic Codes		
Hex	**Dec**	**Meaning**
00	0	Clear or reset generated during restart.
01	1	Incorrect P(S) in packet from DCE.
02	2	Incorrect P(R) in packet from DCE.
10	16	Incorrect packet type.
11	17	Incorrect packet from DCE for state r1.
12	18	Incorrect packet from DCE for state r2.
13	19	Incorrect packet from DCE for state r3.
14	20	Incorrect packet from DCE for state p1.
15	21	Incorrect packet from DCE for state p2.
16	22	Incorrect packet from DCE for state p3.
17	23	Incorrect packet from DCE for state p4.
18	24	Incorrect packet from DCE for state p5.
19	25	Incorrect packet from DCE for state p6.
1A	26	Incorrect packet from DCE for state p7.
1B	27	Incorrect packet from DCE for state d1.
1C	28	Incorrect packet from DCE for state d2.
1D	29	Incorrect packet from DCE for state d3.
20	32	Packet not allowed.
21	33	Unidentifiable packet received from DCE.
22	34	Call received on one-way channel.
23	35	Clear or call packet received from DCE on a permanent virtual circuit (PVC).

24	36	Packet received on an unassigned logical channel.
25	37	REJECT not subscribed.
26	38	Packet received from DCE was too short.
27	39	Packet received from DCE was too long.
28	40	Incorrect general format identifier (GFI).
29	41	Restart packet received from DCE with non-zero logical channel identifier.
2A	42	Incorrect fast-select packet received from DCE.
2B	43	Unauthorized interrupt confirmation.
2C	44	Interrupt packet received from DCE when acknowledgment was still outstanding.
2D	45	Unauthorized reject.
30	48	Timer expired (or limit surpassed).
31	49	Timeout or retries reached on call response from DCE.
32	50	Timeout or retries reached on clear response from DCE.
33	51	Timeout or retries reached on reset response from DCE.
34	52	Timeout or retries reached on restart response from DCE.
35	53	Time expired for call deflection.
40	64	Call setup clearing or registration problem.
41	65	Facility/registration code not allowed.
42	66	Facility not allowed.
43	67	Incorrect called address.
44	68	Invalid address in incoming call from DCE.
45	69	Incorrect facility/registration length field.
46	70	Incoming call barred.
47	71	No logical channel available.
48	72	Call collision.
49	73	Duplicate facility requested.
4A	74	Nonzero address length in fast-select clear from DCE.
4B	75	Nonzero facility length in fast-select clear from DCE.
4C	76	Facility not provided when expected.
4D	77	Incorrect CCITT-specified DTE facility.
4E	78	Maximum number of call redirections or call deflections exceeded.
50	80	Miscellaneous.
51	81	Improper cause code from DTE.
52	82	Non-octet aligned.
53	83	Inconsistent Q-bit settings.
54	84	NUI problem.
70	112	International problem.
71	113	Remote network problem.
72	114	International protocol problem.
73	115	International link out of order.

74	116	International link busy.
75	117	Transit network facility problem.
76	118	Remote network facility problem.
77	119	International routing problem.
78	120	Temporary routing problem.
79	121	Unknown called DNIC.
7A	122	Maintenance action (may also apply within a national network).
80	128	Reserved for DTE-defined diagnostic information.

X.25 Licensed Program Specific Diagnostic Codes

The product-specific diagnostic codes are defined as follows:

X.25 Licensed Program Specific Diagnostics		
Hex	**Dec**	**Meaning**
81	129	No listener for incoming call.
82	130	No available LCN for call.
83	131	User error.
84	132	User call rejection.
85	133	Call cleared before accept.
86	134	Invalid call reference.
87	135	Registration timer expired.
88	136	Invalid Link layer state.

ISO 8208 Diagnostic Codes

The ISO 8208 diagnostic codes are defined as follows:

ISO 8208 Diagnostics		
Hex	**Dec**	**Meaning**
90	144	Timer expired or retransmission count surpassed.
91	145	Timer expired or retransmission count surpassed on interrupt-confirm from DCE.
92	146	T25 timer expired for data packet transmission.
93	147	Timer expired or retransmission count surpassed for reject.
A0	160	DTE-specific signals.
A1	161	DTE operational.
A2	162	DTE not operational (level 2) or no application listening (network).
A3	163	DTE resource constraint.
A4	164	Fast select not subscribed.
A5	165	Incorrect partially full data packet received from DCE.
A6	166	D-bit procedure not supported.
A7	167	Registration or cancellation confirmed.
E0	224	OSI Network Service Problem.
E1	225	Disconnection (transient condition).

E2	226	Disconnection (permanent condition).
E3	227	Connection rejection - reason unspecified (transient condition).
E4	228	Connection rejection - reason unspecified (permanent condition).
E5	229	Connection rejection - quality of service not available (transient condition).
E6	230	Connection rejection - quality of service not available (permanent condition).
E7	231	Connection rejection - NSAP unreachable (transient condition).
E8	232	Connection rejection - NSAP unreachable (permanent condition).
E9	233	Reset - reason unspecified.
EA	234	Reset - congestion.
EB	235	Connection rejection - NSAP address unknown (permanent condition).

SNA Diagnostic Codes

The following diagnostic codes are set in clear- and reset-indication packets, when Communications Server (SNA) is being used.

All diagnostic codes are not necessarily used by all DTEs, but those that are used have the meaning indicated.

The first diagnostic in each group is a general code that may be used in place of the more specific codes within the group.

These codes, set by transmitting DTEs in clear, reset, and restart packets that also have the cause code set to x'80' transferred on SNA-to-SNA connections, are normally delivered to the remote DTE in a corresponding indication packet by DCEs. However, DCEs may override DTE requests. In this event, DCEs place a network-generated nonzero cause code less than 128 in the cause field and insert the network diagnostic code in the diagnostic code field of the resulting indication packet delivered to the remote DTE.

List of CCITT SNA Diagnostic Codes		
Hex	**Dec**	**Meaning**
00	0	Normal initialization or termination.
0C	12	Incorrect LLC type.
10	16	Incorrect packet type (general).
11	17	Incorrect packet type for state r1.
12	18	Incorrect packet type for state r2.
13	19	Incorrect packet type for state r3.
14	20	Incorrect packet type for state p1.
15	21	Incorrect packet type for state p2.
16	22	Incorrect packet type for state p3.
17	23	Incorrect packet type for state p4.
18	24	Incorrect packet type for state p5.
19	25	Incorrect packet type for state p6.
1A	26	Incorrect packet type for state p7.

1B	27	Incorrect packet type for state d1.
1C	28	Incorrect packet type for state d2.
1D	29	Incorrect packet type for state d3.
20	32	DCE timer expired (general).
21	33	DCE timer expired: Incoming call.
22	34	DCE timer expired: Clear indication.
23	35	DCE timer expired: Reset indication.
24	36	DCE timer expired: Restart indication.
30	48	DTE timer expired: (general).
31	49	DTE timer expired: Call request.
32	50	DTE timer expired: Clear request.
33	51	DTE timer expired: Reset request.
34	52	DTE timer expired: Restart request.
40	64	Unassigned (general).
50	80	QLLC error: (general).
51	81	QLLC error: Undefined C-field.
52	82	QLLC error: Unexpected C-field.
53	83	QLLC error: Missing I-field.
54	84	QLLC error: Undefined I-field.
55	85	QLLC error: I-field too long.
56	86	QLLC error: Frame reject received.
57	87	QLLC error: Header incorrect.
58	88	QLLC error: Data received in wrong state.
59	89	QLLC error: Time-out condition.
5A	90	QLLC error: Number incorrect.
5B	91	QLLC error: Recovery rejected or ended.
5D	93	QLLC error: ELLC timeout condition.
60	96	PSH error (general).
61	97	PSH error: Sequence error.
62	98	PSH error: Header too short.
63	99	PSH error: PSH format incorrect.
64	100	PSH error: Command undefined.
65	101	PSH error: Protocol incorrect.
66	102	PSH error: Data received in wrong state.
69	105	PAD error: Timeout condition.
70	112	PAD error: (general).
71	113	PAD error: PAD access facility failure.
72	114	PAD error: SDLC FCS error.
73	115	PAD error: SDLC time-out.
74	116	PAD error: SDLC frame incorrect.
75	117	PAD error: I-field too long.
76	118	PAD error: SDLC sequence error.

77	119	PAD error: SDLC frame aborted.
78	120	PAD error: SDLC FRMR received.
79	121	PAD error: SDLC response incorrect.
7B	123	PAD error: Incorrect packet type.
7F	127	PAD error: PAD inoperable.
80	128	DTE-specific (general).
81	129	DTE-specific: 8100_DPPX-specific.
82	130	DTE-specific: INN_QLLC-specific.
83	131	DTE-specific: INN_QLLC-specific.
84	132	DTE-specific: INN_QLLC-specific.
85	133	DTE-specific: INN_QLLC-specific.
86	134	DTE-specific: INN_QLLC-specific.
87	135	DTE-specific: INN_QLLC-specific.
88	136	DTE-specific: INN_QLLC-specific.
89	137	DTE-specific: INN_QLLC-specific.
8A	138	DTE-specific: INN_QLLC-specific.
8B	139	DTE-specific: INN_QLLC-specific.
8C	140	DTE-specific: INN_QLLC-specific.
8D	141	DTE-specific: INN_QLLC-specific.
8E	142	DTE-specific: INN_QLLC-specific.
8F	143	DTE-specific: INN_QLLC-specific.
90	144	Network-specific.
91	145	Network-specific: DDX-P RNR packet received.
A0	160	Packet not allowed (general).
A1	161	Packet not allowed: Incorrect M-bit packet sequence.
A2	162	Packet not allowed: Incorrect packet type received.
A3	163	Packet not allowed: Incorrect packet on PVC.
A4	164	Packet not allowed: Unassigned LC.
A5	165	Packet not allowed: Diagnostic packet received.
A6	166	Packet not allowed: Packet too short.
A7	167	Packet not allowed: Packet too long.
A8	168	Packet not allowed: Incorrect GFI.
A9	169	Packet not allowed: Not identifiable.
AA	170	Packet not allowed: Not supported.
AB	171	Packet not allowed: Incorrect Ps.
AC	172	Packet not allowed: Incorrect Pr.
AD	173	Packet not allowed: Incorrect D-bit received.
AE	174	Packet not allowed: Incorrect Q-bit received.
AE	174	Packet not allowed: Incorrect Q-bit received.
B0	176	DTE-specific: (NPSI gate/date) (general).
B1	177	DTE-specific: No LU-to-LU session.
C0	192	DTE-specific: (general).

C1	193	DTE-specific: Termination pending.
C2	194	DTE-specific: Channel inoperative.
C3	195	DTE-specific: Unauthorized interrupt confirmation.
C4	196	DTE-specific: Unauthorized interrupt request.
C5	197	DTE-specific: PU (PVC) not available.
C6	198	DTE-specific: Inactivity timeout.
C7	199	DTE-specific: Incompatible line configuration.
D0	208	Resources: (general).
D1	209	Resources: Buffers depleted.
D2	210	Resources: PIU too long.
E0	224	Local procedure error: (general).
E1	225	Local procedure error: Packet with LC=0 not received.
E2	226	Local procedure error: Restart or diagnostic packet on LCI =x'000'.
E3	227	Local procedure error: Incoming call received on wrong LC.
E4	228	Local procedure error: Facility not subscribed.
E5	229	Local procedure error: Packet not restart or diagnostic on LCI = x'000'.
E6	230	Local procedure error: Facility parameters not supported.
E7	231	Local procedure error: Facility not supported.
E8	232	Local procedure error: Unexpected calling DTE.
E9	233	Local procedure error: Incorrect D-bit request.
EA	234	Local procedure error: Reset indication on virtual call.
EB	235	Local procedure error: Incorrect protocol identifier.
EC	236	Local procedure error: Connection identifier mismatch.
ED	237	Local procedure error: Missing cause or diagnostic code.

Logical Channel States

The following list describes the logical channel states for the CCITT-defined diagnostic codes.

State	Meaning
d1	Flow control ready
d2	DTE reset request
d3	DCE reset indication
p1	Channel ready
p2	DTE call request
p3	DCE incoming call
p4	Data transfer
p5	Call collision
p6	DTE clear request
p7	DCE clear indication
r1	Packet level ready
r2	DTE restart request
r3	DCE restart indication

Diagnostic Codes Used by the xtalk Command

The following diagnostic codes are set up by the **xtalk** program when clearing connections:

List of xtalk Diagnostic Codes		
Hex	**Dec**	**Meaning**
F1	241	Normal disconnection.
F4	244	Connection request rejected. This may occur if the program is busy (already connected to another program) or if the other program is not listening.

Appendix F. Supported Facilities for X.25 Communications

The X.25 program supports both standard X.25 facilities and CCITT-defined facilities.

Several types of facilities may be requested in a call packet. Standard X.25 facilities are the most usual but you may find also non-X.25 facilities specific to your network or CCITT-defined facilities to be used with the OSI network services. Ask your network provider which facilities are available.

Facilities Format

Nonstandard facilities are preceded by a facility marker: 0x0000, 0x00FF or 0x000F. The following diagram shows the format of these facility markers:

X.25 facilities
0x0
0x0
non-X.25 facilities provided by the local network
0x0
0xFF
non-X.25 facilities provided by the remote network
0x0
0x0F
CCITT-specified DTE facilities
Facilities Format

If any section is not required, both it and the preceding facility marker can be left out.

Within each section, the facilities format is defined as a series of facility codes, followed by a number of bytes of arguments. The number of bytes of arguments is defined by the first two bits of the facility code as shown in the following table.

0	Class						

The class can have one of the following values:

00 Class A. This has a single-byte parameter field.
01 Class B. This has two bytes as a parameter.
10 Class C. This has three bytes as a parameter.
11 Class D. The next byte defines the length of the parameter.

One special facility code, 0xFF, is reserved for extension of the facility codes. The octet following this code indicates an extended facility code having the format A, B, C or D class. Repetition of the facility code 0xFF is permitted, resulting in additional extensions.

X.25 Facilities

The X.25 standard facilities include:

Value	Function	Parameter Length
01	Reverse charging and fast select	1
02	Throughput class	1
03	Closed user group selection	1
04	Charging information request	1
08	Called line address modified notification	1
09	CUG with outgoing access	1
0A	Quality of Service Negotiation - minimum throughput class	1
0B	Expedited Data Negotiation	1
41	Bilateral closed user group selection	2
42	Packet size selection	2
43	Window size selection	2
44	Recognized Private Operating Agency (RPOA) selection (basic format)	2
49	Transit delay selection and indication	2
C1	Charging (call duration)	variable
C2	Charging (segment count)	variable
C3	Call redirection notification	variable
C4	Recognized Private Operating Agency (RPOA) selection (extended format)	variable
C5	Charging (monetary unit)	variable
C6	Network User identification (NUI)	variable
C9	Called Address Extension (OSI)	variable
CA	Quality of Service Negotiation - End to end transit delay	variable
CB	Calling Address Extension (OSI)	variable

Packet Size Selection

The packet size selection facility has a parameter length of 2 bytes. The following table shows the call packet coding.

	Packet Size Selection Call Packet	
0	0x42	
1	Reserved	Transmit packet size
2	Reserved	Receive packet size

where:

0x42 Specifies packet size selection.

Transmit packet size Indicates the requested size for packets transmitted from the calling DTE. Valid values are:

0x04 16 octets

0x05 32 octets

0x06 64 octets

0x07 128 octets

0x08 256 octets

0x09 512 octets

0x0A 1024 octets

0x0B 2048 octets

0x0C 4096 octets

Receive packet size Requested size for packets transmitted from the called DTE. Valid values are the same as those for the transmitted packet size.

Window Size Selection

The window size selection facility has a parameter length of 2 bytes. The following table shows the call packet coding.

	Window Size Selection Call Packet	
0	0x43	
1	Reserved	Transmission window size
2	Reserved	Receive window size

where:

0x43 Specifies window size selection.

Transmit window size Specifies requested size for the window for packets transmitted by the calling DTE. This value represents the maximum number of packets that can be received without an acknowledgment. Values are in the range from 0x01 to 0x07 inclusive.

Received window size Specifies requested size for the window for packets to be transmitted by the called DTE. Values are in the range from 0x01 to 0x07 inclusive.

Throughput Class

The throughput class facility has a parameter length of 1 byte. The following table shows the call packet coding.

	Throughput Class Call Packet	
0	0x02	
1	Outgoing Throughput class	Incoming Throughput Class

where:

0x02 Specifies the throughput class required facility.

Outgoing throughput Specifies throughput class requested for data to be sent by the calling DTE. Valid values are:

0x07 1200 bits

0x08 2400 bits

0x09 4800 bits

0x0A 9600 bits

0x0B 19200 bits

0x0C 48000 bits

Incoming throughput Specifies throughput class request for data sent from the called DTE. Supported values are the same as those for the outgoing throughput class.

Closed User Group (CUG) Selection

The closed user group (CUG) selection facility has a parameter length of 1 byte. The following table shows the call packet coding for both the basic and extended formats.

CUG Selection Call Packets

	Basic Format	
0	0x03	
1	First BCD digit of CUG	Second BCD digit of CUG

Extended Format

	Extended Format	
0	0x47	
1	First BCD digit of CUG	Second BCD digit of CUG
2	Third BCD digit of CUG	Fourth BCD digit of CUG

where:

0x03 Specifies CUG selection required (basic format).
0x47 Specifies CUG selection required (extended format).
CUG Specifies the value of a CUG as follows:

1 to 99 Basic format

1 to 9999
Extended format

CUG with Outgoing Access

The CUG with outgoing access facility has a parameter length of 1 byte for the basic format and 2 bytes for the extended format. The following table shows the call packet coding for both the basic and extended formats.

CUG with Outgoing Access Call Packets

	Basic Format	
0	0x09	
1	First BCD digit of CUG	Second BCD digit of CUG

Extended Format

	Extended Format	
0	0x48	
1	First BCD digit of CUG	Second BCD digit of CUG
2	Third BCD digit of CUG	Fourth BCD digit of CUG

where:

0x09 Specifies CUG with outgoing access (basic format)
0x48 Specifies CUG with outgoing access (extended format)
CUG Specifies the value of a CUG as follows:

1 to 99 Basic format

1 to 9999
Extended format

Bilateral Closed User Group Selection

The bilateral CUG selection facility has a parameter length of 2 bytes. The following table shows the call packet coding.

	Bilateral CUG Selection Call Packet	
0	0x41	
1	First BCD digit of CUG	Second BCD digit of CUG
2	Third BCD digit of CUG	Fourth BCD digit of CUG

where

0x41 Specifies the bilateral CUG selection required facility.
CUG Indicates the value of a CUG. Valid values are **1** to **9999**.

Reverse Charging and Fast Select

The reverse charging and fast select facility has a parameter length of 1 byte. The following table shows the call packet coding.

		Reverse Charging and Fast Select Call Packet					
0	0x01						
1	A						B

where:

0x01 Specifies the fast select facility.

A Specifies whether a restricted response is required when fast select is also requested. Valid values are:

00 Indicates fast select not selected.

01 Indicates fast select selected.

10 Indicates fast select requested with no restriction on response.

11 Indicates fast select requested with restriction on response.

B Specifies reverse charge required. Valid values are:

0 No reverse charging requested.

1 Reverse charging requested.

Network User Identification (NUI)

The network user identification facility has a variable parameter length. The following table shows the call packet coding.

	NUI Call Packet
0	0xC6
1	Length of NUI
2	NUI data
*	

where:

0xC6 Specifies the network user identification facility.

Length of NUI Indicates the number of bytes given in network user identification data

NUI data Indicates network user identification data in the format identified by the network administrator.

Charging Information Request

The charging information request facility has a parameter length of 1 byte. The following table shows the call packet coding.

	Charging Information Request Call Packet							
0	0x04							
1								A

where:

0x04 Specifies the charging information request facility. Valid values are:

A Specifies the requesting service value. Valid values are:

0 Indicates charging information not requested.

1 Indicates charging information requested.

Charging (Monetary Unit)

The charging (monetary unit) facility has a variable parameter length. The following table shows the call packet coding.

	Charging (Monetary Unit) Call Packet
0	0xC5
1	Length of charging information
2	Charging identification
*	

where:

0xC5 Specifies the charging information (monetary unit) facility.
Length of charging information Specifies the length of the charging information in bytes.
Charging identification Specifies monetary unit charging information.

Charging (Segment Count)

The charging (segment count) facility has a variable parameter length. The following table shows the call packet coding.

	Charging (Segment Count) Call Packet
0	0xC2
1	Length of charging information
2	Charging Identification
*	

where:

0xC2 Specifies the charging information (segment count) facility.
Length of charging information Specifies the length of the charging information in bytes.
Charging identification Specifies segment count charging information.

Charging (Call Duration)

The charging (call duration) facility has a variable parameter length. The following table shows the call packet coding.

	Charging (Call Duration) Call Packet
0	0xC1
1	Length of charging information
2	Charging identification
*	

where:

0xC1 Specifies the charging information (call duration) facility.
Length of charging information Specifies the length of the charging information in bytes.
Charging identification Specifies call duration charging information.

Recognized Private Operating Agency (RPOA) Selection

The RPOA selection facility has a parameter length of 1 byte for the basic format and a variable length parameter for the extended format. The following table shows the call packet coding for both the basic and extended formats.

RPOA Selection Call Packets

	Basic Format	
0	0x44	
1	First BCD digit of RPOA	Second BCD digit of RPOA
2	Third BCD digit of RPOA	Fourth BCD digit of RPOA

Extended Format

	Extended Format	
0	0xC4	
1	Length of RPOA information	
2	First BCD digit of RPOA #1	Second BCD digit of RPOA #1
3	Third BCD digit of RPOA #1	Fourth BCD digit of RPOA #1
*	First BCD digit of RPOA #n	Second BCD digit of RPOA #n
*	Third BCD digit of RPOA #n	Fourth BCD digit of RPOA #n

where:

0x44 Specifies the RPOA selection required facility (basic format).
0xC4 Specifies the RPOA selection required facility (extended format).
RPOA Specifies the requested RPOA transit network identification code. Valid values are **1** to **9999**.
Length of RPOA information Specifies the length in bytes of the RPOA information in the facility.

Called Line Address Modified Notification

The called line address modified notification facility has a parameter length of 1 byte. The following table shows the call packet coding when the DCE originates the redirection.

		DCE-Redirected Call Packet						
0	0x08							
1	0							A

where:

0x08 Specifies the called line address modified notification facility.
A Specifies one of the following values:

- **0x7** Call distribution within a hunt group
- **0x1** Call redirection due to original DTE busy
- **0x9** Call redirection due to original DTE out of order
- **0x0F** Call redirection due to prior request from originally called DTE for systematic redirection

The following table shows the call packet coding when the DTE originates the redirection.

		DTE-Redirected Call Packet
0	0x08	
1	1	B

where:

0x08 Specifies the called line address modified notification facility.
B Indicates a reason for the redirection. This value is passed from the remote DTE.

Call Redirection Notification

The call redirection notification facility has a variable parameter length. The following table shows the call packet coding.

	Call Redirection Notification Call Packet	
0	0xC3	
1	Length redirection information	
2	Call redirection reason	
3		Length of called address
4	Called address (BCD)	
*		

where:

0xC3 Specifies the call redirection notification facility.

Length of redirection information Specifies the length in bytes of the call redirection information in the facility.

Call redirection reason Specifies the reason for call redirection. Valid values are:

- **0x1** Call redirection due to original DTE busy
- **0x9** Call redirection due to original DTE out of order
- **0x0F** Call redirection due to prior request from originally called DTE for systematic redirection

Called address Specifies the original called DTE address coded in BCD.

Transit Delay Selection and Indication

The transit delay selection and indication facility has a parameter length of 2 bytes. The following table shows the call packet coding.

	Transit Delay Selection and Indication Call Packet
0	0x49
1	Transit delay (in milliseconds, binary, high byte first)
2	Transit delay (in milliseconds, binary, high byte first)

where:

0x49 Specifies the transit delay selection and notification facility.
Transit delay Specifies the transit delay in milliseconds, coded in binary, high byte first.

Calling Address Extension

The calling address extension has a variable parameter length. The following table shows the call packet coding.

	Calling Address Extension	
0	0xCB	
1	Number of bytes following	
2	Use	Length of calling extension address
3	Calling extension address (BCD)	

where:

0xCB Specifies the calling address extension.
Use May have one of the following values:

- **00** To carry an entire calling OSI NSAP address
- **01** To carry a partial calling OSI NSAP address
- **10** To carry a non-OSI calling address
- **11** Reserved

Calling extension address Specifies up to 40 decimal digits coded in BCD containing the calling address extension.

Called Address Extension

0	0xC9	
1	Number of bytes following	
2	Use	Length of called extension address
3	Called extension address (BCD)	

0xC9 Specifies the called address extension.
Use May have one of the following values:

- **00** To carry an entire calling OSI NSAP address
- **01** To carry a partial calling OSI NSAP address
- **10** To carry a non-OSI calling address
- **11** Reserved

Called address extension Specifies up to 40 decimal digits containing the called address extension coded in BCD.

Quality of Service Negotiation - Minimum Throughput Class

0	0xCA	
2	Calling minimum throughput	Called minimum throughput

0x0A Specifies the Quality of Service Negotiation - minimum throughput class.

Calling minimum throughput Specifies the throughput class requested for data to be sent by the calling DTE. Supported values are:

- **0x07** 1200 bit/s
- **0x08** 2400 bit/s
- **0x09** 4800 bit/s
- **0x0A** 9600 bit/s
- **0x0B** 19200 bit/s
- **0x0C** 48000 bit/s

Called minimum throughput Specifies throughput class request for data sent from the called DTE. Supported values are the same as for the calling minimum throughput class.

Quality of Service Negotiation - End-to-End Transmit Delay

0	0xCA
1	Length of the following area
2	Cumulative transit delay in milliseconds (in binary, high byte first)
3	Cumulative transit delay in milliseconds (in binary, high byte first)
4	Requested end-to-end delay in milliseconds (in binary, high byte first)
5	Requested end-to-end delay in milliseconds (in binary, high byte first)
6	Maximum acceptable transit delay in milliseconds (in binary, high byte first)
7	Maximum acceptable transit delay in milliseconds (in binary, high byte first)

0xCA Quality of Service Negotiation - End-to-end transit delay.

Length Specifies the number of values in the stream. This can be one of 1, 2 or 3, as the requested end-to-end delay and maximum acceptable transit delay are optional.

End-to-end delay Specifies cumulative, requested end-to-end and maximum acceptable transit delays.

Expedited Data Negotiation

0	0x0B							
1								A

0x0B Expedited Data Negotiation

A Can be one of:

- **0** Specifies no use of expedited data.
- **1** Specifies the use of expedited data.

CCITT-Specified Facilities to Support the OSI Network

The CCITT-specified facilities that support the OSI network include:

- Calling Address Extension.
- Called Address Extension.
- Minimum Throughput Class.

- End-to-End Transmit Delay Facility.
- Expedited Data Negotiation.

Calling Address Extension

The calling address extension facility has a variable parameter length. The following table shows the call packet coding.

0	0xCB	
1	Number of bytes following	
2	Use	Length of calling extension address
3	Calling extension address (BCD)	
*		

where:

0xCB Specifies the calling address extension facility.

Use Indicates the usage for the calling address extension. Valid values are:

- **00** Carry an entire calling OSI NSAP address.
- **01** Carry a partial calling OSI NSAP address.
- **10** Carry a non-OSI calling address.
- **11** Reserved.

Length of calling extension address Specifies the length of the calling extension address in bytes.

Calling extension address Specifies the calling address extension. The value can be up to 40 decimal digits, coded in BCD.

Called Address Extension

The called address extension facility has a variable parameter length. The following table shows the call packet coding.

0	0xC9	
1	Number of bytes following	
2	Use	Length of called extension address
3	Called extension address (BCD)	
*		

where:

0xC9 Specifies the called address extension facility.

Use Indicates the usage for the called address extension. Valid values are:

- **00** Carry an entire called OSI NSAP address.
- **01** Carry a partial called OSI NSAP address.
- **10** Carry a non-OSI called address.
- **11** Reserved.

Length of calling extension address Specifies the length of the called extension address in bytes.

Calling extension address Specifies the called address extension. The value can be up to 40 decimal digits, coded in BCD.

Minimum Throughput Class

The minimum throughput facility has a parameter length of 1 byte. The following table shows the call packet coding.

0	0x0A	
1	Calling minimum throughput	Called minimum throughput

where:

0x0A Specifies the minimum throughput class facility for quality of service negotiation.

Calling minimum throughput Specifies the throughput class requested for data to be sent by the calling DTE. Valid values are:

0x07 1200 bits

0x08 2400 bits

0x09 4800 bits

0x0A 9600 bits

0x0B 19200 bits

0x0C 48000 bits

Called minimum throughput Specifies the throughput class request for data sent from the called DTE. Valid values are the same as those for the calling minimum throughput class.

End-to-End Transmit Delay Facility

The end-to-end transmit delay facility has a variable parameter length. The following table shows the call packet coding.

0	0xCA
1	Length (of the following area)
2	Cumulative (transit delay in milliseconds, binary, high byte first)
3	Cumulative (transit delay in milliseconds, binary, high byte first)
4	Requested end-to-end (delay in milliseconds, binary, high byte first)
5	Requested end-to-end (delay in milliseconds, binary, high byte first)
6	Maximum acceptable (transit delay in milliseconds, binary, high byte first)
7	Maximum acceptable (transit delay in milliseconds, binary, high byte first)

where:

0xCA Specifies the end-to-end transit delay facility for quality of service negotiation.

Length Specifies he number of values in the stream. Valid values are **1**, **2**, or **3**, since requested end-to-end delay and maximum acceptable transit delay are optional.

Cumulative Specifies the cumulative transit delay in milliseconds, coded in binary, high byte first.

Requested end-to-end Specifies requested end-to-end delay in milliseconds, coded in binary, high byte first.

Maximum acceptable Specifies the maximum acceptable transit delay in milliseconds, coded in binary, high byte first.

Expedited Data Negotiation

The expedited data negotiation facility has a parameter length of 1 byte. The following table shows the call packet coding.

0	0x0B							
1								A

where

0x0B Specifies the expedited data negotiation facility.

A Specifies one of the following values:

- **0** No use of expedited data
- **1** Use of expedited data

Appendix G. Communications Server (SNA) Problem Determination

The following information, techniques and procedures have been reviewed for technical accuracy and applicability, but have not been tested in every possible environment or situation. Normal precautions should be taken in adopting these same techniques and procedures, because as product and system interfaces change, so would the use of this information.

When you experience a problem with Communications Server, certain information is necessary to investigate your problem. This appendix informs you of what information is available, how to create that information, and how to package that information to send to the support center.

The more information that is available when you report the problem, the more quickly the problem can be resolved. It is very important that all information be collected at the same time. Synchronize the clocks of all of the machines involved in the problem so that the time stamps of the events will be as close as possible.

When you report a problem, be prepared to send certain information to the support center. Send the information, in a single directory, to the support center by using one of the following methods:

- Copy the directory to a diskette in compressed **tar** format.
- Upload the directory, in binary format, to a mainframe and send it over the network.

The problem determination information uses the following terminology:

/tmp/pmr	Specifies the directory into which problem determination files are copied. This directory is the directory that would be sent to the support center.
ProfName	Specifies the SNA Services link station profile.

Information Required for Communications Server (SNA) Support for X.25

Supplying specific details to your support personnel helps them determine the problem more quickly.

Basic Information

The **/usr/bin/snagetpd** command creates a compressed **tar** file of all the SNA files needed for debugging. The compressed file is **pd.tar.Z** and is left in the current directory. Run **/usr/bin/snagetpd** as soon as possible after you encounter a problem.

> **Note:** You must log in as the root user to run the commands necessary to provide the appropriate information to the support center.

Problem Definition

First, and most important, is a clear definition of the problem. Include this definition in a file called **README** on the diskette. This file should also include a list of the files on the diskette and a brief description of those files. Be sure to answer the following questions in the problem description:

- What happened?
- Is the problem reproducible? If so, list the steps to re-create it.
- What were the exact error messages generated? Include messages from all machines involved (such as AS/400 or mainframes).
- Which release of the Communications Server for AIX program is active and committed?

- Which release of the operating system is active and committed?
- What fixes have been applied to the system?
- Has Communications Server for AIX ever worked? If so, what changes occurred before it stopped working?

Definitions

Always provide your CS/AIX definitions when you have a problem. To copy definitions to the **/tmp/pmr** directory, log in as the root user and type the following command:

```
cp  -pr  /etc/sna/*  /tmp/pmr
```

SNA Error Log

The SNA error log is used to keep track of SNA errors. This log is located in the **/var/sna** directory and is named **sna.err**. The **sna.err** log is always active. When it reaches a specified size (defaults to 10Megabytes), it is renamed to **bak.err** and a new **sna.err** log is created.

Run a test case to re-create the problem.

Copy the **sna.err** log that was current for the duration of the test to the **/tmp/pmr** directory:

```
cp  -pr  /var/sna/*err  /tmp/pmr
```

System Error Log

Before updating a system error log, ensure that it is cleared, so that all entries relate to the current problem. The **errclear** command deletes error log entries that are older than the number of days specified by the *Days* parameter. To delete all error log entries, specify a 0 value for the *Days* parameter. Type the following command:

```
errclear 0
```

Run a test case to reproduce the problem, then type the following command:

```
cp /var/adm/ras/errlog /tmp/pmr/errlog
```

SNA Link Station Trace

To turn on tracing for an X.25 link station, type the following command:

```
snaadmin add_dlc_trace
```

If you have multiple link stations or dlc types, tracing can be limited by using the optional **resource_name** and **resource_type** fields. For example:

```
snaadmin add_dlc_trace, resource_type=LS, resource_name=MYX25LS
```

The **add_dlc_trace** command can be run before the link station is started to capture the link start-up exchanges.

After the problem has been reproduced, turn off link station tracing using the following command:

```
snaadmin remove_dlc_trace
```

The raw trace data is stored in the **/var/sna/sna1.trc** and **/var/sna/sna2.trc** files. These files can be formatted with the **snatrcfmt** command:

```
snatrcfmt -D -f /var/sna/sna.trc
```

The **snatrcfmt** command writes the formatted output to the **snatrc.dmp** file. Copy the resulting link station trace file to **/tmp/pmr**:

```
cp  -pr  /var/sna/*trc  /tmp/pmr
```

LU0 Information

You may need to supply LU0 information to the support center.

LU0 Line Trace

The **lu0** command initiates the LU0 subsystem. The LU0 subsystem initiates and centralizes control of both the LU0 primary and secondary support servers. The servers' data paths are independent of each other. However, pass-through support provides for the logical coupling of the two servers.

The **lu0** command provides a common operator interface through the interactive commands. These commands allow you to manipulate the LU0 subsystem while it is running, and help to minimize system resource consumption. You can use the interactive commands to display status summaries, start a secondary server, stop a server, terminate all servers, and exit the program. The **lu0server** process may also be run in the background by entering the **lu0 &** command. While running the **lu0server** in the background, the **lu0sndmsg** command is used to send commands to the **lu0server**. A line trace facility for the lu0 subsystem may be initiated with the **lu0 -T** command or if the **lu0server** is started in the background, with the **lu0sndmsg -T** command. The trace facility records the first 20 bytes of the SNA PIU block for traffic in either direction.

Up to 15 characters of run data are saved in the file. Two files are created containing trace information for the primary server and the secondary server, respectively: **/var/lu0/LU0Prime** and **/var/lu0/LU0Sec**.

To create LU0 line trace files, do the following:

```
# sna - sna    /*start sna
# lu0 -T       /*start the lu0server process
or
# lu0 &        /*either in the foreground or background
# lu0sndmsg -T /*specifying the -T option for trace if *lu0 is background
```

Run a test case to reproduce the problem, then stop the **lu0server** process by entering **X** to the lu0 shell (foreground) or **kill -9** xxxxx (background), where xxxxx is the process ID. The process ID was returned from the **lu0 &** command. You can also obtain the process ID by entering **kill %1** if no other background processing has been started from the shell.

Event Tracing

The support organization may ask you to generate an event trace. All of the traces generate arguments to the **trace** command. The **trace** command uses hook IDs to determine what to trace. The support organization needs both a formatted and an unformatted copy of the event trace.

View the **/usr/include/sys/trchkid.h** file to see a list of all hook IDs.

For example, to create and capture event information for the X.25 packet driver do the following:

1. Start system trace using the hook ID for the X.25 packet driver (25C):

   ```
   trace -a -j 25c
   ```

2. Start SNA:

   ```
   sna -s sna
   ```

3. Run a test case to reproduce the problem.
4. Stop the system trace:

   ```
   trcstop
   ```

5. Format and save the trace report.

   ```
   trcrpt > /tmp/pmr/trcfile.fmt
   ```

6. Copy the unformatted trace file to **/tmp/pmr**.

   ```
   cp /var/adm/ras/trcfile/tmp/pmr
   ```

Additional Problem Determination Information for X.25

In addition to the information required for basic problem determination, you may be asked to supply additional information.

The support center may ask that additional CS/AIX traces be enabled and collected. For details, see the ″IBM Communications Server for AIX Diagnostics Guide Version 6″. In all cases, make sure that the **/usr/bin/snagetpd** command is run as soon as possible after the problem is detected.

System Error Log

To ensure that all the data in the system error log is relevant, clear the error log first.

Clearing the System Error Log

The **errclear** command deletes error log entries that are older than the number of days specified by the *Days* parameter. To delete all error log entries, specify a 0 value for the *Days* parameter.

To use the SMIT fast path, type:

```
smit errclear
```

Alternatively, you can type the following at the command line:

```
errclear 0
```

Showing the System Error Log

The **errpt** command generates an error report from entries in the system error log. The **errpt** command includes flags for limiting the report to events matching specified criteria. A concurrent error report flag is provided that formats and displays each error entry at the time the entry is logged.

The default report is a summary report consisting of a single line of data for each error entry. Error log entries display the most recent entries first.

The system error report may be viewed by using either SMIT or the **errpt** command. To use the SMIT fast path, type:

```
smit errpt
```

To use the **errpt** command, enter:

```
errpt -a | more
```

QLLC - Data Link Control

For suspected problems in the data link control (DLC) area, the following information may be helpful in debugging trace information provided by the **/dev/dlcqllc** device:

Starting a QLLC Trace	Type the **trace -a -j 227** command.
Stopping a QLLC Trace	Type the **trcstop** command.
Showing a QLLC Trace	Type the **trcrpt -d 227 -t /etc/trcfmt.x25 /var/sna/snaservice.X** command.

Note: The X.25 device driver and dlcqllc currently share the 227 tracehook. The **/etc/trcfmt.x25** formatter file is necessary to interpret and format the trace information.

Appendix H. X.25 Virtual License Information

The AIXlink/X.25 Version 2.0 product requires the virtual circuit license information.

To enter the X.25 license level, use the SMIT tool with the following path:

```
Software License Management

Manage X.25 Server License Database
```

A screen appears entitled Change/Show Number of X.25 Virtual Circuits.

In this screen, select one of the following license levels depending on the X.25 virtual circuit license purchased.

```
Basic (<17)
Extended (<65)
Advanced (<257)
Unrestricted
```

Once this information is entered, X.25 ports can be configured on the system.

Appendix I. Using AIXlink/X.25 over the IBM 2-Port Multiprotocol Adapter

This chapter describes the use of the AIXlink/X.25 product over the IBM 2-Port Multiprotocol adapter.

Overview

The AIXlink/X.25 product supports the use of the IBM 2-Port Multiprotocol **dpmp** adapter for X.25 communications. As compared to previous releases of the AIXlink/X.25 product, there are minor differences in configuration. However, it is important to note that these differences are primarily internal to the product and do not change the product's capabilities or usage.

The **dpmp** adapter is different from the adapters historically used by the AIXlink product, most notably because it has no internal CPU with its own resident operating system (microcode). The **dpmp** adapter is called a ″shallow″ adapter, whereas other adapters supported by the AIXlink product are called ″deep,″ since they have the capability to change their operation by loading and running a different microcode.

Also, the **dpmp** adapter driver has its own interface, different from the interface to the existing AIXlink/X.25 **twd** driver. The **dpmp** driver (called **hdlc**) presents a CDLI (non-STREAMS) interface and the **twd** driver uses a STREAMS interface.

The net effect of these differences is the following:

1. AIXlink/X.25 ports are configured over instances of either the **twd** driver or the **hdlc** driver, depending on which adapter type is available for use in the system.
2. The Frame layer of the X.25 product runs in the kernel when a port is configured over the **hdlc** driver. Standard kernel trace utilities provide diagnostics.
3. Applications which use the AIXlink DLPI interface to ports that are configured over the **hdlc** driver no longer require the **twd** driver as an intermediary. These user streams are linked directly to the frame layer running in the kernel.
4. The AIXlink/X.25 'sx25debug' utility, which interfaces with microcode running on a deep adapter, cannot be used on ports which are configured over the **hdlc** driver. The kernel trace is used for diagnosing frame layer problems when using the IBM 2–Port Multiprotocol Adapter.
5. **hdlc** driver utilities are available for diagnosing problems below the frame layer for AIXlink ports on **dpmp** adapters.

   Note: Item 3 above requires existing applications which use the AIXlink DLPI interface to ″open″ the DLPI devices differently. Note that this change is required only if the application needs to utilize AIXlink ports configured over **dpmp** adapters.

   The AIXlink/X.25 sample programs included with the product show how to perform the new ″open.″ The updated ″open″ method allows DLPI applications to communicate with all adapter types supported by the AIXlink/X.25 product.

Configuration Object Model

The relationships between objects of the AIXlink/X.25 product is different for ports using the **dpmp** driver, as shown in the following figure.

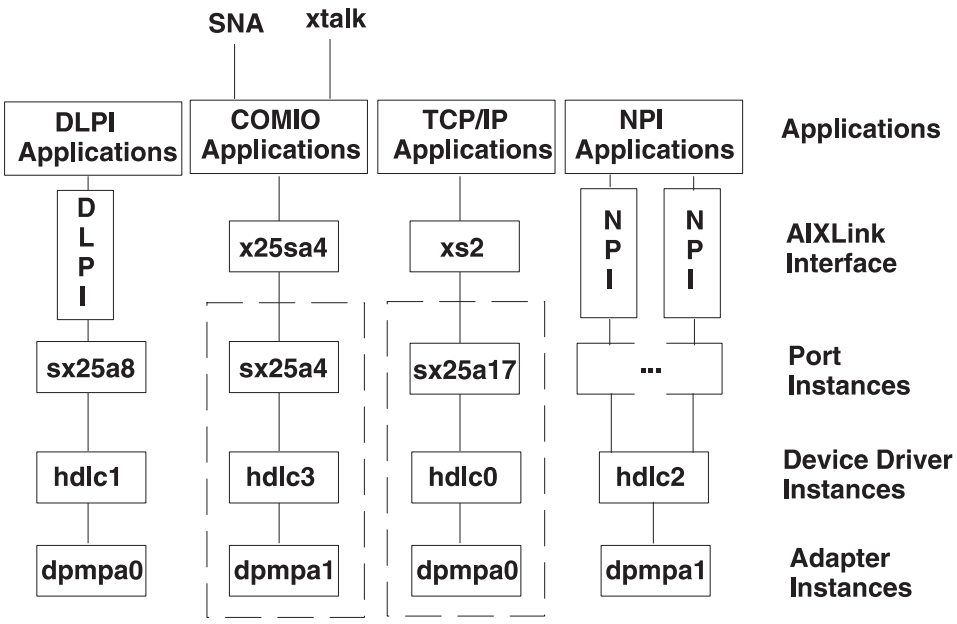


Figure 13. STREAMS Configuration

As shown in Figure 13, the overall structure of the different objects is similar, but different drivers and adapters are used. The figure shows the four different API interfaces available through AIXlink/X.25 and the different layers that form the stack from the API interface to the dpmp adapter. The layers within the stack are: Applications, AIXlink interface, Port Instances, Device Driver Instances, and Adapter Instances.

The first column depicts information for DLPI applications. Reading from the top of the stack to the bottom, the DLPI application corresponds to the Application layer, DLPI is the AIXlink Interface, sx25a8 is the Port Instance, hldc1 is the Device Driver Instance, and dpmpa0 is the Adapter Instance.

The second column depicts information for COMIO applications. The COMIO stack has additional interfaces on top of the Application layer. These interfaces are SNA and xtalk. This depicts the fact that SNA and xtalk run over the COMIO layer of AIXlink/X.25. Reading the rest of the stack from top to bottom: COMIO Applications correspond to the Application layer, x25s4 is the AIXlink Interface, sx25a4 is the Port Instance, hdlc3 is the Device Driver Instance, and dpmpa1 is the Adapter Instance.

The third column depicts information for TCP/IP applications. The TCP/IP applications correspond to the Application Layer, xs2 is the AIXlink Interface, sx25a17 is the Port Instance, hdlc0 is the Device Driver Instance, and dpmpa0 is the Adapter Instance.

The fourth column depicts information for NPI Applications. The NPI applications correspond to the Application Layer, NPI corresponds to the AIXlink Interface, hdlc2 is the Device Driver Instance, and dpmpa1 is the Adapter Instance. Because NPI is not concerned with the Port Instance, this information is blank.

The physical linking of STREAMS devices is also different for the AIXlink/X.25 ports using the **dpmp** driver and is illustrated in the following chart:

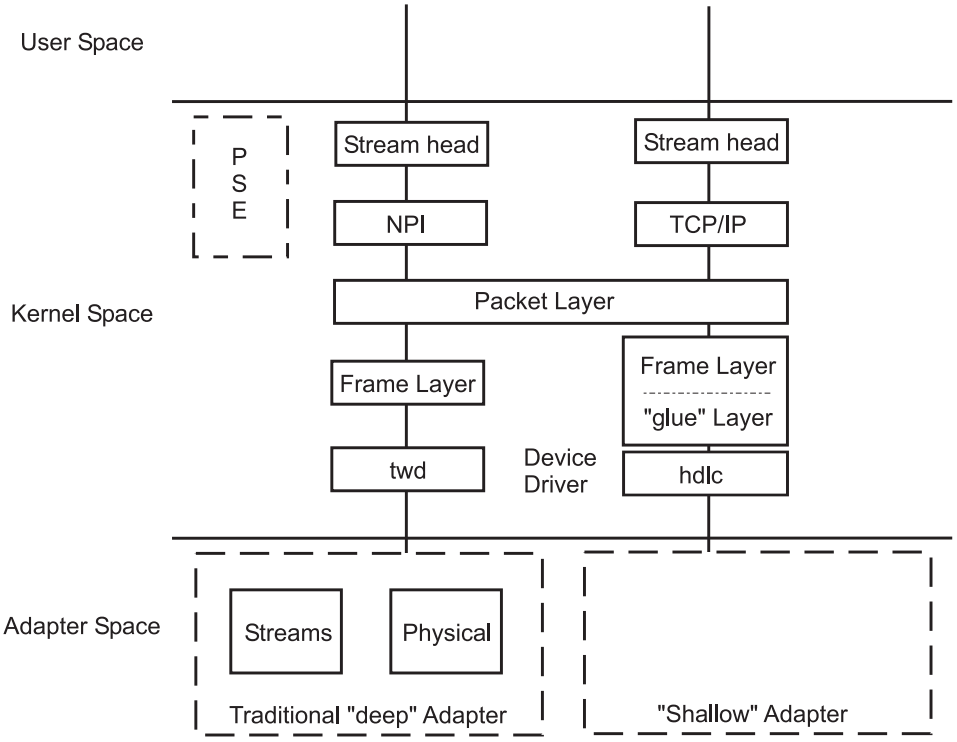


Figure 14. Linking STREAMS Devices

The chart displays the differences between deep adapters and a shallow adapters. The chart is divided horizontally between User Space (top), Kernel Space (middle), and Adapter Space (bottom) and shows the stack of layers for each type of adapter. PSE is the Portable Streams Environment.

The stack of layers for the deep adapter from top to bottom are:

- Directly available to the User Space are the Stream head, the application interface layer (in this case NPI), Packet Layer, the Frame layer, and the **twd** Device Driver.
- In Adapter Space is the adapter code consisting of Streams and the Physical Layer.

The stack of layers for the shallow adapter from top to bottom are:

- Directly available to the User Space are the Stream head, the application interface layer (in this case TCP/IP), the Packet Layer, the Frame and ″glue″ layers, and the **hdlc** Device Driver which contains the adapter code.
- The Adapter Space consists of hardware only, no software.

> **Note:** The ″glue″ layer shown in the diagram above presents the physical layer interface to the frame layer. It's purpose is to translate the CDLI interface presented by the **hdlc** driver into a STREAMS interface that the Frame layer requires.

The X.25 applications attached above the Stream head are shielded from these internal differences.

Appendix J. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 003
11400 Burnet Road
Austin, TX 78758-3498
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- AS/400
- CICS
- IBM
- IBM eServer pSeries
- IBM eServer zSeries
- Micro Channel
- Portmaster
- PS/2
- RS/6000
- System/390
- VTAM

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be the trademarks or service marks of others.

Index

Numerics

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

X

Readers' Comments — We'd Like to Hear from You

AIXlink/X.25 Version 2.0 for AIX: Guide and Reference

Publication No. SC23-2520-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

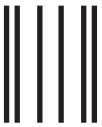
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC23-2520-04

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department H6DS-905-6C006
11501 Burnet Road
Austin, TX 78758-3493

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.

SC23-2520-04